ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

«Ανάπτυξη εργαστηριακού οδηγού αριθμητικών μεθόδων με την γλώσσα προγραμματισμού Python»

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κατσάνη Βασιλική

Επιβλέπων

Βαρσάμης Δημήτριος

Σέρρες – Φεβρουάριος 2023

Πρόλογος

Πριν την ανάπτυξη και παρουσίαση του εργαστηριακού οδηγού, αισθάνομαι την υποχρέωση να ευχαριστήσω από καρδιάς τον επιβλέπων καθηγητή μου κ. Βαρσάμη Δημήτριο πρωτίστως για την πολύτιμη βοήθεια του καθ' όλη την διάρκεια σπουδών μου, αλλά και για την σπουδαία βοήθεια του στην διεκπεραίωση της μεταπτυχιακής διπλωματικής μου εργασίας. Προσδοκώ να συνεργαστώ μαζί του ξανά.

Επίσης, θέλω να ευχαριστήσω τους ανθρώπους μου, την οικογένεια μου, τα ανίψια μου, τους φίλους μου για την ανοχή που έδειξαν όλο αυτό το διάστημα σπουδών μου για τις στιγμές που έπρεπε να απουσιάζω από δίπλα τους λόγω σπουδαστικών υποχρεώσεων.

Τέλος, θέλω να ευχαριστήσω έναν άνθρωπο που είναι δίπλα μου σε όλα, με στηρίζει σε όλες μου τις προσπάθειες και πιστεύει σε εμένα όσο κανένας άλλος.

Ευχή μου, ο θεός να έχει τον κάθε ένα από εσάς καλά και να σας ανταποδώσει όλα τα καλά που μου έχετε προσφέρει.

Περίληψη

Στην παρούσα μεταπτυχιακή διπλωματική εργασία έχει αναπτυχθεί ένας εργαστηριακός οδηγός του προπτυχιακού μαθήματος «Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός» του τμήματος «Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών» του ΔΙΠΑΕ Σερρών σε γλώσσα προγραμματισμού Python η οποία είναι Open Source.

Αρχικά, γίνεται μια αναφορά στις βασικές λειτουργίες της γλώσσας, στα χαρακτηριστικά της αλλά και στην δομή της. Αναφέρονται κάποιοι ορισμοί και παραδείγματα για την καλύτερη κατανόηση και εξοικείωση με τις εντολές της Python.

Εν συνεχεία, γίνεται μια σύγκριση με την γλώσσα προγραμματισμού Matlab, αφού είναι η γλώσσα προγραμματισμού που διδάσκεται αυτή την στιγμή το εργαστηριακό μάθημα «Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός».

Λέξεις κλειδιά: Αριθμητική Ανάλυση, Python

Πίνακας Περιεχομένων

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Εργαστηριακός οδηγός αριθμητική ανάλυση και Επιστημονικός Προγραμματισμός με τη γλώσσα προγραμματισμού Python

Στον εργαστηριακό οδηγό του μαθήματος «Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός» που υλοποιήθηκε για τις ανάγκες του προπτυχιακού προγράμματος σπουδών του τμήματος «Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών» του ΔΙΠΑΕ Σερρών έχει στόχο οι φοιτητές :

- να γνωρίσουν την γλώσσα προγραμματισμού Python
- να δημιουργούν γραφικές παραστάσεις αλλά και να μάθουν τις βασικές δομές του περιβάλλοντος της Python
- να κατανοήσουν την διαδικασία μετατροπής αριθμών σε αριθμητικά συστήματα, η οποία υλοποιείται προγραμματιστικά σε Python
- να εμπεδώσουν την έννοια της αριθμητικής επίλυσης εξισώσεων και συγκεκριμένα με την γενική επαναληπτική μέθοδο, η οποία υλοποιείται προγραμματιστικά σε Python
- να αντιληφθούν την έννοια της αριθμητικής επίλυσης εξισώσεων και ειδικότερα με τη μέθοδο Διχοτόμησης, η οποία υλοποιείται προγραμματιστικά σε Python
- να εξοικειωθούν με την έννοια της αριθμητικής επίλυσης εξισώσεων και ειδικότερα με την επαναληπτική μέθοδο Newton, η οποία υλοποιείται προγραμματιστικά σε Python
- να γνωρίσουν την έννοια της παρεμβολής και τις διάφορες μεθόδους υλοποίησης της μέσα από την Python
- να εμπεδώσουν την έννοια της αριθμητικής παραγώγισης και τις διάφορες μεθόδους υλοποίησης της μέσα από την Python
- να κατανοήσουν την έννοια της αριθμητικής ολοκλήρωσης και τις διάφορες μεθόδους υλοποίησης της αριθμητικής ολοκλήρωσης μέσα από την Python

Περιεχόμενα εργαστηριακού οδηγού Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός με τη γλώσσα προγραμματισμού Python

Ο εργαστηριακός οδηγός αποτελείται από δώδεκα (12) ενότητες. Παρακάτω παρουσιάζονται αναλυτικά οι στόχοι κάθε ενότητας.

Ενότητα 1: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την γλώσσα προγραμματισμού Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Εισαγωγή στην Python
    - Τελεστές στην Python
    - Σταθερές τιμές στην Python
    - Συναρτήσεις στην Python
    - Εντολές ελέγχου στην Python
2. Πίνακες
3. Μαθηματικές Συναρτήσεις

Ενότητα 2: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την γλώσσα προγραμματισμού Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Python file
    - Δημιουργία file
    - Εντολές σε Shell
2. Python δημιουργία *.py - Παράδειγμα
    - Δημιουργία *.py
3. Python Συνάρτηση
    - Δημιουργία συνάρτησης

Ενότητα 3: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την δημιουργία γραφικών παραστάσεων και την χρήση βασικών δομών στο περιβάλλον της Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Γραφικές παραστάσεις
    - Γραφική παράσταση συνάρτησης
    - Γραφική παράσταση σημείων
2. Βασικές δομές
    - Παράδειγμα (for)
    - Παράδειγμα (while)

Ενότητα 4: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με τη διαδικασία μετατροπής αριθμών σε αριθμητικά συστήματα, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητικά συστήματα – Μετατροπές
    - Μετατροπή ακέραιου μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα
    - Μετατροπή κλασματικού μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα
    - Παραδείγματα
    - Ασκήσεις

Ενότητα 5: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με τη διαδικασία μετατροπής αριθμών σε αριθμητικά συστήματα, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητικά συστήματα – Μετατροπές
    - Μετατροπή ακέραιου μέρους αριθμού x από δεκαδικό σύστημα σε αριθμητικό σύστημα με βάση b
    - Μετατροπή κλασματικού μέρους αριθμού x από δεκαδικό σύστημα σε αριθμητικό σύστημα με βάση b
    - Παραδείγματα
    - Ασκήσεις

Ενότητα 6: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής επίλυσης εξισώσεων και συγκεκριμένα με την γενική επαναληπτική μέθοδο, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική επίλυση εξισώσεων
    - Επαναληπτική μέθοδος
    - Ακρίβεια δεκαδικών ψηφίων
    - Ακρίβεια σημαντικών ψηφίων
    - Επαναληπτική μέθοδος με κριτήριο τερματισμού
    - Ασκήσεις

Ενότητα 7:Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής επίλυσης εξισώσεων και ειδικότερα με τη μέθοδο Διχοτόμησης, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική επίλυση εξισώσεων
    - Μέθοδος διχοτόμησης
    - Μέθοδος διχοτόμησης – αλγόριθμος
    - Μέθοδος διχοτόμησης – υλοποίηση
    - Μέθοδος διχοτόμησης – βελτίωση
    - Μέθοδος διχοτόμησης – παραδείγματα
    - Πλήθος επαναλήψεων – ακρίβεια
    - Ασκήσεις

Ενότητα 8: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής επίλυσης εξισώσεων και ειδικότερα με την επαναληπτική μέθοδο Newton, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική επίλυση εξισώσεων
    - Μέθοδος Newton
    - Μέθοδος Newton – Αλγόριθμος
    - Μέθοδος Newton - Υλοποίηση
    - Μέθοδος Newton - Παραδείγματα
    - Ασκήσεις

Ενότητα 9: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της παρεμβολής και τις διάφορες μεθόδους υλοποίησης της μέσα από την Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Πολυωνυμική παρεμβολή
    - Γενική μέθοδος
    - Παρεμβολή – Συνάρτηση interp1d
    - Παρεμβολή – Συνάρτηση np.polyfit(x,y,n)
2. Polynomial Functions

Ενότητα 10: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής παραγώγισης και τις διάφορες μεθόδους υλοποίησης της μέσα από την Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική παραγώγιση
    - Διαφορές
    - Πρώτη παράγωγος με προς τα εμπρός διαφορές
    - Πρώτη παράγωγος με προς τα πίσω διαφορές

Ενότητα 11: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής ολοκλήρωσης και τις διάφορες μεθόδους υλοποίησης της αριθμητικής ολοκλήρωσης μέσα από την Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική ολοκλήρωση
2. Κανόνας τραπεζίου
    - Κανόνας τραπεζίου – υλοποίηση
    - Κανόνας τραπεζίου - παραδείγματα
3. Κανόνας Simpson
    - Κανόνας Simpson - υλοποίηση
4. Συνδυαστική Άσκηση

Ενότητα 12: Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής ολοκλήρωσης και τις διάφορες μεθόδους υλοποίησης της μέσα από την Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική ολοκλήρωση
2. Κανόνας ορθογωνίου
    - Κανόνας ορθογωνίου - παραδείγματα
3. Κανόνας τραπεζίου
    - Κανόνας τραπεζίου - παραδείγματα
4. Κανόνας Simpson
    - Κανόνας Simpson - παραδείγματα
5. Συνδυαστική άσκηση

ΚΕΦΑΛΑΙΟ 2

Python

Εισαγωγή στην Python

Η Python είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου που σκοπό έχει την άμεση κατανόηση του κώδικα με κύριο πλεονέκτημα την σύνταξη της. Με την Python μπορεί κανείς να συντάξει κώδικα απλούστερα και σε λιγότερες γραμμές από ότι αν έγραφε τον ίδιο κώδικα σε κάποια άλλη γλώσσα προγραμματισμού, όπως για παράδειγμα στην C++ και στην Java. Βασικό της προτέρημα είναι ότι διαθέτει μεγάλο πλήθος βιβλιοθηκών, οι οποίες είναι εύκολες στην εκμάθηση τους αλλά και διευκολύνουν την χρήση της Python.

Ωστόσο, η Python δεν περιορίζεται σε ένα είδος προγραμματισμού αλλά μπορεί κανείς να την χρησιμοποιήσει για διάφορα είδη προγραμματισμού, όπως για παράδειγμα τον συναρτησιακό και αντικειμενοστραφή προγραμματισμό. Παρατηρείτε ότι το μεγαλύτερο μέρος που γίνεται ο έλεγχος των τύπων της, εκτελείτε στον χρόνο εκτέλεσης αντί στον χρόνο μεταγλώττισης, έτσι διαπιστώνεται ότι χρησιμοποιεί δυναμικούς τύπους. Για παράδειγμα, ο έλεγχος αν κάτι είναι αλφαριθμητικό ή ακέραιος πραγματοποιείται όταν η μεταβλητή που περιέχει το αντίστοιχο αντικείμενο έχει πάρει συγκεκριμένη τιμή. Μια μεταβλητή μπορεί να αναφέρετε σε τιμές κάθε είδους, επειδή οι μεταβλητές δεν έχουν τιμές αλλά τύπους. Κατά την συγγραφή κώδικα υπάρχει αυτόματη διαχείριση μνήμης με αποτέλεσμα ο χρήστης να έχει λιγότερες ανησυχίες και μεγαλύτερη ευελιξία.

Μπορεί να εγκατασταθεί σε πολλά λειτουργικά συστήματα αλλά και με την χρήση τρίτων εργαλείων, όπως είναι το Cx_freeze και το Py2exe μπορεί να εκτελεστεί κώδικας Python, χωρίς την εγκατάσταση πρόσθετων διερμηνευτών, γιατί τα εργαλεία αυτά μετατρέπουν τον κώδικα της Python σε εκτελέσιμα αρχεία .exe και .msi .

Είναι μια απλουστευμένη γλώσσα προγραμματισμού διότι χρησιμοποιεί απλές και καθημερινές λέξεις (στα Αγγλικά) για την λειτουργία της. Για παράδειγμα, η εντολή «Εκτύπωσε» αντιστοιχεί στην δεσμευμένη λέξη «print». Αντίθετα με τις άλλες γλώσσες προγραμματισμού, οι οποίες χρησιμοποιούν αγκύλες για τον διαχωρισμό των συντακτικών δομών του προγράμματος, η Python χρησιμοποιεί τα κενά διαστήματα για αυτό τον διαχωρισμό.

Εισαγωγή στην βιβλιοθήκη numpy

Η Numpy είναι μια βιβλιοθήκη της Python για μαθηματικούς υπολογισμούς. Αποτελεί βασική υποδομή για επιστημονικές εφαρμογές καθώς υποστηρίζει εύκολα και πλήρως πολυδιάστατους πίνακες και συναρτήσεις γραμμικής άλγεβρας. Η Numpy διευκολύνει τον χρήστη στο να κάνει υπολογισμούς με πίνακες, να διαβάζει και να γράφει σύνολα δεδομένων και να συνεργάζεται με άλλες γλώσσες προγραμματισμού όπως η C και η C++.

Εισαγωγή στην βιβλιοθήκη matplotlib

Η Matplotlib είναι μια βιβλιοθήκη της Python για απεικόνιση. Επιτρέπει να πραγματοποιούνται εύκολα γραφήματα γραμμών, διαγράμματα πίτας, ιστογράμματα και άλλες επαγγελματικές απεικονίσεις.

Εισαγωγή στην βιβλιοθήκη Scipy

Η βιβλιοθήκη Spicy είναι μια βιβλιοθήκη που εξαρτάται από την numpy και έχει κατασκευαστεί για να λειτουργεί με τους αριθμητικούς πίνακες numpy. Παρέχει πολλές και αποτελεσματικές αριθμητικές ρουτίνες φιλικές προς το χρήστη, όπως ρουτίνες για αριθμητική ενσωμάτωση και βελτιστοποίηση. Τέλος, η βιβλιοθήκη Spicy διαθέτει ενότητες βελτιστοποίησης, γραμμικής άλγεβρας, ολοκλήρωσης και άλλων κοινών καθηκόντων στην επιστήμη των δεδομένων.

Εγχειρίδια Python

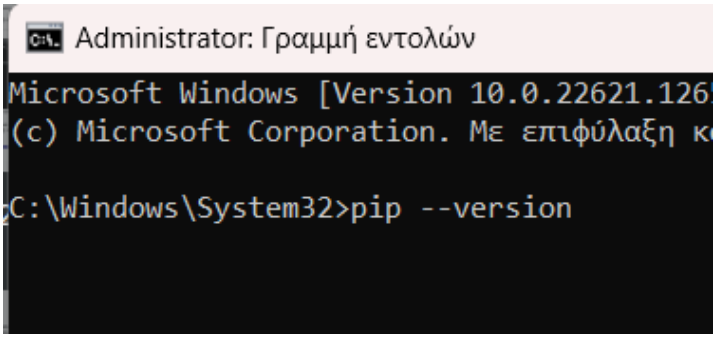
Ηλεκτρονικά εγχειρίδια για την εκμάθηση της Python υπάρχουν διαθέσιμα στην Ελληνική Κοινότητα Προγραμματιστών Python στην ιστοσελίδα http://python.org.gr/index.php/files/file/9-python , όπου βρίσκεται ο Οδηγός Εκμάθησης Python Βήμα Βήμα και στον σύνδεσμο https://drive.google.com/file/d/0B6ICSe14eQI1RFZZUTZsLXlJQ1E/view?resourcekey=0-pw_RVqSSYRDNNjn4jE3BWA , όπου βρίσκεται το εγχειρίδιο A byte of Python.

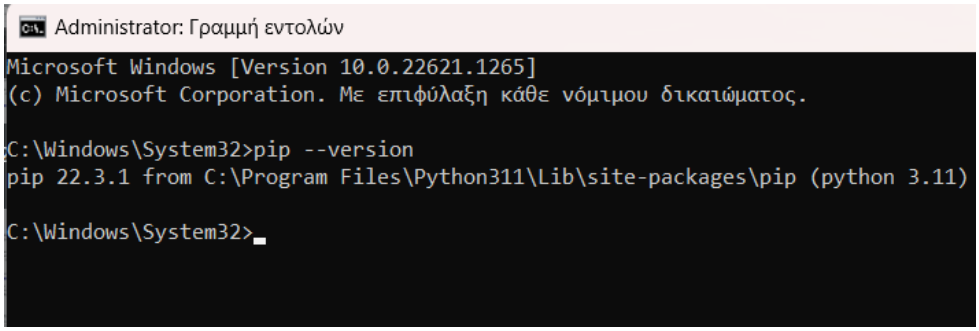
Εγκατάσταση της Python

Στην επίσημη ιστοσελίδα της Python (https://www.python.org/) υπάρχει διαθέσιμη και ελεύθερη για εγκατάσταση η Python σύμφωνα με τα χαρακτηριστικά του υπολογιστή που θα εγκατασταθεί. Αφού εγκατασταθεί η Python μπορεί να χρησιμοποιηθεί η εφαρμογή Python (Command Line) και το IDLE της Python. Ωστόσο, για την συγγραφή προγραμμάτων μπορεί να χρησιμοποιηθεί ο ενσωματωμένος κειμενογράφος του IDLE της Python αλλά και το λογισμικό ανοικτού κώδικα Notepad++ .

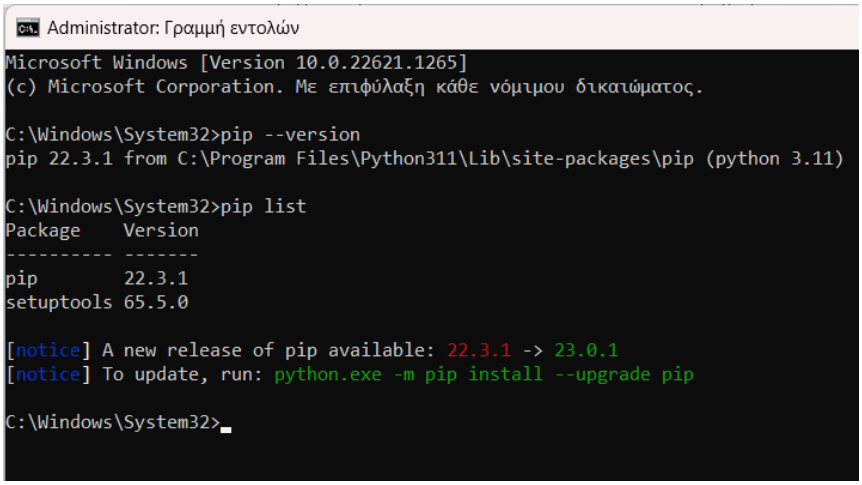
Εγκατάσταση βιβλιοθήκης numpy

Η numpy δεν είναι προ-εγκατεστημένη για αυτό και πρέπει να εγκατασταθεί πριν χρησιμοποιηθεί. Η εγκατάσταση είναι απλή και γίνεται αυτόματα με την χρήση της εντολής pip. Ανοίγουμε την Γραμμή εντολών και την εκτελούμε με δικαιώματα διαχειριστή. Στο παράθυρο εντολών πληκτρολογούμε την εντολή pip –version και πατάμε enter όπως φαίνεται και στην εικόνα παρακάτω, για να ενημερωθούμε σχετικά με την θέση και έκδοση του προγράμματος.

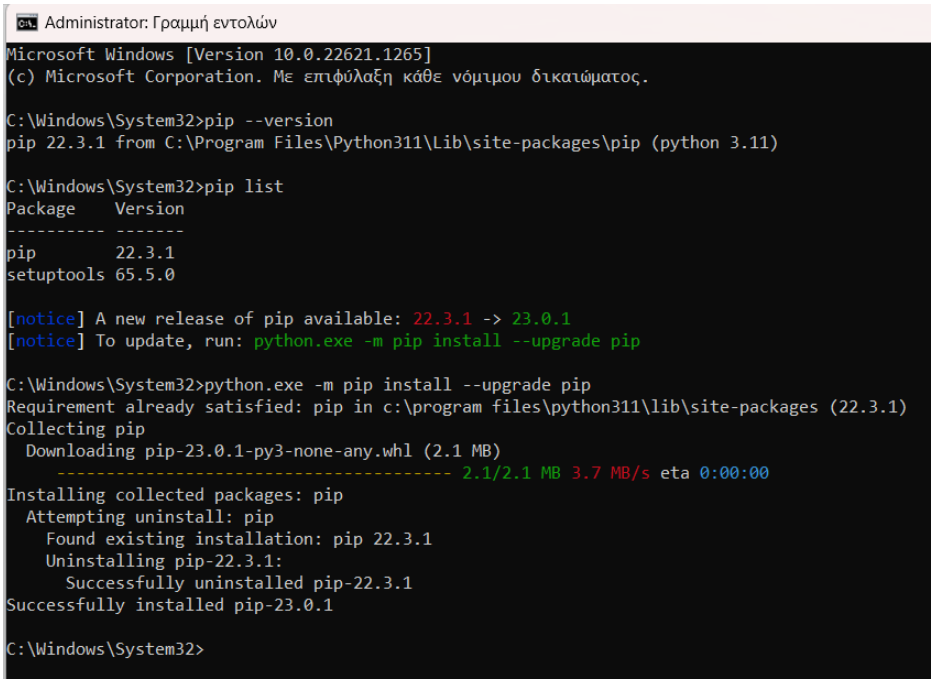




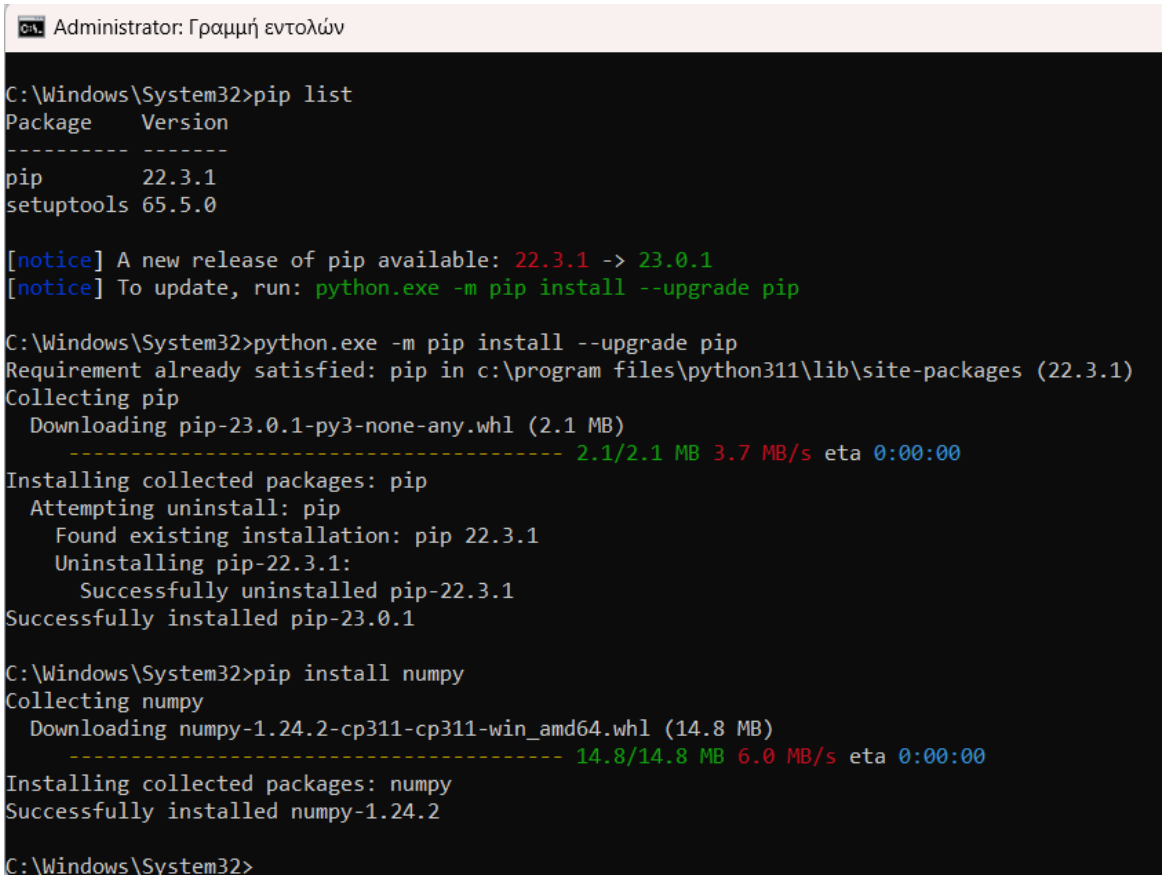
Στην συνέχεια πληκτρολογούμε την εντολή **pip list** όπως φαίνεται στην παρακάτω εικόνα, η οποία μας εμφανίζει τα πακέτα που είναι εγκατεστημένα στον υπολογιστή μας. Ακόμη, εδώ βλέπουμε μια ειδοποίηση που μας ενημερώνει ότι η έκδοση του pip η οποία είναι εγκατεστημένη στον υπολογιστή μας, δεν είναι η πιο πρόσφατη. Η pip μας ενημερώνει πως θα εγκαταστήσουμε την ενημέρωση αλλά αυτό δεν είναι απαραίτητο.



Αφού εγκαταστήσαμε και την ενημέρωση πατώντας python.exe -m pip install --upgrade pip, εγκαθιστούμε την Numpy.

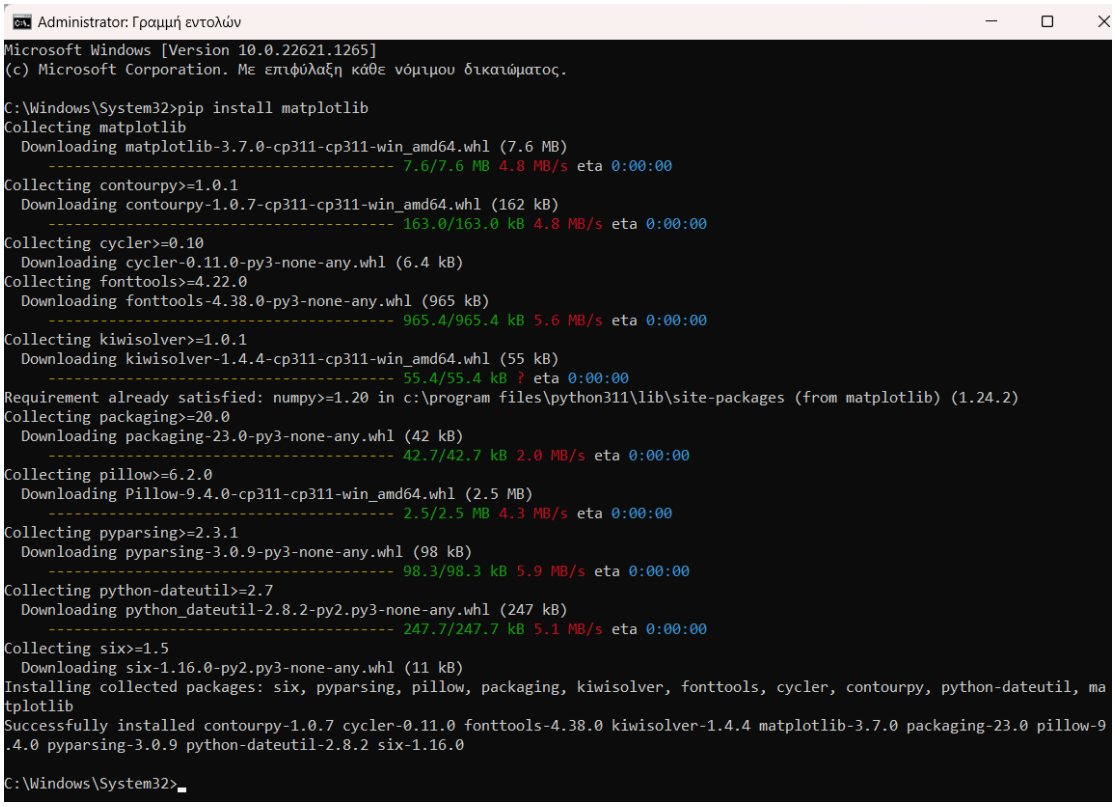


Τέλος, πληκτρολογώντας την εντολή pip install numpy και πατώντας enter μας εμφανίζεται το μήνυμα επιτυχούς εγκατάστασης, όπως φαίνεται στην παρακάτω εικόνα.



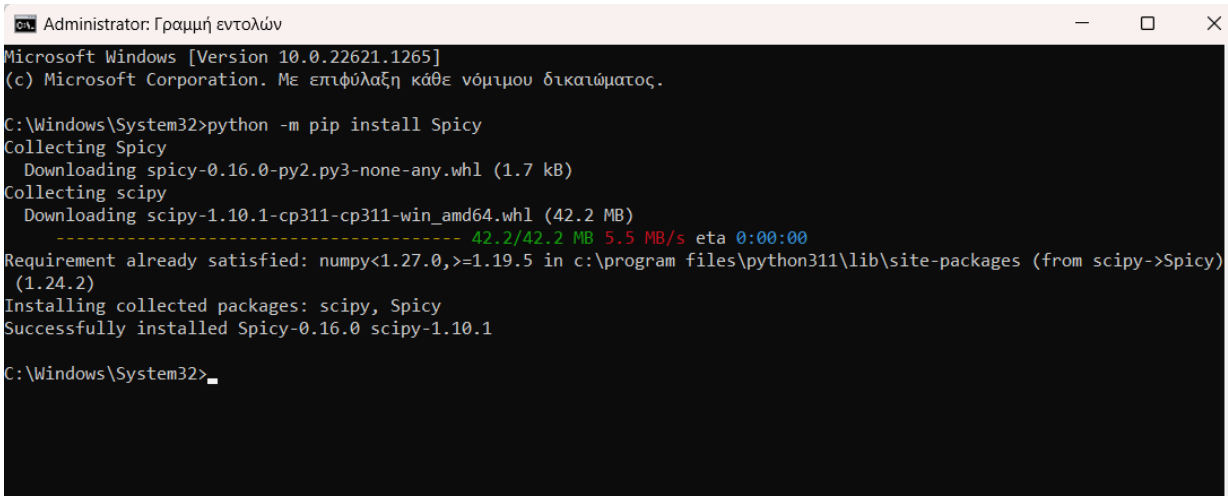
Εγκατάσταση βιβλιοθήκης matplotlib

Η matplotlib δεν είναι προ-εγκατεστημένη για αυτό και πρέπει να εγκατασταθεί πριν χρησιμοποιηθεί. Η εγκατάσταση είναι απλή και γίνεται αυτόματα με την χρήση της εντολής pip. Ανοίγουμε την Γραμμή εντολών και την εκτελούμε με δικαιώματα διαχειριστή. Στο παράθυρο εντολών πληκτρολογούμε την εντολή pip install matplotlib και πατάμε enter όπως φαίνεται και στην εικόνα παρακάτω.



Εγκατάσταση βιβλιοθήκης spicy

Η spicy δεν είναι προ-εγκατεστημένη για αυτό και πρέπει να εγκατασταθεί πριν χρησιμοποιηθεί. Η εγκατάσταση είναι απλή και γίνεται αυτόματα με την χρήση της εντολής pip. Ανοίγουμε την Γραμμή εντολών και την εκτελούμε με δικαιώματα διαχειριστή. Στο παράθυρο εντολών πληκτρολογούμε την εντολή python -m pip install Spicy και πατάμε enter όπως φαίνεται και στην εικόνα παρακάτω.



Εκκίνηση της Python

Η εκκίνηση της Python μπορεί να γίνει με δύο τρόπους. Είτε εκκινώντας την κονσόλα της Python επιλέγοντας το εικονίδιο Python (Command Line), είτε εκκινώντας το IDLE της Python επιλέγοντας το εικονίδιο Python (IDLE).

ΚΕΦΑΛΑΙΟ 3

Παρουσίαση εργαστηριακού οδηγού

Ενότητα 1

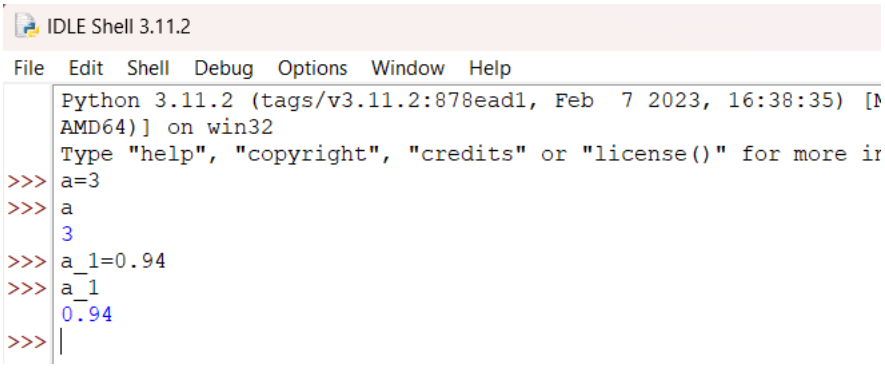
Αριθμητικές Μέθοδοι σε Προγραμματιστικό Περιβάλλον (Εργαστήριο 1)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την γλώσσα προγραμματισμού Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

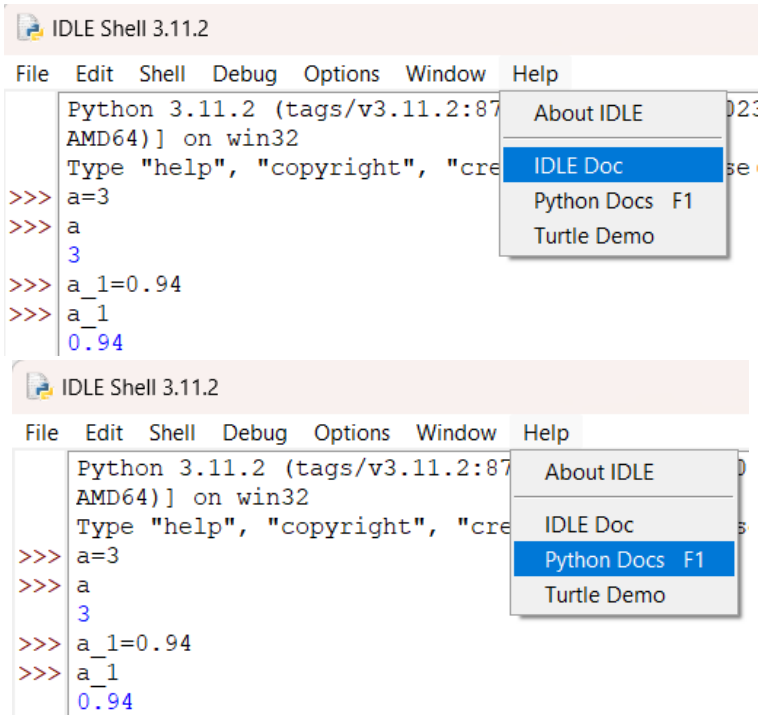
1. Εισαγωγή στην Python
    - Τελεστές στην Python
    - Σταθερές τιμές στην Python
    - Συναρτήσεις στην Python
    - Εντολές ελέγχου στην Python
2. Πίνακες
3. Μαθηματικές Συναρτήσεις

Εισαγωγή στην Python

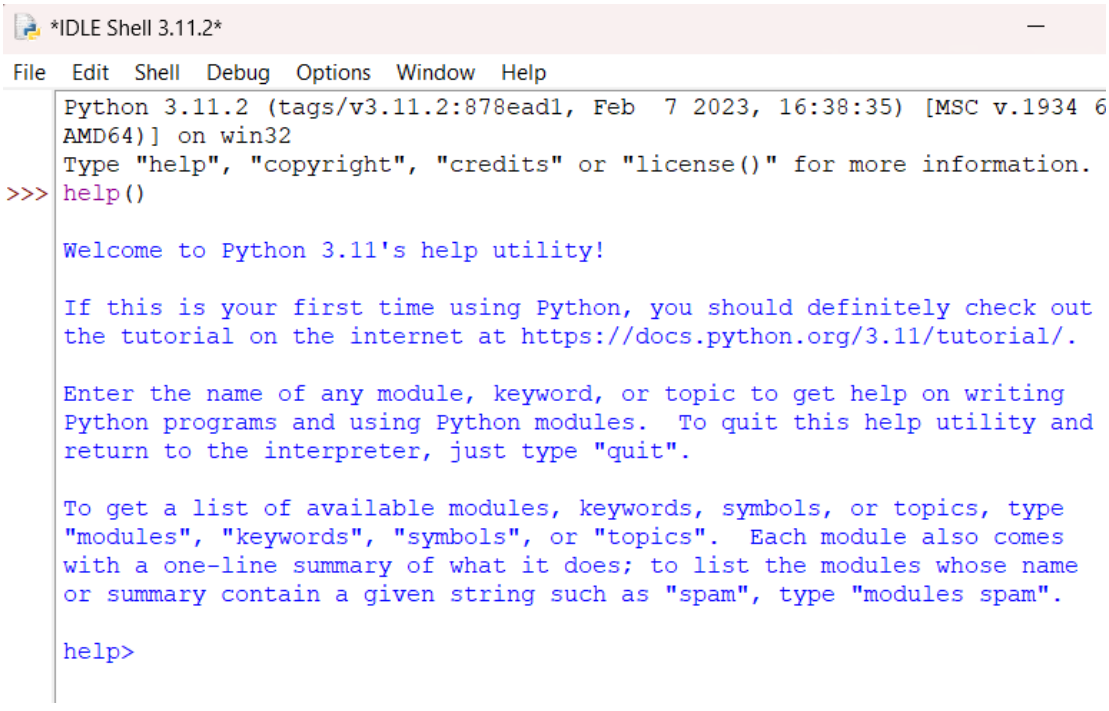
- Μεταβλητές
    - Προσοχή στην ονοματολογία των μεταβλητών. Μόνο λατινικοί χαρακτήρες, όχι κενά, όχι σύμβολα, να ξεκινά από γράμμα. (Case sensitive)
- Ανάθεση τιμής σε μεταβλητή
    - Η ανάθεση γίνεται με το (=)



- Βοήθεια στην Python
  - Από την μπάρα εργασιών επιλέγουμε το εικονίδιο της βοήθειας (Help) και από το μενού που αναδύεται επιλέγουμε είτε το IDLE Doc είτε το Python Docs F1 ανάλογα με την βοήθεια που αναζητούμε.

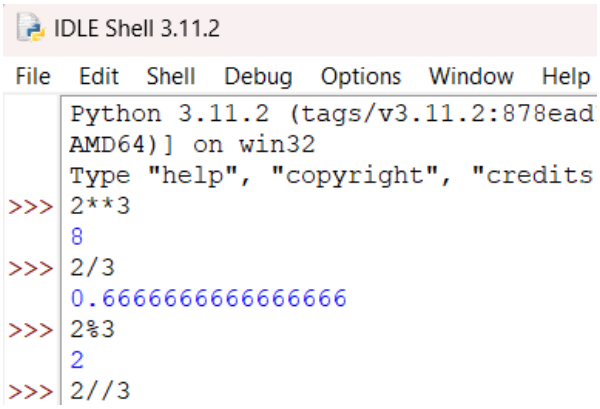


  - Απευθείας στο IDLE Shell με την χρήση της συνάρτησης help().

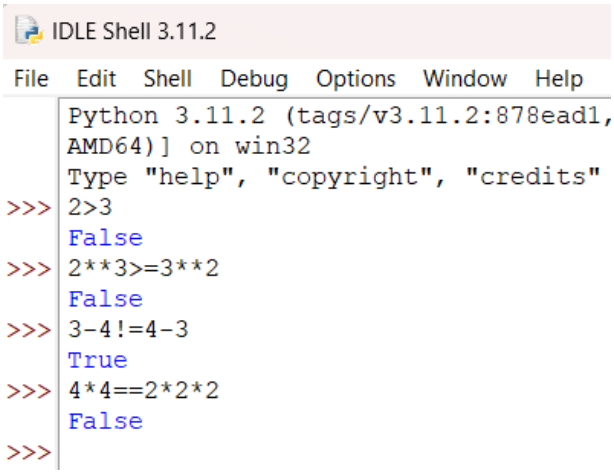


Τελεστές στην Python

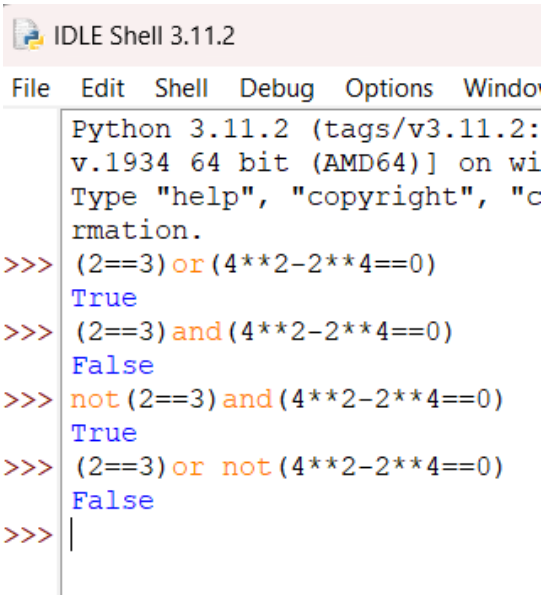
- Αριθμητικοί Τελεστές
  - \+ Πρόσθεση
  - \- Αφαίρεση
  - * Πολλαπλασιασμός
  - / Απλή διαίρεση
  - ** Ύψωση σε δύναμη
  - // Πηλίκο ακέραιας διαίρεσης
  - % Υπόλοιπο ακέραιας διαίρεσης



- Συγκριτικοί Τελεστές
  - \>
  - <
  - >=
  - <=
  - == Έλεγχος ισότητας
  - != Έλεγχος μη ισότητας

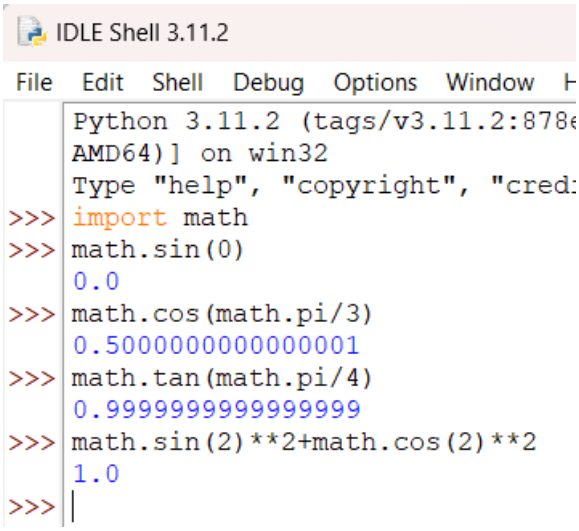


- Λογικοί Τελεστές
  - not Άρνηση
  - or Διάζευξη
  - and Σύζευξη



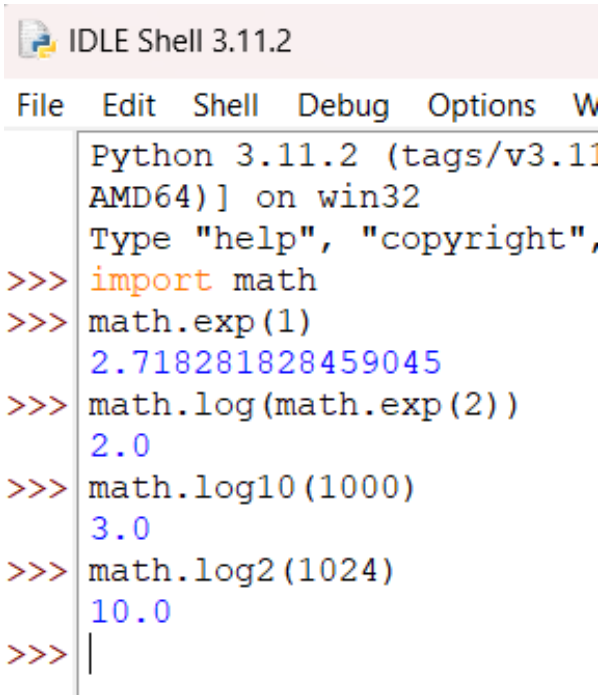
Συναρτήσεις στην Python

- Τριγωνομετρικές
  - sin() ημίτονο γωνίας σε rad
  - cos() συνημίτονο γωνίας σε rad
  - tan() εφαπτομένη γωνίας σε rad



- Εκθετικές Λογαριθμικές

  - exp() -> e^x
  - log() -> ln(x)
  - log10() -> log(x)
  - log2() -> $log_2(x)$



math.exp(1) --> e^1

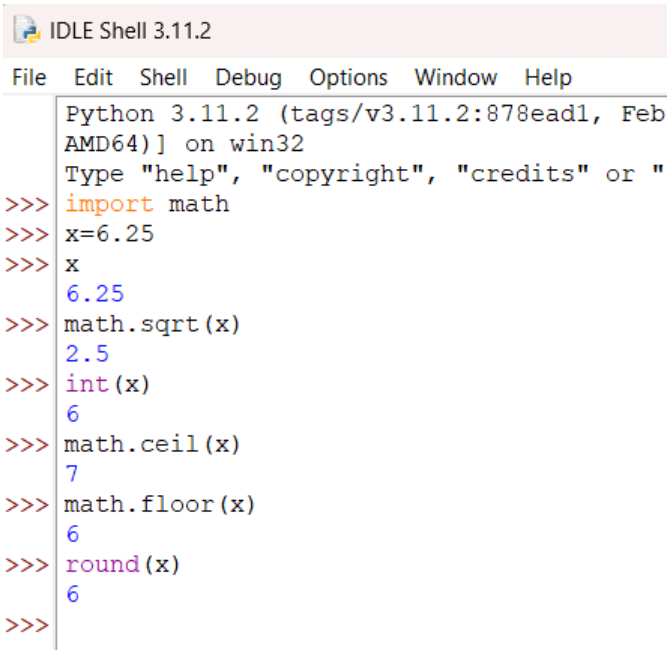
math.log(math.exp(2)) --> lne^2

math.log10(1000) --> log1000

math.log2(1024) --> $log_2 1024$

- Διάφορες

  - sqrt() -> $\sqrt{x}$
  - abs() -> |x|
  - int() -> [x] Ακέραιο μέρος
  - floor() -> ⌊x⌋ Κάτω ακέραιο φράγμα
  - ceil() -> ⌈x⌉ Άνω ακέραιο φράγμα
  - round() -> Στρογγυλοποίηση



- Για μιγαδικούς αριθμούς

Η Python έχει ενσωματωμένη υποστήριξη για μιγαδικούς αριθμούς και χρησιμοποιεί το j ή το J ως κατάληξη για να δείξει το φανταστικό μέρος (π.χ. 3+4j).

  - np.angle() -> πρωτεύον όρισμα μιγαδικού {*Για την χρήση της συνάρτησης np.angle() πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη numpy*}
  - x.real -> πραγματικό μέρος του μιγαδικού x
  - x.imag -> φανταστικό μέρος του μιγαδικού x

- np.conj() -> συζυγής μιγαδικός του μιγαδικού {*Για την χρήση της συνάρτησης np.conj() πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη numpy*}
- np.abs() -> μέτρο του μιγαδικού {*Για την χρήση της συνάρτησης np.abs() πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη numpy*}

Εντολές Ελέγχου στην Python

- locals() Εμφάνιση των μεταβλητών του IDLE Shell
- del x Καθαρισμός της μεταβλητής από το IDLE Shell
- os.system('cls') Καθαρισμός του terminal {*Για την χρήση της συνάρτησης os.system('cls') πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη os*}

Πίνακες

- Ορισμός πινάκων
  - Ορισμός διανυσμάτων (μονοδιάστατοι πίνακες)
    - Πίνακας γραμμή

      import numpy as np

      x=np.array([1,2,3])

    - Πίνακας στήλη

      import numpy as np

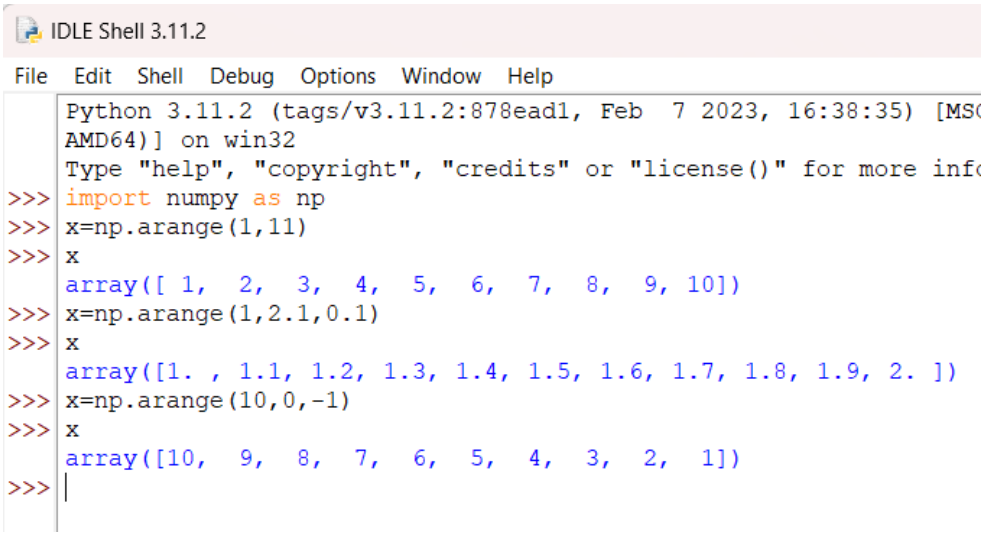
      x=np.array([[1],[2],[3]])
  - Ορισμός πινάκων δύο διαστάσεων
    - import numpy as np

      A=np.array([[1,2,3],[4,5,6],[7,8,9]]) ή

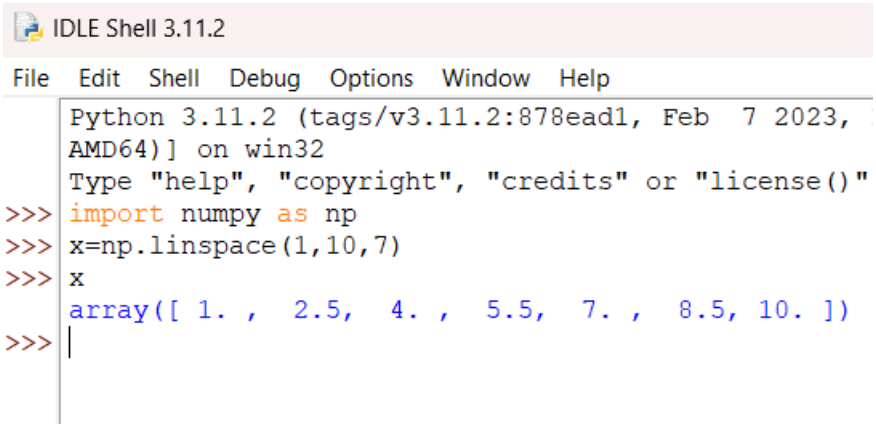
      import numpy as np

      A=np.array([[1,2,3],
      [4,5,6],
      [7,8,9]])
- Ορισμός ειδικών πινάκων
  - import numpy as np

    x=np.arange(a,b,s) ή {*η συνάρτηση arange δημιουργεί μια ακολουθία ακέραιων αριθμών αλλά δεν περιλαμβάνει την τελική τιμή, δηλαδή το b*}
  - a αρχική τιμή, s βήμα, b τελική τιμή (η οποία δεν περιλαμβάνεται)
  - όταν το βήμα παραλείπεται, τότε s=1



- x=np.linspace(a,b,n)
- γραμμικό διάστημα (ακολουθία ισαπέχοντων αριθμών) με a πρώτο αριθμό, b τελευταίο αριθμό, n πλήθος αριθμών



- x=np.ones((n,n)) ή x=np.ones((n,m))
- x=np.zeros((n,n)) ή x=np.zeros((n,m))
- x=np.eye(n) ή x=np.eye(n,m)
- x=np.random.rand(n,n) ή x=np.randpm.rand(n,m)
- Με ένα όρισμα (n) ή με (n,n) τετραγωνικοί πίνακες n x n, με δυο ορίσματα (n,m) διαστάσεων n x m

- Πράξεις πινάκων
  - Τελεστές (+), (-), (*), (/), (**)
    - Πράξη ανά στοιχείο (+), (-), (*), (/), (**)
    - Οι πίνακες πρέπει να έχουν τις ίδιες διαστάσεις
  - Συναρτήσεις για αντιστροφή πίνακα
    - δηλαδή,

$$A/B=A\cdot B^{-1}$$

$$A\backslash B=A^{-1}\cdot B$$

Χρησιμοποιείται η συνάρτηση np.matmul() για τον πολλαπλασιασμό δύο πινάκων και η συνάρτηση np.linalg.inv() για την αντιστροφή πίνακα

```
import numpy as np

A=np.array([[1,2],[3,4]])
B=np.array([[5,6],[7,8]])
x=np.matmul(A,np.linalg.inv(B))
print(x)
```

- ★ Οι πίνακες πρέπει να έχουν ίδιες διαστάσεις και να είναι τετραγωνικοί
- Συνάρτηση np.transpose()
  - ★ Αναστροφή πίνακα

- Συναρτήσεις
  - np.linalg.det(), np.trace(), np.linalg.eigvals(), np.linalg.inv()

Μαθηματικές Συναρτήσεις

- Ορισμός μαθηματικής συνάρτησης
  - Η σύνταξη της εντολής είναι

    def f(x):

        return math expression
  - Για παράδειγμα την συνάρτηση

    $$f(x) = x^2 + 3x$$

    την ορίζουμε:

    def f(x):

        return x**2+3*x

    Όλες οι συναρτήσεις συντάσσονται σε New File
- Κλήση της συνάρτησης
  - Η κλήση της συνάρτησης πραγματοποιείται ως εξής

    f(list of value)
  - Για παράδειγμα

    f(3)

    f(x) όπου x=1:10

Μαθηματικές Συναρτήσεις – Παραδείγματα

- Να οριστεί η συνάρτηση

  $$f(x) = x^2 + e^x + \ln(x+1)$$
- Να βρεθούν τα εξής
  - f(3),
  - f(x) όπου x = 1, 2, ……, 10,
  - $f^2(5) - 4f(1) + f(10)$

Ενότητα 2

Αριθμητικές Μέθοδοι σε Προγραμματιστικό Περιβάλλον (Εργαστήριο 2)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την γλώσσα προγραμματισμού Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Python file
    - Δημιουργία file
    - Εντολές σε Shell
2. Python δημιουργία *.py
    - Δημιουργία *.py
3. Python Συνάρτηση
    - Δημιουργία συνάρτησης

Δημιουργία file

- File -> New File ή Ctrl+N
- Τις εντολές που γράφουμε στον editor του περιβάλλοντος του IDLE μπορούμε να τις σώσουμε σε ένα αρχείο *.py . Το αρχείο αυτό μπορούμε να το εκτελέσουμε είτε από τον editor του περιβάλλοντος του IDLE επιλέγοντας Run -> Run Module (F5) είτε από το Command promt των Windows γράφοντας python name.py. Το αρχείο name.py και το εκτελέσιμο python πρέπει να είναι στον ίδιο κατάλογο διαφορετικά θα πρέπει να δηλωθεί όλη η διαδρομή. Επίσης, μπορούμε τις εντολές που γράφουμε στο Shell της Python να τις σώσουμε και αυτές σε ένα αρχείο *.py .

Εντολές σε Shell

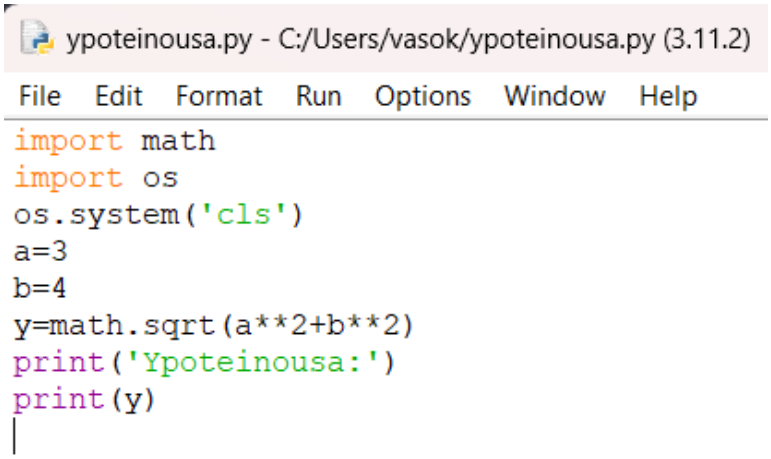
- os.system('cls') Καθαρισμός του terminal {*Για την χρήση της συνάρτησης os.system('cls') πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη os*}
- plt.close('all') Κλείνει όλα τα γραφικά παράθυρα {Για την χρήση της συνάρτησης plt.close('all') πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη matplotlib.pyplot}
- print(a) Εμφανίζει το περιεχόμενο της μεταβλητής a
- print('text') Εμφανίζει το κείμενο text

- a=input() Εισαγωγή τιμών από το πληκτρολόγιο στην μεταβλητή a
- #comment Σχόλια

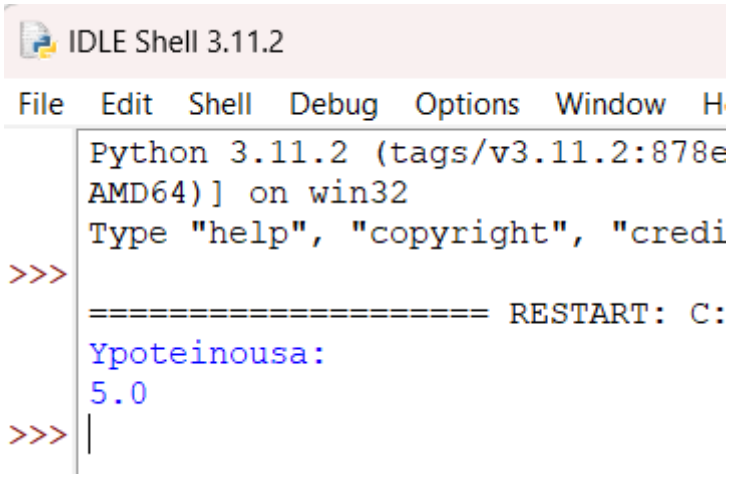
Python δημιουργία *.py - Παράδειγμα

Να γίνει αρχείο py το οποίο να υπολογίζει την υποτείνουσα ορθογωνίου τριγώνου δοθέντων των δύο κάθετων πλευρών a=3, b=4.

- Στον Editor γράφουμε

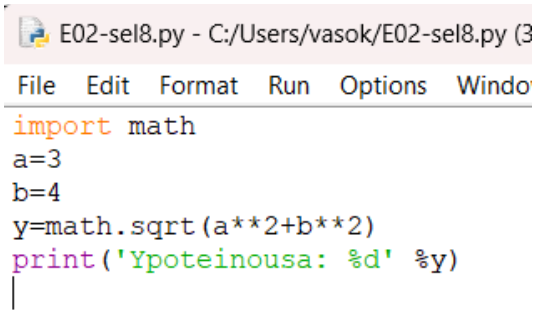


- Εκτελούμε και στο Shell έχουμε

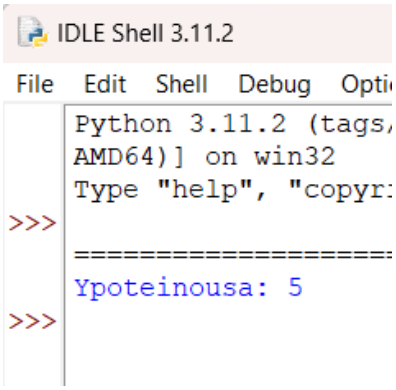


Στο προηγούμενο παράδειγμα χρησιμοποιούμε την εντολή print δύο φορές. Μια για να εμφανίσουμε κείμενο και μια για να εμφανίσουμε την τιμή της μεταβλητής. Μπορούμε να χρησιμοποιήσουμε μια print η οποία θα εμφανίζει στην ίδια σειρά και το κείμενο που επιθυμούμε και την τιμή της μεταβλητής.

- Στο editor γράφουμε



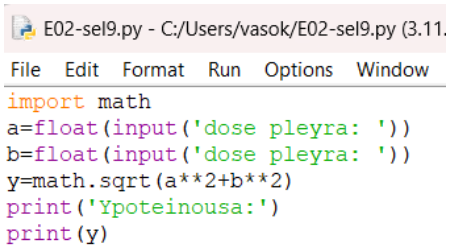
- Εκτελούμε και στο Shell έχουμε



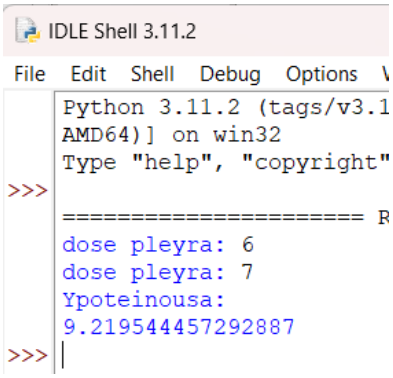
Python δημιουργία *.py – Παράδειγμα 2

Να γίνει αρχείο py το οποίο να υπολογίζει την υποτείνουσα ορθογωνίου τριγώνου δοθέντων των δύο κάθετων πλευρών.

- Στον editor γράφουμε

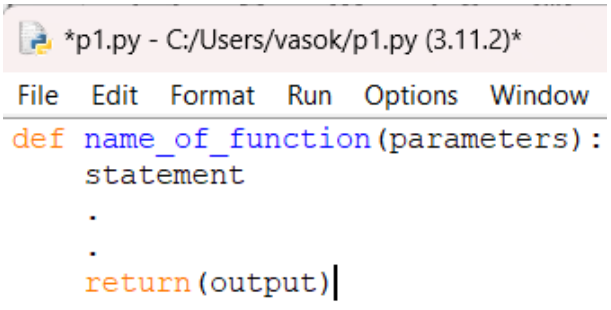


- Εκτελούμε και στο Shell έχουμε



Δημιουργία Συνάρτησης

- Η συνάρτηση αποτελεί ένα ξεχωριστό τμήμα προγράμματος το οποίο επιτελεί μια συγκεκριμένη διαδικασία.
- Μια συνάρτηση μπορεί να έχει είσοδο και έξοδο
  - Συναρτήσεις χωρίς είσοδο και χωρίς έξοδο
  - Συναρτήσεις μόνο με είσοδο
  - Συναρτήσεις μόνο με έξοδο
  - Συναρτήσεις με είσοδο και έξοδο
- Οι μεταβλητές στην συνάρτηση έχουν τοπική εμβέλεια
- Μια συνάρτηση στην Python έχει την ακόλουθη σύνταξη:



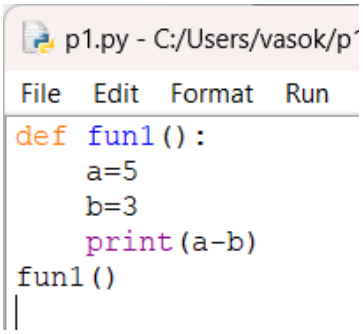
  - name_of_function -> είναι το όνομα της συνάρτησης
  - parameters -> είναι οι παράμετροι εισόδου της συνάρτησης
  - return -> είναι η εντολή η οποία επιστρέφει τιμή ή τιμές η συνάρτηση

- output -> είναι η επιστρεφόμενη τιμή της συνάρτησης

- Το σώμα της συνάρτησης οριοθετείτε από τη στοίχιση
- Παρατηρήσεις
  - Η συνάρτηση καλείται με το όνομα που της έχει δοθεί (name_of_function) και όχι το όνομα του αρχείου *.py .
  - Τα σχόλια μέσα στην συνάρτηση είναι προαιρετικά αλλά μας βοηθούν σημαντικά για την υπόδειξη των ορισμάτων εισόδου, τις μεταβλητές εξόδου αλλά και για την λειτουργία που εκτελεί η συνάρτηση.

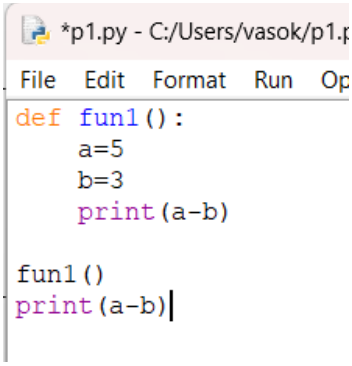
Κλήση Συνάρτησης

- Κλήση Συνάρτησης
  - Μια συνάρτηση την καλούμε ανάλογα με την περίπτωση
    - ★ Συνάρτηση χωρίς είσοδο και χωρίς έξοδο



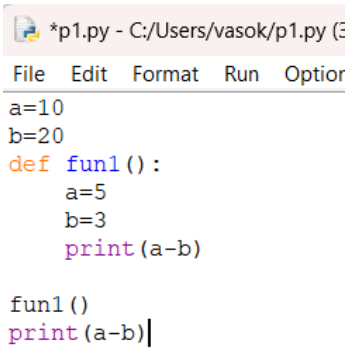
Στις τέσσερις πρώτες γραμμές έχουμε την συνάρτηση και στην πέμπτη γραμμή την κλήση της συνάρτησης.

- ★ Συνάρτηση χωρίς είσοδο και χωρίς έξοδο, εμβέλεια μεταβλητών



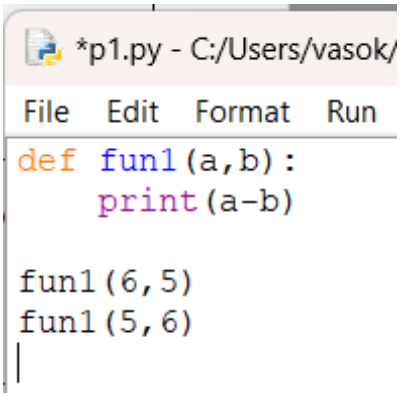
Το παραπάνω πρόγραμμα παρουσιάζει πρόβλημα στην εντολή print(a-b) λόγω της τοπικής εμβέλειας των μεταβλητών στη συνάρτηση

★ Συνάρτηση χωρίς είσοδο και χωρίς έξοδο, εμβέλεια μεταβλητών



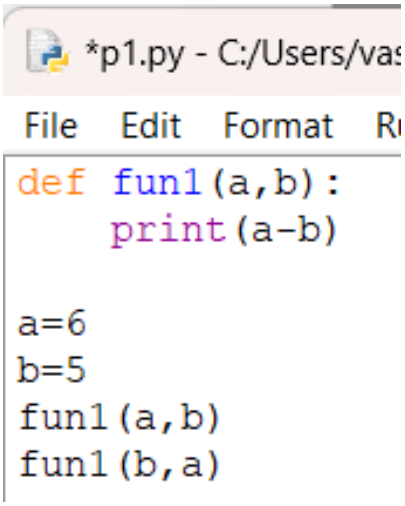
Οι τιμές a,b έχουν άλλες τιμές στη συνάρτηση και άλλες τιμές στο πρόγραμμα.

★ Συνάρτηση μόνο με είσοδο



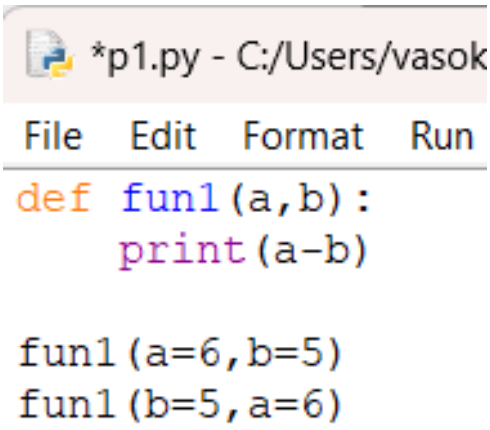
Κλήση με απευθείας ανάθεση τιμών εισόδου.
Προσοχή στη σειρά.

★ Συνάρτηση μόνο με είσοδο



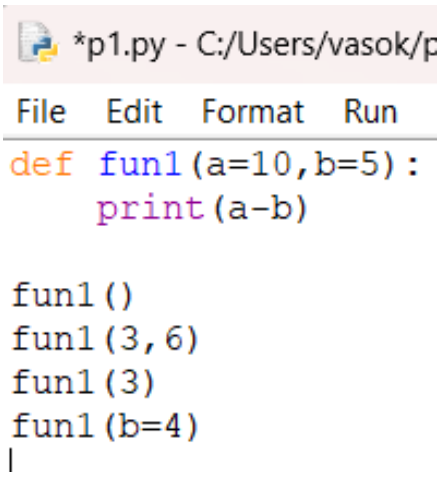
Κλήση με την χρήση μεταβλητών στην ανάθεση τιμών εισόδου. Προσοχή στη σειρά.

★ Συνάρτηση μόνο με είσοδο



Κλήση με τη χρήση των ορισμάτων εισόδου της συνάρτησης

★ Συνάρτηση μόνο με είσοδο και με προκαθορισμένες τιμές



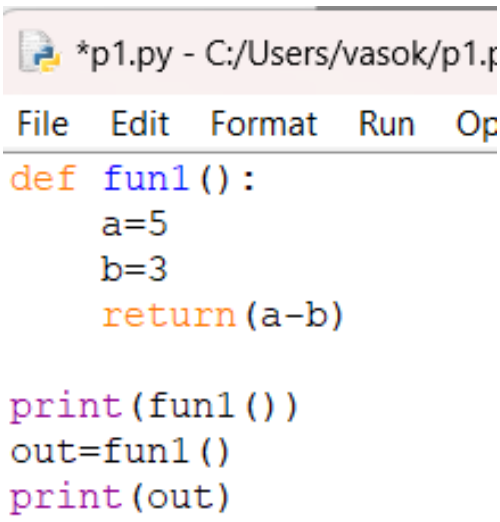
Στην πρώτη κλήση χρησιμοποιούνται οι προκαθορισμένες τιμές των ορισμάτων εισόδου της συνάρτησης.

Στη δεύτερη κλήση χρησιμοποιούνται με απευθείας ανάθεση οι τιμές 3 και 6.

Στην τρίτη κλήση χρησιμοποιούνται με απευθείας ανάθεση η τιμή 3 στο πρώτο όρισμα και η προκαθορισμένη τιμή του δεύτερου ορίσματος.

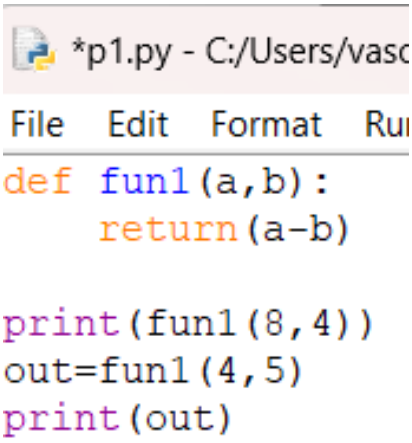
Στην τέταρτη κλήση χρησιμοποιούνται η προκαθορισμένη τιμή του πρώτου ορίσματος και ορίζουμε το δεύτερο όρισμα εισόδου της συνάρτησης.

★ Συνάρτηση μόνο με έξοδο



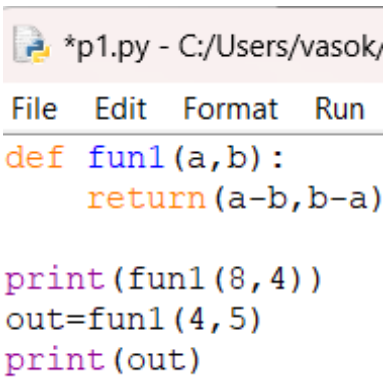
Με την εντολή return(a-b) η συνάρτηση επιστρέφει το αποτέλεσμα της πράξης a-b μέσω του ονόματος της κατά την κλήση της.
Το αποτέλεσμα της συνάρτησης μπορούμε να το καταχωρήσουμε σε μεταβλητή.

★ Συνάρτηση με είσοδο και έξοδο



Με την εντολή return(a-b) η συνάρτηση επιστρέφει το αποτέλεσμα της πράξης a-b μέσω του ονόματος της κατά την κλήση της.
Κατά την κλήση ορίζουμε τα ορίσματα εισόδου της συνάρτησης.
Το αποτέλεσμα της συνάρτησης μπορούμε να το καταχωρήσουμε σε μεταβλητή.

★ Συνάρτηση με είσοδο και πολλαπλή έξοδο



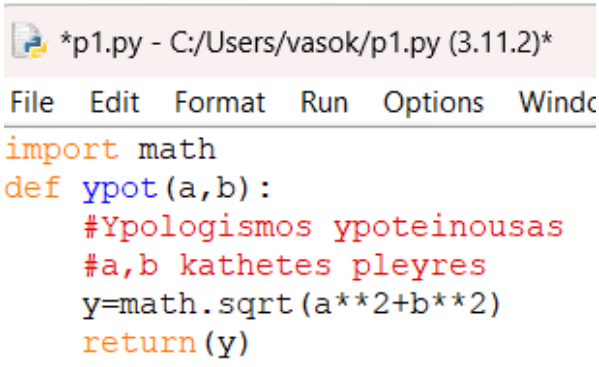
Με την εντολή return(a-b,b-a) η συνάρτηση επιστρέφει τα αποτελέσματα των πράξεων a-b και b-a μέσω του ονόματος της κατά την κλήση της.
Κατά την κλήση ορίζουμε τα ορίσματα εισόδου της συνάρτησης.
Το αποτέλεσμα της συνάρτησης μπορούμε να το καταχωρήσουμε σε μεταβλητή τύπου tuple.

- Μια συνάρτηση την καλούμε με το όνομα της μέσα στο αρχείο *.py και τα ορίσματα της είτε με το όνομα της και τα ορίσματα της στο Shell.

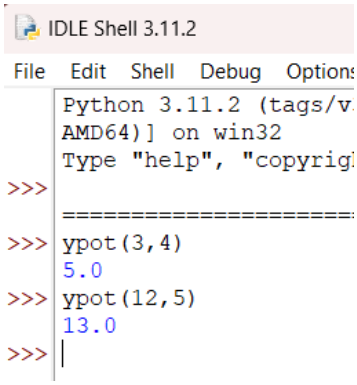
Συνάρτηση – Παράδειγμα

Να γίνει συνάρτηση που να δέχεται τις κάθετες πλευρές ενός ορθογωνίου τριγώνου και να επιστρέφει την υποτείνουσα

- Στον Editor γράφουμε



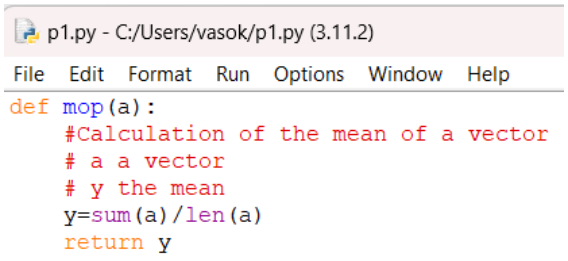
- Στο Shell καλούμε την συνάρτηση και έχουμε



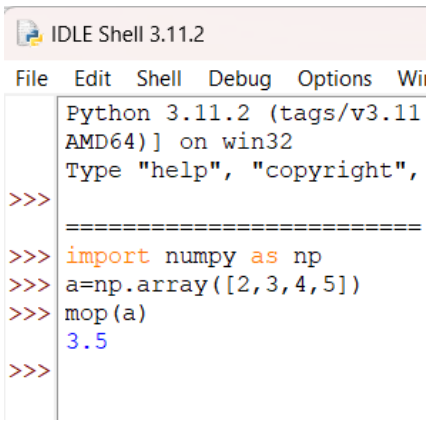
Συνάρτηση – Παράδειγμα 2

Να γίνει συνάρτηση που να δέχεται έναν πίνακα (μονοδιάστατο) και να επιστρέφει το μέσο όρο των στοιχείων του πίνακα.

- Στον Editor γράφουμε



- Στο Shell καλούμε την συνάρτηση και έχουμε



Συνάρτηση – Παράδειγμα 3

Να γίνει συνάρτηση που να δέχεται έναν πίνακα (μονοδιάστατο) και να επιστρέφει το μέγιστο και το ελάχιστο των στοιχείων του πίνακα.

- Στον Editor γράφουμε

p1.py - C:/Users/vasok/p1.py (3.11.2)

File Edit Format Run Options Window Help

```
def pin(a):
    #Calculation of the max and min of a vector
    # a  a vector
    # y  the max element
    # z  the min element
    y=max(a)
    z=min(a)
    return y,z
```

- Στο Shell καλούμε την συνάρτηση και έχουμε

IDLE Shell 3.11.2

File Edit Shell Debug Options Window

```
Python 3.11.2 (tags/v3.11.2:87
AMD64)] on win32
Type "help", "copyright", "cre
>>>
========================== RES
>>> import numpy as np
>>> a=np.array([3,5,7,9])
>>> pin(a)
(9, 3)
>>> y1,y2=pin(a)
>>> y1
9
>>> y2
3
>>>
```

Ενότητα 3

Αριθμητικές Μέθοδοι σε Προγραμματιστικό Περιβάλλον (Εργαστήριο 3)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την δημιουργία γραφικών παραστάσεων και την χρήση βασικών δομών στο περιβάλλον της Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Γραφικές παραστάσεις
    - Γραφική παράσταση συνάρτησης
    - Γραφική παράσταση σημείων
2. Βασικές δομές
    - Παράδειγμα (for)
    - Παράδειγμα (while)

Γραφική Παράσταση Συνάρτησης

- Για την δημιουργία γραφικής παράστασης μιας συνάρτησης χρησιμοποιούμε την εντολή plt.plot(x,y,'parameters') {*Για την χρήση της συνάρτησης* plt.plot(x,y,'parameters') *πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη matplotlib*}
- Πρώτα ορίζουμε τα διανύσματα x και y.
    - x είναι το διάνυσμα των τετμημένων των σημείων της συνάρτησης.
    - y είναι το διάνυσμα των τεταγμένων των σημείων της συνάρτησης.
- Το όρισμα 'parameters' είναι προαιρετικό με το οποίο καθορίζουμε το χρώμα, το σχήμα των σημείων και το είδος της γραμμής.
- Έπειτα μπορούμε να προσθέσουμε και κάποιες άλλες εντολές για την καλύτερη παρουσίαση της γραφικής παράστασης.
    - plt.title('Title of the Plot') Εισαγωγή τίτλου στο γράφημα
    - plt.xlabel('x') Ετικέτα στον οριζόντιο άξονα
    - plt.ylabel('y') Ετικέτα στον κατακόρυφο άξονα
    - plt.text(x,y, 'some text') Κείμενο μέσα στο γράφημα

Γραφική Παράσταση Συνάρτησης – Παράδειγμα

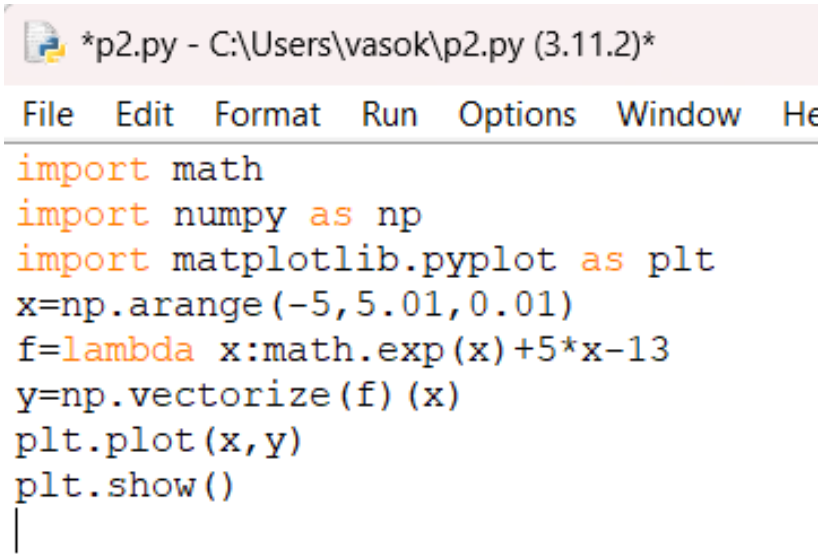
Να γίνει η γραφική παράσταση της συνάρτησης

$f(x) = e^x + 5x - 13$ $στο$ $διάστημα$ $[-5,5]$.

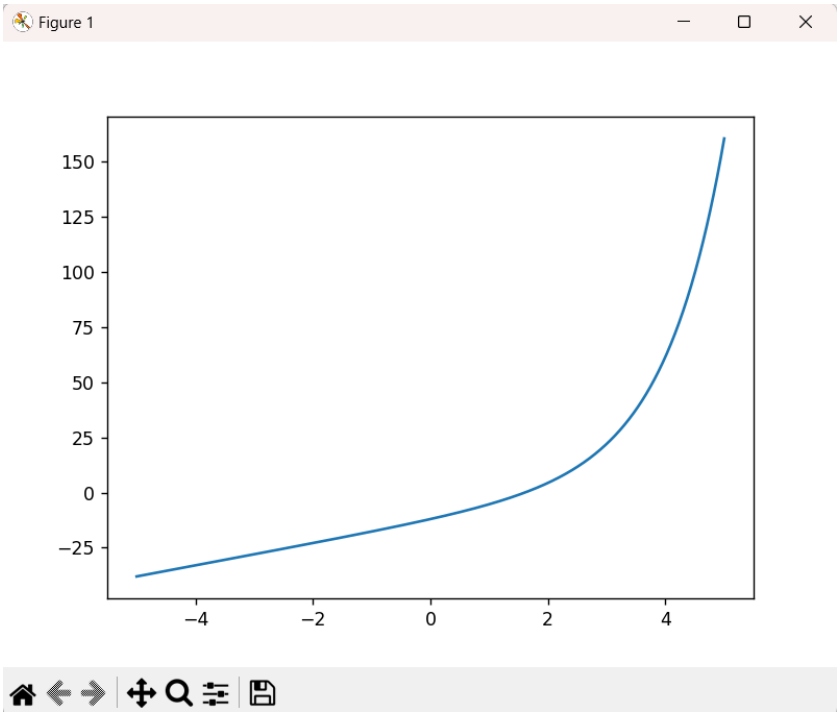
Στην Python θα έχουμε

- ορίζουμε την συνάρτηση f(x)
  f=lambda x: math.exp(x)+5*x-13 {η συνάρτηση lambda δέχεται ένα μόνο όρισμα στην περίπτωση μας το x και επιστρέφει την τιμή της έκφρασης exp(x)+5*x-13 για οποιαδήποτε τιμή του χ}
- δημιουργούμε τα διανύσματα x και y
  x=np.arange(-5,5.01,0.01)
  y=np.vectorize(f)(x) {η συνάρτηση vectorize εφαρμόζει την συνάρτηση f σε κάθε στοιχείο του πίνακα x}
- καλούμε την συνάρτηση plt.plot() με τα κατάλληλα ορίσματα
  plt.plot(x,y)
- καλούμε την συνάρτηση plt.show() για να εμφανιστεί η γραφική παράσταση

- Στο Editor γράφουμε



- Το εκτελούμε και μας επιστρέφει το παρακάτω figure



Στις γραφικές παραστάσεις των συναρτήσεων χρειάζεται να αποτυπωθεί και ο άξονας x'x

- Στην συνάρτηση plt.plot(x,y) προσθέτουμε το ζεύγος των διανυσμάτων που αναπαριστά τον x'x
  x,np.zeros_like(x)

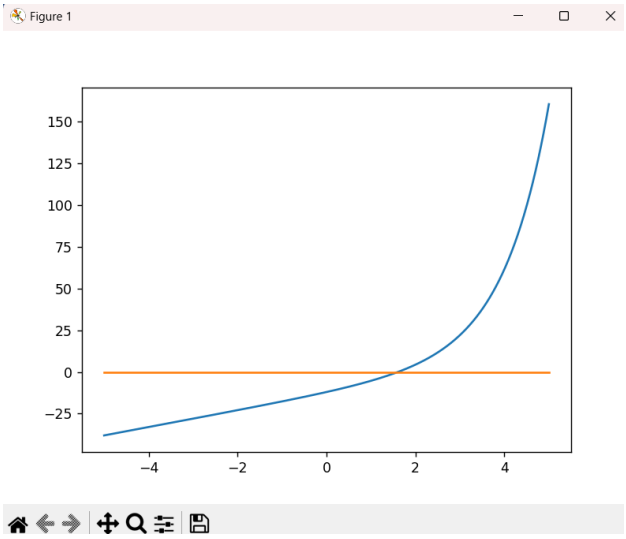
- Στον Editor γράφουμε

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

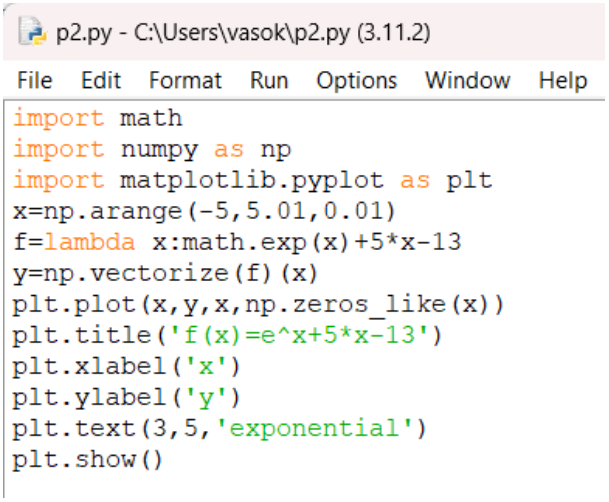
```
import math
import numpy as np
import matplotlib.pyplot as plt
x=np.arange(-5,5.01,0.01)
f=lambda x:math.exp(x)+5*x-13
y=np.vectorize(f)(x)
plt.plot(x,y,x,np.zeros_like(x))
plt.show()
```

- Το εκτελούμε και μας επιστρέφει το παρακάτω figure

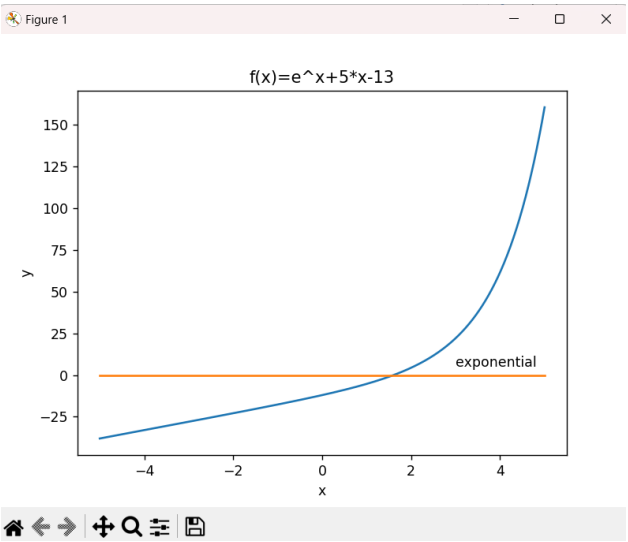


Επιπλέον στο figure μπορούμε να προσθέσουμε πληροφορίες όπως φαίνεται παρακάτω

- Στον Editor γράφουμε



- Το εκτελούμε και μας επιστρέφει το παρακάτω figure



Γραφική Παράσταση Σημείων

- Για την δημιουργία γραφικής παράστασης ενός συνόλου διακριτών σημείων χρησιμοποιούμε την εντολή plt.plot(x,y,'parameters')
- Πρώτα ορίζουμε τα διανύσματα x και y.
  - x είναι το διάνυσμα των τετμημένων των σημείων.
  - y είναι το διάνυσμα των τεταγμένων των σημείων.
- Στο όρισμα 'parameters' στο οποίο ορίζουμε το χρώμα, το σχήμα των σημείων και το είδος της γραμμής, πρέπει να ορίσουμε υποχρεωτικά το σχήμα των σημείων και να μην ορίσουμε το είδος της γραμμής.

Γραφική Παράσταση Σημείων – Παράδειγμα

Δίνεται ο παρακάτω πίνακας τιμών

X	2	3	4	5	6
Y	-1	3	5	2	7

Να γίνει η γραφική παράσταση των παραπάνω σημείων.

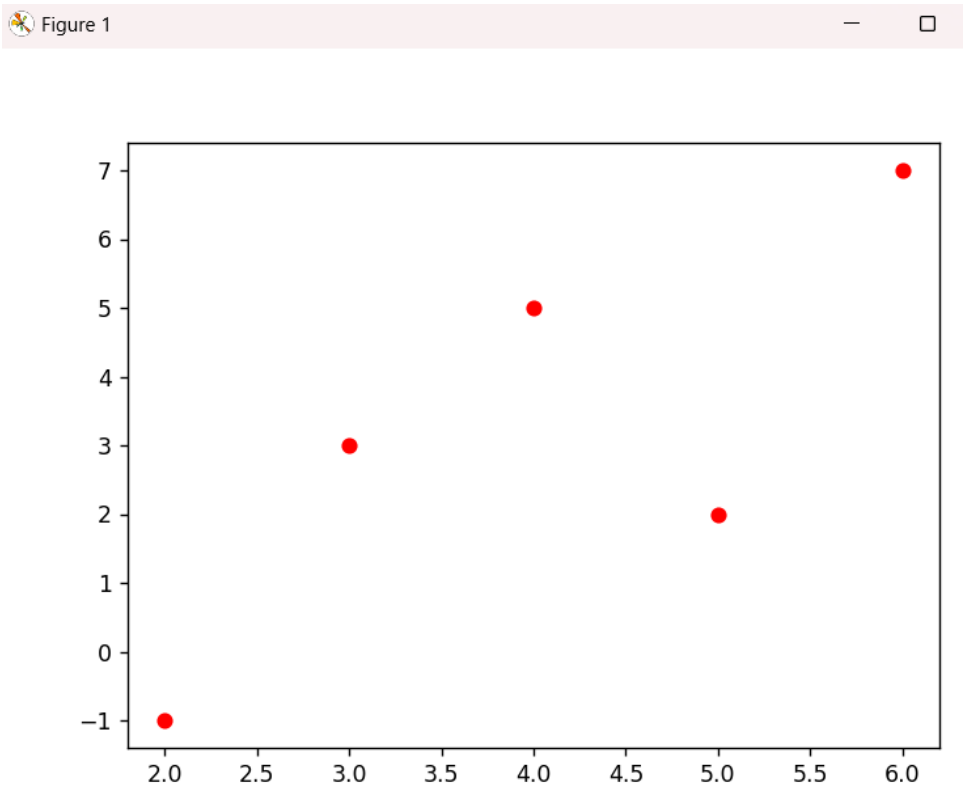
- Στον Editor γράφουμε

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
import numpy as np
import matplotlib.pyplot as plt

x=np.array([2,3,4,5,6])
y=np.array([-1,3,5,2,7])

plt.plot(x,y,'ro')
plt.show()
```

- Το εκτελούμε και μας επιστρέφει το παρακάτω figure



Βασικές Δομές

- Δομές Επιλογές
  - if
  - if-else
  - if-elif

- Δομές Επανάληψης
  - for
  - while

- Πρόσθετες Εντολές
  - break

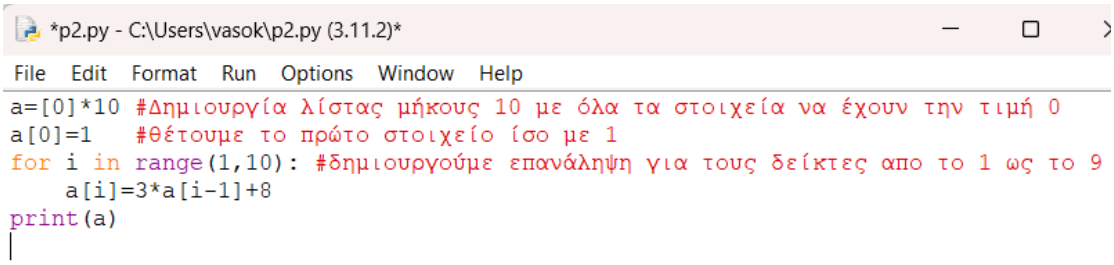
Βασικές Δομές – Παράδειγμα (for)

Να βρεθούν οι 10 πρώτες τιμές της ακολουθίας (αναδρομικός τύπος) που δίνεται από τον τύπο

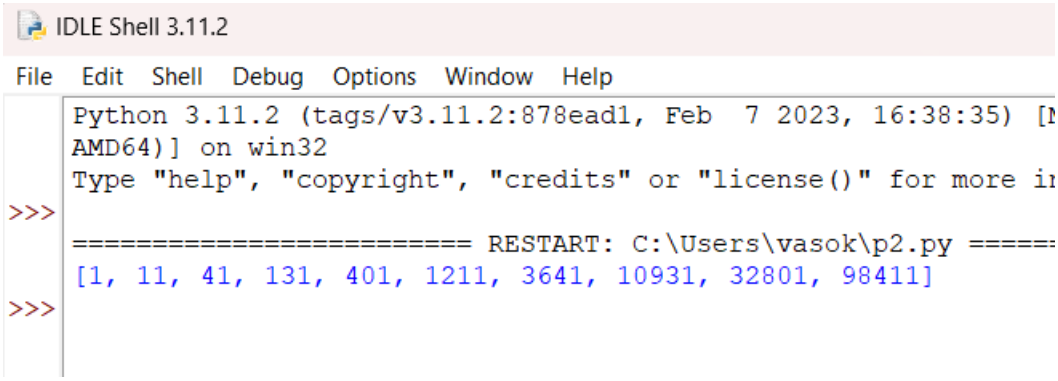
$$a_n = 3 \cdot a_{n-1} + 8$$

με αρχική τιμή $a_1 = 1$.

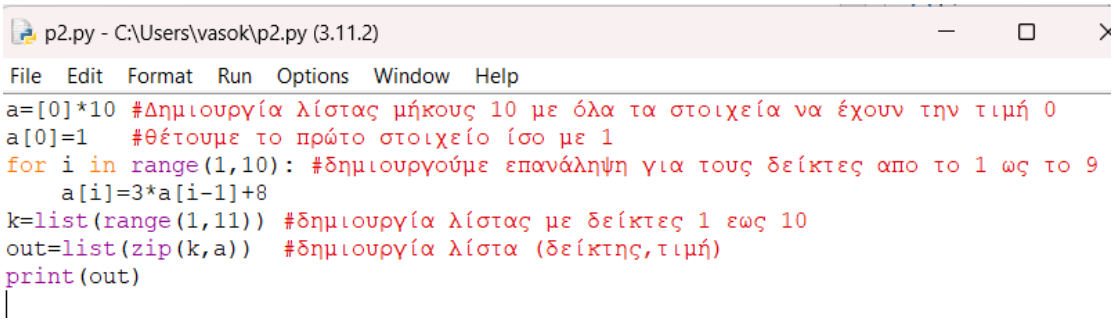
- Στον Editor γράφουμε



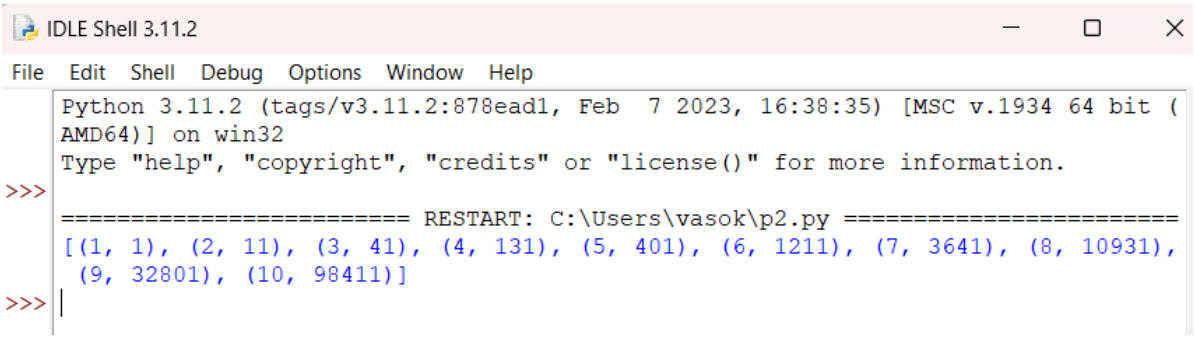
- Εκτελούμε και στο Shell έχουμε



- Μπορούμε στο αποτέλεσμα να προσθέσουμε τον αριθμό των βημάτων. Στον Editor γράφουμε



- Εκτελούμε και στο Shell έχουμε



Βασικές Δομές – Παράδειγμα (While)

Να βρεθούν οι τιμές της ακολουθίας (αναδρομικός τύπος) α_n για τις οποίες να ισχύει η σχέση

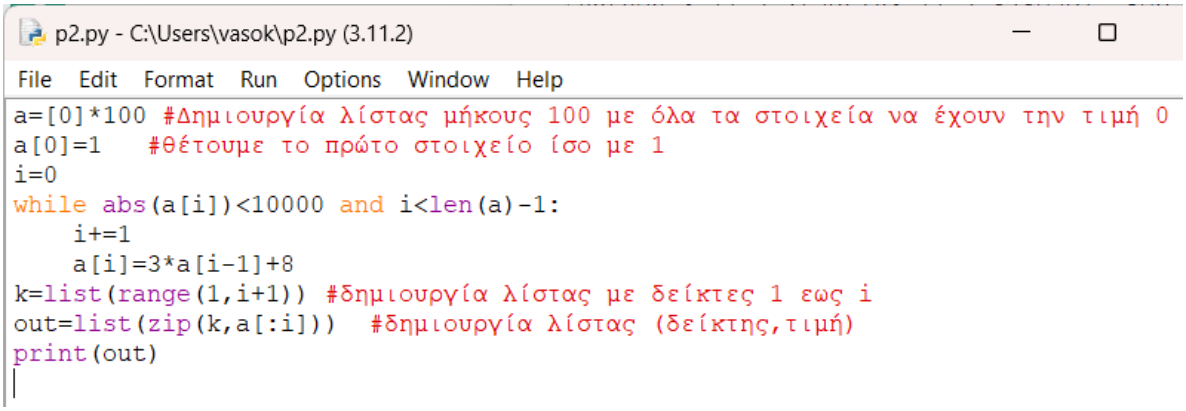
$$|\alpha_n| < 10000$$

Οι τιμές της ακολουθίας α_n δίνονται από τον τύπο

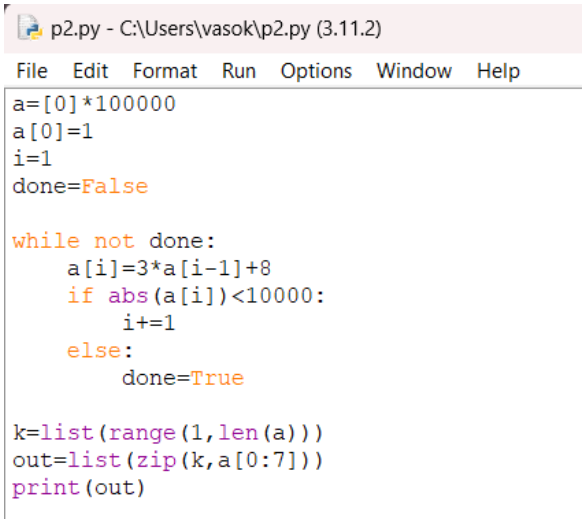
$$a_n = 3 \cdot a_{n-1} + 8$$

με αρχική τιμή $\alpha_1 = 1$.

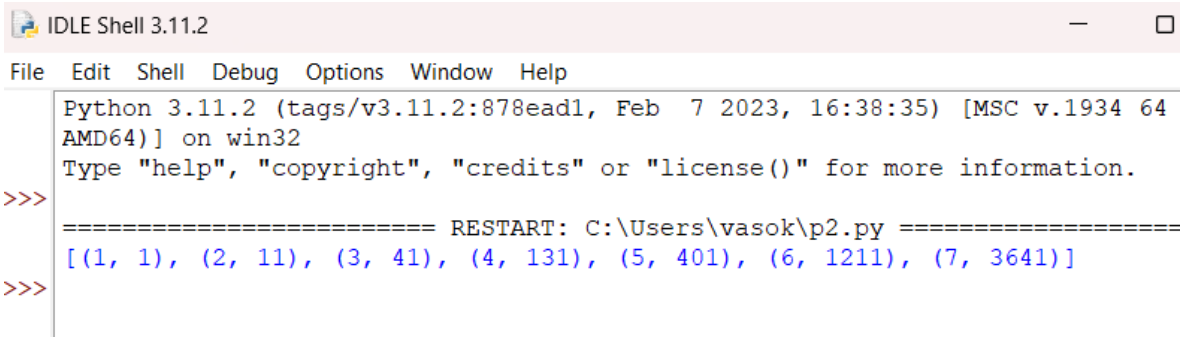
- Στον Editor γράφουμε είτε αυτό



- Στον Editor γράφουμε είτε αυτό



- Εκτελούμε και στο Shell έχουμε



Βασικές Δομές – Παράδειγμα 2 (While)

Να βρεθούν οι τιμές της ακολουθίας (αναδρομικός τύπος) a_n για τις οποίες να ισχύει η σχέση

$$|a_n - a_{n-1}| < 10^6$$

Οι τιμές της ακολουθίας a_n δίνονται από τον τύπο

$$a_n = 3 \cdot a_{n-1} + 8$$

με αρχική τιμή $a_1 = 1$.

- Στον Editor γράφουμε

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
a=[0]*100000
a[0]=1
i=1
done=False

while not done:
    a[i]=3*a[i-1]+8
    if abs(a[i]-a[i-1])<10**6:
        i+=1
    else:
        done=True

k=list(range(1,len(a)))
out=list(zip(k,a[0:12]))
print(out)
```

- Εκτελούμε και στο Shell έχουμε

IDLE Shell 3.11.2

File Edit Shell Debug Options Window Help

```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\vasok\p2.py ==========
[(1, 1), (2, 11), (3, 41), (4, 131), (5, 401), (6, 1211), (7, 3641), (8, 10931),
 (9, 32801), (10, 98411), (11, 295241), (12, 885731)]
>>>
```

Ενότητα 4

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 4)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με τη διαδικασία μετατροπής αριθμών σε αριθμητικά συστήματα, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητικά συστήματα – Μετατροπές
    - Μετατροπή ακέραιου μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα
    - Μετατροπή κλασματικού μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα
    - Παραδείγματα
    - Ασκήσεις

Αριθμητικά Συστήματα

- Μετατροπές αριθμών από ένα αριθμητικό σύστημα σε άλλο
    - Μετατροπή ακέραιου μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα
    - Μετατροπή κλασματικού μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα
    - Μετατροπή ακέραιου μέρους αριθμού x από δεκαδικό σύστημα σε αριθμητικό σύστημα με βάση b
    - Μετατροπή κλασματικού μέρους αριθμού x από δεκαδικό σύστημα σε αριθμητικό σύστημα με βάση b

Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$

- Απλή διαδικασία, αν ακολουθήσετε τον τύπο

$$[x] = \pm a_n b^n + a_{n-1} b^{n-1} + \cdots + a_0 b^0$$

$$= \pm \sum_{i=n}^{0} a_i b^i$$

- Για παράδειγμα

$$(53473)_8 = 5 \cdot 8^4 + 3 \cdot 8^3 + 4 \cdot 8^2 + 7 \cdot 8^1 + 3 \cdot 8^0$$
$$= (22331)_{10}$$

Algorithm 1 Μετατροπή ακέραιου μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα (Απ' ευθείας)

Input : x ∈ **Z, b, n** ∈ **N**

y ← 0

for i=0 to n **do**

y ← y + $a_i * b^i$

end for

Output: y

---

a_i είναι τα ψηφία του αριθμού x.

Επομένως, για τον αριθμό $x = (53473)_8$ θα έχουμε

i	$y \leftarrow y + a_i * b^i$
–	0
0	$0 + 3 \cdot 8^0 = 3$
1	$3 + 7 \cdot 8^1 = 59$
2	$59 + 4 \cdot 8^2 = 315$
3	$315 + 3 \cdot 8^3 = 1851$
4	$1851 + 5 \cdot 8^4 = 22331$

δηλαδή, $y = (22331)_{10}$.

- Συνάρτηση σε Python – Πρώτη εκδοχή

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Hel

```
def b2decM(a, b):
    n = len(a)
    a = a[::-1] #Αντιστροφή πίνακα
    y = 0
    for i in range(n):
        y += a[i] * b**(i)
    return y
```

- Η μεταβλητή a είναι ένας πίνακας με στοιχεία τα ψηφία του αριθμού x.
- Εκτελούμε στο Shell και έχουμε

IDLE Shell 3.11.2

File Edit Shell Debug Options Window

```
Python 3.11.2 (tags/v3.11.2:8
AMD64)] on win32
Type "help", "copyright", "cr
>>>
========================= RES
>>> import numpy as np
>>> a=np.array([5,3,4,7,3])
>>> b=8
>>> b2decM(a, b)
22331
>>>
```

- Συνάρτηση σε Python – Δεύτερη εκδοχή

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def b2dec(x, b):
    xc = str(x) #μετατροπή αριθμού σε συμβολοσειρά
    n = len(xc)
    a = [int(xc[n-i-1]) for i in range(n)] #δημιουργία πίνακα a με αντίστροφη σειρά
    #και μετατροπή με int σε ακέραιο αριθμό
    y = 0
    for i in range(n):
        y += a[i] * b**(i)
    return y
```

- Εκτελούμε στο Shell και έχουμε

IDLE Shell 3.11.2

File Edit Shell Debug Options Window Help

```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb
AMD64)] on win32
Type "help", "copyright", "credits" or "
>>>
========================= RESTART: C:\Us
>>> b2dec(53473,8)
22331
>>>
```

- Στα ορίσματα της συνάρτησης έχουμε τον αριθμό x ο οποίος χωρίζεται στα ψηφία του μέσα στη συνάρτηση.

Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ (Σχήμα horner)

- Αν ακολουθήσουμε διαφορετική τακτική (Σχήμα Horner) μπορούμε να μειώσουμε τον αριθμό των πράξεων, π.χ.

$$\begin{aligned}(53473)_8 &= 5\cdot 8^4 + 3\cdot 8^3 + 4\cdot 8^2 + 7\cdot 8^1 + 3\cdot 8^0 \\ &= \left(5\cdot 8^3 + 3\cdot 8^2 + 4\cdot 8^1 + 7\right)\cdot 8 + 3 \\ &= \left(\left(5\cdot 8^2 + 3\cdot 8^1 + 4\right)\cdot 8 + 7\right)\cdot 8 + 3 \\ &= \left(\left(\left(5\cdot 8^1 + 3\right)\cdot 8 + 4\right)\cdot 8 + 7\right)\cdot 8 + 3 \\ &= (22331)_{10}\end{aligned}$$

Algorithm 2 Μετατροπή ακέραιου x από βάση b σε δεκαδικό σύστημα (Σχήμα Horner)

Input: a_i, b
  $y \leftarrow a_n$
  for i=n-1 to 0 **do**
    $y \leftarrow a_i + y * b$
  end for
Output: y

a_i είναι τα ψηφία του αριθμού x.

Επομένως, για τον αριθμό $x = (53473)_8$ θα έχουμε

i	$y \leftarrow a_i + y * b$
-	5
3	$3 + 5\cdot 8 = 43$
2	$4 + 43\cdot 8 = 348$
1	$7 + 348\cdot 8 = 2791$
0	$3 + 2791\cdot 8 = 22331$

δηλαδή, $y = (22331)_{10}$.

- Συνάρτηση σε Python – Πρώτη εκδοχή

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def b2decMH(a, b):
    n = len(a)
    a = a[::-1]
    y = a[n-1]
    for i in range(n-2, -1,-1):
        y = a[i] + y*b
    return y
```

- Η μεταβλητή a είναι ένας πίνακας με στοιχεία τα ψηφία του αριθμού x.

- Εκτελούμε στο Shell και έχουμε

IDLE Shell 3.11.2

File Edit Shell Debug Options Wi

```
Python 3.11.2 (tags/v3.11.
AMD64)] on win32
Type "help", "copyright",
>>>
==========================
>>> import numpy as np
>>> a=np.array([5,3,4,7,3])
>>> b=8
>>> b2decMH(a, b)
22331
>>>
```

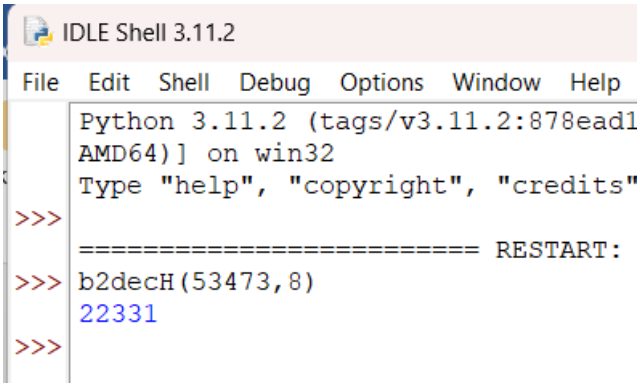
- Συνάρτηση σε Python – Δεύτερη εκδοχή

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def b2decH(x,b):
    xc = str(x)
    n = len(xc)
    a = [int(xc[n-i-1]) for i in range(n)]
    y = a[n-1]
    for i in range(n-2,-1,-1):
        y = a[i] + b*y
    return y
```

- Εκτελούμε στο Shell και έχουμε



- Στα ορίσματα της συνάρτησης έχουμε τον αριθμό x ο οποίος χωρίζεται στα ψηφία του μέσα στη συνάρτηση.

Μετατροπή $(x-[X])_b \to (\cdot)_{10}$

- Απλή διαδικασία, αν ακολουθήσουμε τον τύπο

$$\begin{aligned} x-[x] &= \pm a_{-1}b^{-1}+a_{-2}b^{-2}+\dots \\ &= \pm\sum_{i=-1}^{-\infty} a_i b^i \end{aligned}$$

- Για παράδειγμα

$$\begin{aligned} (.53)_8 &= 5\cdot 8^{-1}+3\cdot 8^{-2} \\ &= 5\cdot\frac{1}{8}+3\cdot\frac{1}{8^2} \\ &= (.671875)_{10} \end{aligned}$$

- Ισοδύναμα με σχήμα horner

$$\begin{aligned} (.53)_8 &= 5\cdot 8^{-1}+3\cdot 8^{-2} \\ &= \left(5+3\cdot\frac{1}{8}\right)\cdot\frac{1}{8} \\ &= (.671875)_{10} \end{aligned}$$

Algorithm 3 Μετατροπή κλασματικού μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα.

Input: $x \in \mathbb{Z},\ b,\ k \in \mathbb{N}$
  $y \leftarrow 0$
  for $i = -k$ to -1 **do**
    $y \leftarrow (a_i + y)/b$
  end for
Output: y

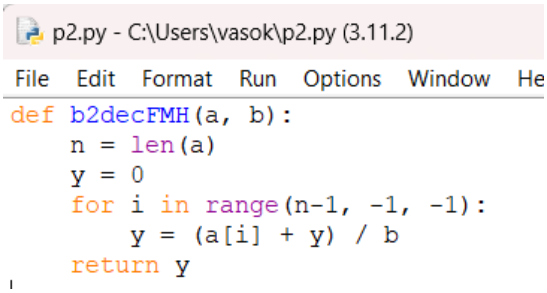
a_i είναι τα ψηφία του αριθμού x.

Επομένως, για τον αριθμό $x = (.53)_8$ θα έχουμε

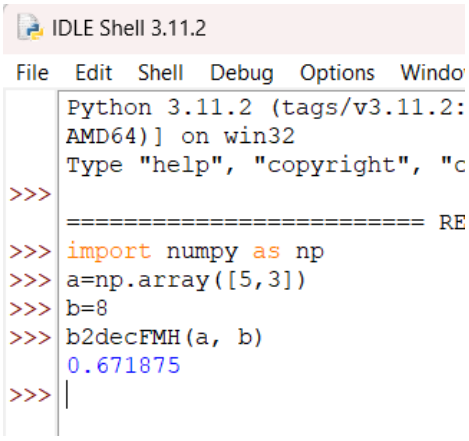
i	$y \leftarrow (a_i + y)/b$
-	0
-2	$(3+0)/8 = 0.375$
-1	$(0.375+5)/8 = 0.671875$

δηλαδή, $y = (0.671875)_{10}$.

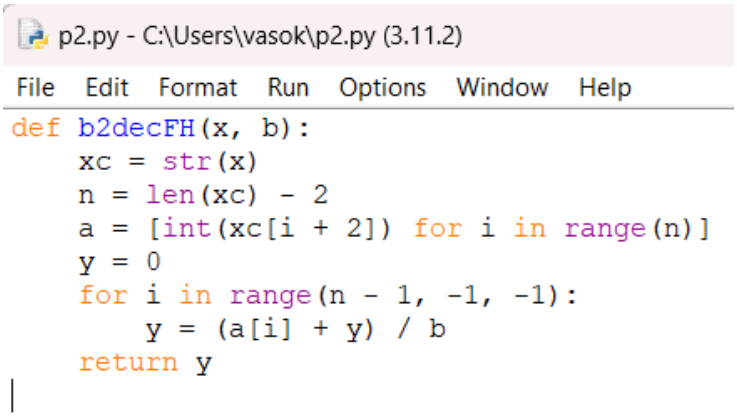
- Συνάρτηση σε Python – Πρώτη εκδοχή



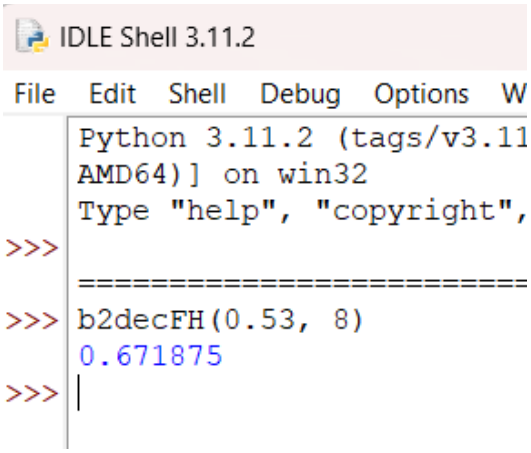
- Η μεταβλητή a είναι ένας πίνακας με στοιχεία τα ψηφία του αριθμού x.
- Εκτελούμε στο Shell και έχουμε



- Συνάρτηση σε Python – Δεύτερη εκδοχή



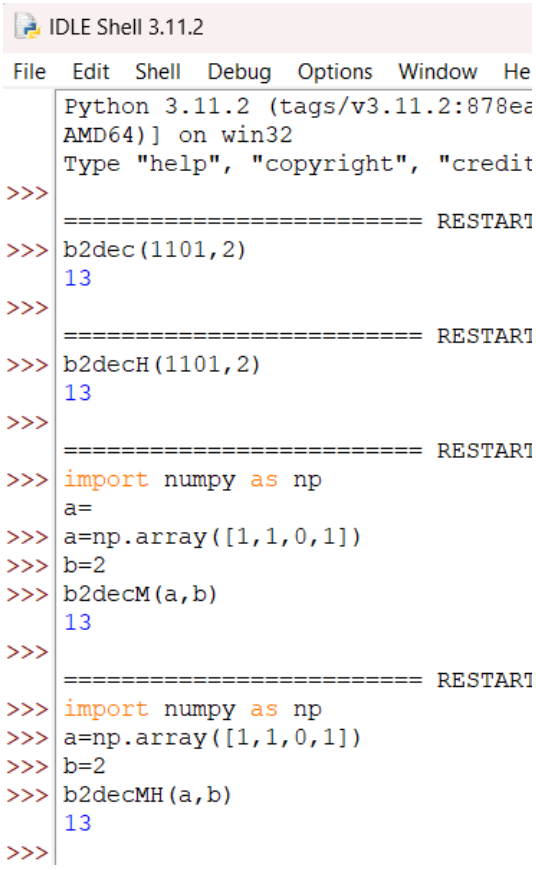
- Εκτελούμε στο Shell και έχουμε



- Στα ορίσματα της συνάρτησης έχουμε τον αριθμό x ο οποίος χωρίζεται στα ψηφία του μέσα στη συνάρτηση.

Παραδείγματα

- Να μετατραπεί ο αριθμός $(1101)_2$ σε δεκαδική βάση.
  Στο Shell έχουμε



Μπορούμε να χρησιμοποιήσουμε οποιαδήποτε εκδοχή, με είσοδο τον αριθμό ή τα ψηφία του αριθμού σε πίνακα

- Είτε τον αριθμό (b2dec(1101,2) ή b2decH(1101,2))
- Είτε τα ψηφία του σε πίνακα (import numpy as np

  a=np.array([1,1,0,1])

  b=2

  b2decM(a,b) ή

  import numpy as np

  a=np.array([1,1,0,1])

  b=2 b2decMH(a,b))

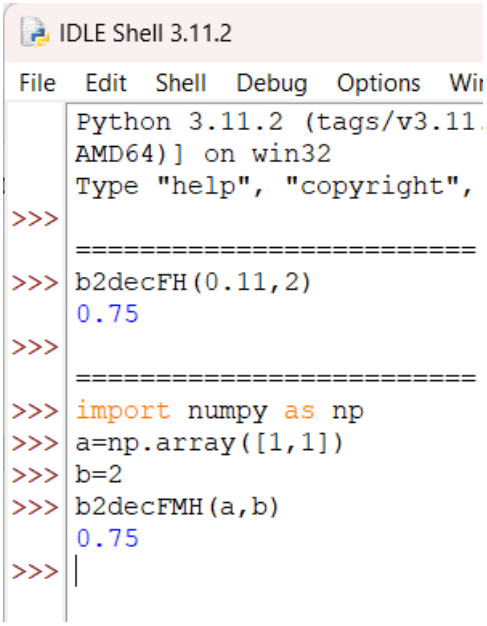
ή με τη μέθοδο horner ή όχι

- Στο όνομα της συνάρτησης αν υπάρχει το H ή όχι.

Επομένως, έχουμε

$$(1101)_2 = (13)_{10}$$

- Να μετατραπεί ο αριθμός $(0.11)_2$ σε δεκαδική βάση.
  Στο Shell έχουμε



Μπορούμε να χρησιμοποιήσουμε οποιαδήποτε εκδοχή, με είσοδο τον αριθμό ή τα ψηφία του αριθμού σε πίνακα

- Είτε τον αριθμό (b2decFH(0.11,2))
- Είτε τα ψηφία του σε πίνακα (import numpy as np
  a=np.array([1,1])
  b=2
  b2decFMH(a,b))

Επομένως, έχουμε

$$(0.11)_2 = (0.75)_{10}$$

Ασκήσεις

- Να μετατραπεί ο αριθμός $(12345)_6$ σε δεκαδική βάση.
  - **Απάντηση:** $(12345)_6 = (1865)_{10}$

- Να μετατραπεί ο αριθμός $(0.125)_8$ σε δεκαδική βάση.
  - **Απάντηση:** $(0.125)_8 = (0.166015625)_{10}$

- Να μετατραπεί ο αριθμός $(1011.1001)_2$ σε δεκαδική βάση.
  - **Απάντηση:** $(1011.1001)_2 = (11.5625)_{10}$

Ενότητα 5

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 5)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με τη διαδικασία μετατροπής αριθμών σε αριθμητικά συστήματα, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητικά συστήματα – Μετατροπές
    - Μετατροπή ακέραιου μέρους αριθμού x από δεκαδικό σύστημα σε αριθμητικό σύστημα με βάση b
    - Μετατροπή κλασματικού μέρους αριθμού x από δεκαδικό σύστημα σε αριθμητικό σύστημα με βάση b
    - Παραδείγματα
    - Ασκήσεις

Αριθμητικά Συστήματα

- Μετατροπές αριθμών από ένα αριθμητικό σύστημα σε άλλο
    - Μετατροπή ακέραιου μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα
    - Μετατροπή κλασματικού μέρους αριθμού x από αριθμητικό σύστημα με βάση b σε δεκαδικό σύστημα
    - Μετατροπή ακέραιου μέρους αριθμού x από δεκαδικό σύστημα σε αριθμητικό σύστημα με βάση b
    - Μετατροπή κλασματικού μέρους αριθμού x από δεκαδικό σύστημα σε αριθμητικό σύστημα με βάση b

Μετατροπή $([x])_{10} \rightarrow (\cdot)_b$

- Αριθμός της Ευκλείδειας Διαίρεσης.
- Διαιρούμε διαδοχικά με την βάση b, αρχικά τον αριθμό x και στην συνέχεια τα πηλίκα μέχρις ότου το πηλίκο να γίνει μηδέν.
- Τα ψηφία του αριθμού είναι τα υπόλοιπα που μας επιστρέφουν οι διαδοχικές διαιρέσεις.

Algorithm 1 Μετατροπή ακέραιου x από δεκαδικό σύστημα σε βάση b (Αλγόριθμος της Διαίρεσης)

Input: $x \in \mathbb{Z},\ b$
  $i \leftarrow 0$
  while $x \neq 0$ **do**
    $a_i \leftarrow x \mod b$
    $x \leftarrow [x/b]$
    $i \leftarrow i + 1$
  end while
Output: a_i

a_i είναι τα ψηφία του αριθμού που μετατράπηκε.

Επομένως, για τον αριθμό $x = (369)_{10}$ σε βάση b=8, θα έχουμε

i	$a_i \leftarrow x \mod b$	$x \leftarrow [x/b]$
0	1	46
1	6	5
2	5	0

δηλαδή, $a_0 = 1, a_1 = 6, a_2 = 5$ ή ισοδύναμα $(561)_8$

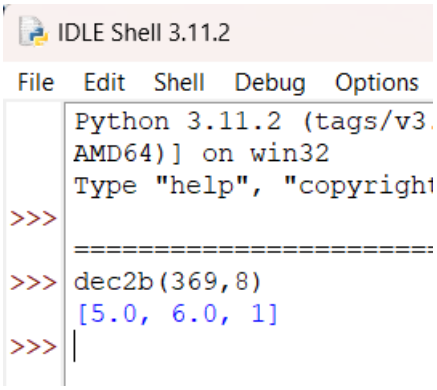
- Συνάρτηση σε Python

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def dec2b(x, b):
    a = []
    i = 1
    while x != 0:
        a.append(x % b)
        x = round(x // b * 1000) / 1000
        i += 1
    return a[::-1]
```

- Εκτελούμε στο Shell και έχουμε



- Η μεταβλητή a είναι ένας πίνακας με στοιχεία τα ψηφία του αριθμού που μετατράπηκε.

Μετατροπή $(x-[x])_{10} \to (\cdot)_b$

- Αρχικά, πολλαπλασιάζουμε τον αριθμό x με το b για να ορίσουμε το y.
- Όσο το y είναι διάφορο του μηδενός
  - Κρατάμε το ακέραιο μέρος του y (ψηφία του ζητούμενου αριθμού)
  - Πολλαπλασιάζουμε το κλασματικό μέρος του αριθμού y με το b για να ορίσουμε εκ' νέου το y.

Algorithm 2 Μετατροπή κλασματικού x από δεκαδικό σύστημα σε βάση b

Input: $x \in \mathbb{Z}$, b
  $y \leftarrow b * x$
  $i \leftarrow -1$
  while $y \neq 0$ **do**
    $a_i \leftarrow [y]$
    $y \leftarrow (y - [y]) * b$
    $i \leftarrow i - 1$
  end while
Output: a_i

a_i είναι τα ψηφία του αριθμού που μετατράπηκε.

Επομένως, για τον αριθμό $x = (.875)_{10}$ σε βάση b=2, θα έχουμε

i	a_i	y
-	-	1.75
-1	1	1.5
-2	1	1
-3	1	0

δηλαδή, $a_{-1} = 1$, $a_{-2} = 1$, $a_{-3} = 1$ ή ισοδύναμα $(.111)_2$

- Συνάρτηση σε Python

```
p2.py - C:\Users\vasok\p2.py (3.11.2)
File  Edit  Format  Run  Options  Window  Help
def dec2bF(x, b):
    y = b * x
    a = []
    i = 1
    while y != 0:
        a.append(round(int(y) * 1000) / 1000)
        y = (y - int(y)) * b
        i += 1
    return a
```

- Η μεταβλητή a είναι ένας πίνακας με στοιχεία τα ψηφία του αριθμού που μετατράπηκε.

Παραδείγματα

- Να μετατραπεί ο αριθμός $(11)_{10}$ σε αριθμητικό σύστημα με βάση b=2. Στο Shell έχουμε

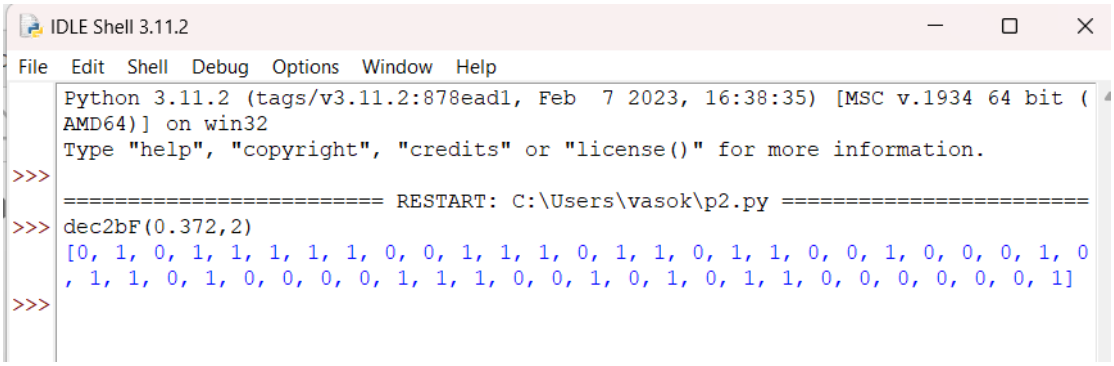
```
IDLE Shell 3.11.2
File  Edit  Shell  Debug  Options  Windov
Python 3.11.2 (tags/v3.11.2:
AMD64)] on win32
Type "help", "copyright", "c
>>>
========== RE
>>> dec2b(11,2)
[1.0, 0.0, 1.0, 1]
>>>
```

Επομένως, έχουμε

$$(11)_{10} = (1011)_2$$

- Να μετατραπεί ο αριθμός $(0.372)_2$ σε αριθμητικό σύστημα με βάση b=2.

  Στο Shell έχουμε



  - Ο αλγόριθμος τερματίζει λόγω προβλημάτων στην αριθμητική ακρίβεια.
  - Ο παραπάνω αριθμός έχει άπειρα ψηφία στο κλασματικό του μέρος.

  Επομένως, έχουμε

$$(0.372)_2 = (0.01011\ldots)_{10}$$

Ασκήσεις

- Να μετατραπεί ο αριθμός $(12345)_{10}$ σε αριθμητικό σύστημα με βάση b=6.
  - Απάντηση: $(12345)_{10} = (133053)_6$
- Να μετατραπεί ο αριθμός $(0.171875)_{10}$ σε αριθμητικό σύστημα με βάση b=8.
  - Απάντηση: $(0.171875)_{10} = (0.13)_8$
- Να μετατραπεί ο αριθμός $(123.125)_{10}$ σε αριθμητικό σύστημα με βάση b=2.
  - Απάντηση: $(123.125)_{10} = (1111011.001)_2$
- Να μετατραπεί ο αριθμός $(17.4)_8$ σε αριθμητικό σύστημα με βάση b=6.
  - Απάντηση: $(17.4)_8 = (23.3)_6$

Ενότητα 6

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 6)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής επίλυσης εξισώσεων και συγκεκριμένα με την γενική επαναληπτική μέθοδο, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική επίλυση εξισώσεων
    - Επαναληπτική μέθοδος
    - Ακρίβεια δεκαδικών ψηφίων
    - Ακρίβεια σημαντικών ψηφίων
    - Επαναληπτική μέθοδος με κριτήριο τερματισμού
    - Ασκήσεις

Επαναληπτική Μέθοδος

- Υπολογισμός μιας ρίζας της εξίσωσης $f(x) = 0$ με χρήση ενός αναδρομικού τύπου της μορφής
$$x_n = g(x_n - 1)$$
- Για να δημιουργήσουμε τον αναδρομικό τύπο πρέπει η εξίσωση $f(x) = 0$ να γραφεί με την μορφή $x = g(x)$.
- Για παράδειγμα, θέλουμε να βρούμε την ρίζα της εξίσωσης $x^2 - 2 = 0$.
- Δημιουργούμε τον αναδρομικό τύπο
$$x^2-2 = 0 \Rightarrow x^2+2x = 2+2x \Rightarrow x(x+2) = 2+2x \Rightarrow x = \frac{2+2x}{x+2}$$
Επομένως, ο αναδρομικός τύπος είναι
$$x_n = \frac{2+2x_{n-1}}{2+x_{n-1}}$$

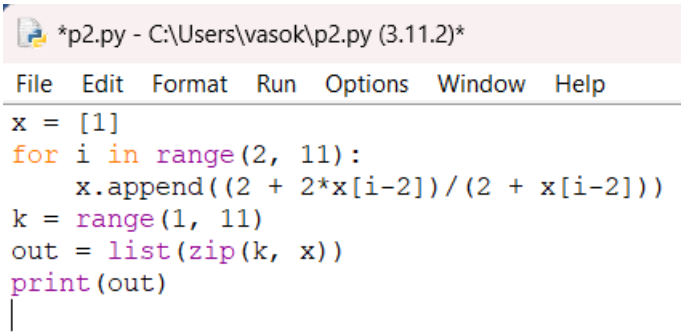
Επαναληπτική Μέθοδος – Παράδειγμα

Δίνεται ο αναδρομικός τύπος

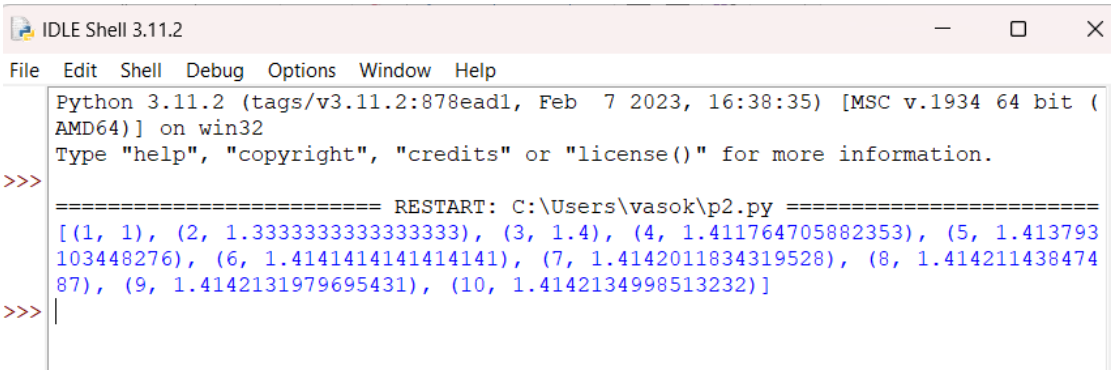
$$x_n = \frac{2+2x_{n-1}}{2+x_{n-1}}$$

με αρχική τιμή $x_1 = 1$. Να βρεθεί η τιμή του x_{10}

- Στον Editor γράφουμε



- Εκτελούμε και στο Shell έχουμε



- Επομένως, $x_{10} = 1.41421349985132$

Ακρίβεια Δεκαδικών Ψηφίων

- Μια αριθμητική μέθοδος μας επιστρέφει σε κάθε βήμα μια προσέγγιση της λύσης.
- Αν η αριθμητική μέθοδος συγκλίνει, τότε η μέθοδος προσεγγίζει την λύση.
- Η ακρίβεια δεκαδικών ψηφίων μας δείχνει πόσα καλά προσεγγίζει η μέθοδος την λύση σε σχέση με το πλήθος των δεκαδικών ψηφίων.
- Η ακρίβεια δεκαδικών ψηφίων δίνεται από τον τύπο

$$|x_n - x_{n-1}| < \frac{1}{2}10^{-k}$$

όπου $k \in N$ το πλήθος των δεκαδικών ψηφίων.

- Λύνουμε ως προς k και έχουμε

$$|x_n - x_{n-1}| < \frac{1}{2}10^{-k} \Rightarrow 2 \cdot |x_n - x_{n-1}| < 10^{-k} \Rightarrow$$

$$\log\left(2 \cdot |x_n - x_{n-1}|\right) < \log\left(10^{-k}\right) \Rightarrow \log\left(2 \cdot |x_n - x_{n-1}|\right) < -k \Rightarrow$$

$$\boxed{k < -\log\left(2 \cdot |x_n - x_{n-1}|\right)}$$

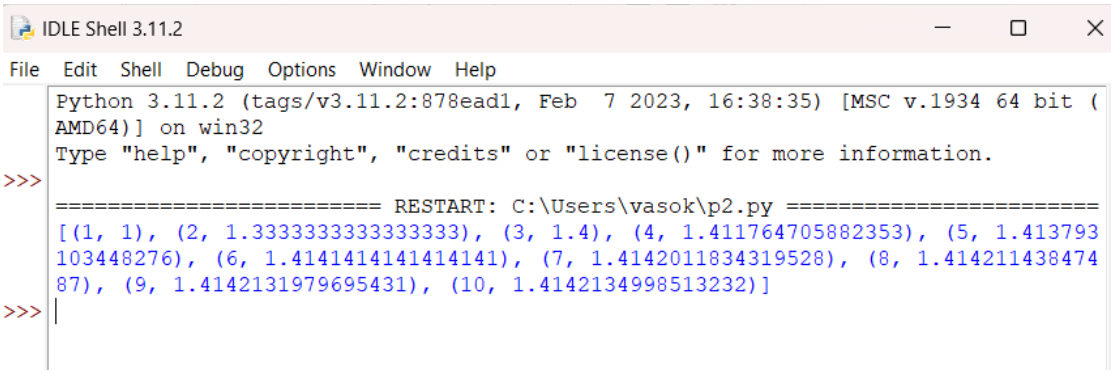
Ακρίβεια Δεκαδικών Ψηφίων – Παράδειγμα

Να βρεθεί η ακρίβεια σε δεκαδικά ψηφία που έχουν οι προσεγγιστικές λύσεις x_{10} και x_8 του τύπου

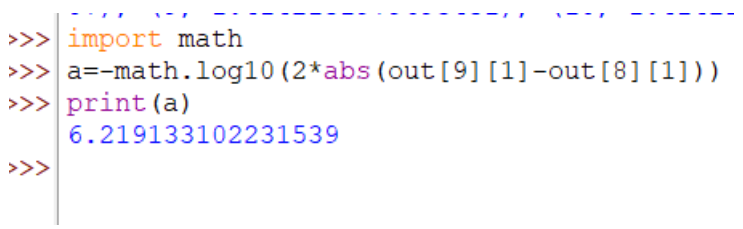
$$x_n = \frac{2 + 2x_{n-1}}{2 + x_{n-1}}$$

με αρχική τιμή $x_1 = 1$.

- Από το προηγούμενο παράδειγμα έχουμε



- Για την ακρίβεια του x_{10} , στο Shell γράφουμε



- Επειδή $k \in N$ και

$$k < -\log\left(2 \cdot |x_{10} - x_9|\right) = 6.21913310223154$$

Θα έχουμε

$$k = 6$$

δηλαδή, η προσεγγιστική λύση x_{10} έχει ακρίβεια 6 δεκαδικών ψηφίων.

- Για την ακρίβεια του x_8 , στο Shell γράφουμε

```
>>> import math
>>> a=-math.log10(2*abs(out[7][1]-out[6][1]))
>>> print(a)
4.688032522107932
>>>
```

- Επειδή $k \in N$ και

$$k < -\log\left(2 \cdot |x_8 - x_7|\right) = 4.68803252210793$$

Θα έχουμε

$$k = 4$$

δηλαδή, η προσεγγιστική λύση x_8 έχει ακρίβεια 4 δεκαδικών ψηφίων.

Ακρίβεια Σημαντικών Ψηφίων

- Μία αριθμητική μέθοδος μας επιστρέφει σε κάθε βήμα μια προσέγγιση της λύσης.
- Αν η αριθμητική μέθοδος συγκλίνει, τότε η μέθοδος προσεγγίζει την λύση.
- Η ακρίβεια σημαντικών ψηφίων μας δείχνει πόσο καλά προσεγγίζει η μέθοδος την λύση σε σχέση με το πλήθος των σημαντικών ψηφίων.
- Η ακρίβεια σημαντικών ψηφίων δίνεται από τον τύπο

$$\left|\frac{x_n - x_{n-1}}{x_{n-1}}\right| < \frac{1}{2}10^{-(k-1)}$$

όπου $k \in N$ το πλήθος των δεκαδικών ψηφίων.

- Λύνουμε ως προς k και έχουμε

$$\left|\frac{x_n - x_{n-1}}{x_{n-1}}\right| < \frac{1}{2}10^{-(k-1)} \Rightarrow 2 \cdot \left|\frac{x_n - x_{n-1}}{x_{n-1}}\right| < 10^{-(k-1)} \Rightarrow$$

$$\log\left(2 \cdot \left|\frac{x_n - x_{n-1}}{x_{n-1}}\right|\right) < \log\left(10^{-(k-1)}\right) \Rightarrow$$

$$\log\left(2 \cdot \left|\frac{x_n - x_{n-1}}{x_{n-1}}\right|\right) < -(k-1) \Rightarrow$$

$$k - 1 < -\log\left(2 \cdot \left|\frac{x_n - x_{n-1}}{x_{n-1}}\right|\right) \Rightarrow$$

$$\boxed{k < -\log\left(2 \cdot \left|\frac{x_n - x_{n-1}}{x_{n-1}}\right|\right) + 1}$$

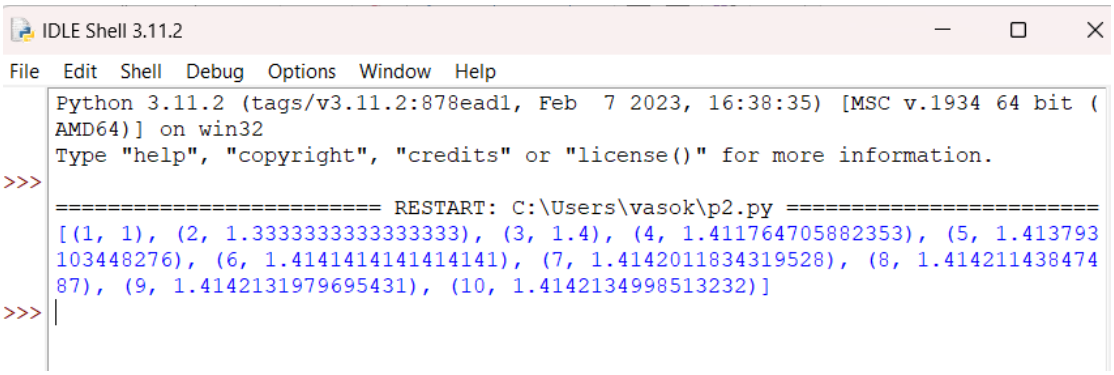
Ακρίβεια Σημαντικών Ψηφίων – Παράδειγμα

Να βρεθεί η ακρίβεια σε σημαντικά ψηφία που έχουν οι προσεγγιστικές λύσεις x_{10} και x_8 του τύπου

$$x_n = \frac{2 + 2x_{n-1}}{2 + x_{n-1}}$$

με αρχική τιμή $x_1 = 1$.

- Από το προηγούμενο παράδειγμα έχουμε



- Για την ακρίβεια του x_{10} , στο Shell γράφουμε

```
>>> import math
>>> a=-math.log10(2*abs(out[9][1]-out[8][1])/out[8][1])+1
>>> print(a)
7.369647988157891
>>>
```

- Επειδή $k \in N$ και

$$k < -\log\left(2 \cdot \left|\frac{x_{10} - x_9}{x_9}\right|\right) + 1 = 7.36964798815789$$

Θα έχουμε

$$k = 7$$

δηλαδή, η προσεγγιστική λύση x_{10} έχει ακρίβεια 7 σημαντικών ψηφίων.

- Για την ακρίβεια του x_8 , στο Shell γράφουμε

```
>>> import math
>>> a=-math.log10(2*abs(out[7][1]-out[6][1])/out[6][1])+1
>>> print(a)
5.838543718442396
>>>
```

- Επειδή $k \in N$ και

$$k < -\log\left(2 \cdot \left|\frac{x_8 - x_7}{x_7}\right|\right) + 1 = 5.8385437184424$$

Θα έχουμε

$$k = 5$$

δηλαδή, η προσεγγιστική λύση x_8 έχει ακρίβεια 5 σημαντικών ψηφίων.

Επαναληπτική Μέθοδος με κριτήριο τερματισμού

Να βρεθούν οι τιμές των x_n μέχρι να ισχύει το κριτήριο τερματισμού

$$|x_n - x_{n-1}| < \frac{1}{2}10^{-4}$$

δηλαδή, η προσεγγιστική λύση να έχει ακρίβεια τεσσάρων δεκαδικών ψηφίων.

Οι τιμές των x_n δίνονται από τον τύπο

$$x_n = \frac{2 + 2x_{n-1}}{2 + x_{n-1}}$$

με αρχική τιμή $x_1 = 1$.

- Στον Editor γράφουμε

```
p2.py - C:\Users\vasok\p2.py (3.11.2)
File  Edit  Format  Run  Options  Window  Help
x = [0]*10
x[0] = 1
i = 1
done = False

while not done:
    x[i] = (2 + 2*x[i-1]) / (2 + x[i-1])
    if abs(x[i] - x[i-1]) < 1/2*10**(-4):
        done = True
    else:
        i += 1

k = list(range(1, len(x)+1))
out = [[k[i], x[i]] for i in range(len(k))]
print(out)
```

- Εκτελούμε και στο Shell έχουμε

```
>>>
========== RESTART: C:\Users\vasok\p2.py ==========
[[1, 1], [2, 1.3333333333333333], [3, 1.4], [4, 1.411764705882353], [5, 1.413793
103448276], [6, 1.4141414141414141], [7, 1.4142011834319528], [8, 1.414211438474
87], [9, 0], [10, 0]]
>>>
```

Άσκηση

- Δίνεται ο τύπος

$$x_n = \frac{3 + x_{n-1}}{1 + x_{n-1}}$$

με αρχική τιμή $x_1 = 2$. Να βρεθεί η τιμή του x_{20}

  - Απάντηση: $x_{20} = 1.7320508075655$
- Να βρεθεί η ακρίβεια σε δεκαδικά ψηφία που έχουν οι προσεγγιστικές λύσεις x_{15} και x_{20} .
  - Απάντηση: 7 και 10 δεκαδικά ψηφία αντίστοιχα
- Να βρεθεί η ακρίβεια σε σημαντικά ψηφία που έχουν οι προσεγγιστικές λύσεις x_{15} και x_{20} .
  - Απάντηση: 8 και 11 σημαντικά ψηφία αντίστοιχα

Ενότητα 7

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 7)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής επίλυσης εξισώσεων και ειδικότερα με τη μέθοδο Διχοτόμησης, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική επίλυση εξισώσεων
    - Μέθοδος διχοτόμησης
    - Μέθοδος διχοτόμησης – αλγόριθμος
    - Μέθοδος διχοτόμησης – υλοποίηση
    - Μέθοδος διχοτόμησης – βελτίωση
    - Μέθοδος διχοτόμησης – παραδείγματα
    - Πλήθος επαναλήψεων – ακρίβεια
    - Ασκήσεις

Μέθοδος Διχοτόμησης

- Υπολογισμός μιας ρίζας της εξίσωσης $f(x) = 0$ στο διάστημα [a,b] όπου ισχύει $f(a) \cdot f(b) < 0$.
- Είσοδος
    - Η συνάρτηση $f(x)$
    - Το διάστημα [a,b]
    - Η ακρίβεια σε δεκαδικά ψηφία $(tol = \frac{1}{2}10^{-k})$
    - Ο μέγιστος αριθμός επαναλήψεων
- Έξοδος
    - Η προσεγγιστική ρίζα
    - Μήνυμα αποτυχίας

Μέθοδος Διχοτόμησης – Αλγόριθμος

ΕΙΣΟΔΟΣ: $f(x)$, a, b, tol, n

ΒΗΜΑ 1 Αν $f(a) \cdot f(b) < 0$ πήγαινε στο βήμα 2

Διαφορετικά

ΕΞΟΔΟΣ: Δεν ισχύει το Θ. Bolzano στο αρχικό διάστημα, τερμάτισε

ΒΗΜΑ 2 Θέσε i=1

ΒΗΜΑ 3 Όταν i<=n εκτέλεσε τα βήματα 3-6

ΒΗΜΑ 4 Θέσε $x = \frac{a+b}{2}$

ΒΗΜΑ 5 Αν $f(x) = 0$ ή $\frac{b-a}{2} < tol$ τότε

ΕΞΟΔΟΣ: το x είναι η λύση, τερμάτισε

ΒΗΜΑ 6 $f(a) \cdot f(\chi) > 0$ τότε

Θέσε a=x

Διαφορετικά

Θέσε b=x

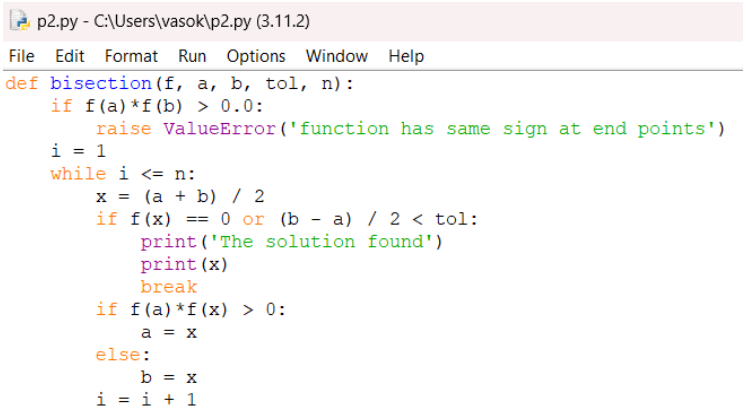
ΒΗΜΑ 7 Θέσε i=i+1

ΒΗΜΑ 8 ΕΞΟΔΟΣ: Η μέθοδος εξάντλησε όλες τις επαναλήψεις, τερμάτισε

- Η μέθοδος διχοτόμησης πρέπει να εφαρμόζεται σε διαστήματα στα οποία υπάρχει ακριβώς μια ρίζα.
- Τον παραπάνω αλγόριθμο αν τον εφαρμόσουμε σε διαστήματα με καμία ή 2 ή 4 ρίζες (γενικά άρτιο αριθμό ριζών) θα σταματήσει από το Βήμα 1.
- Τον παραπάνω αλγόριθμο αν τον εφαρμόσουμε σε διάστημα με 3 ή 5 ρίζες (γενικά περιττό αριθμό ριζών) θα βρει μια από όλες τις ρίζες.

Μέθοδος Διχοτόμησης – Υλοποίηση

- Υλοποίηση της μεθόδου διχοτόμησης σε συνάρτηση Python με while και break



Μέθοδος Διχοτόμησης – Βελτίωση

- Είσοδος παραμέτρων από τον χρήστη – Δημιουργία συνάρτησης με έξοδο πίνακα def bisect(f, a, b, tol, n):
  - Η συνάρτηση f(x)
  - Το διάστημα [a,b]
  - Η ακρίβεια σε δεκαδικά ψηφία (tol)
  - Ο μέγιστος αριθμός επαναλήψεων (n)
- Περισσότερες πληροφορίες στην έξοδο
  - Αριθμός βημάτων
  - Νέο διάστημα σε κάθε βήμα
  - Τιμή της ρίζας και της συνάρτησης f(x) σε κάθε βήμα

Μέθοδος Διχοτόμησης – Υλοποίηση (1)

- Υλοποίηση της μεθόδου διχοτόμησης σε συνάρτηση Python

p3.py - C:/Users/vasok/p3.py (3.11.2)

File Edit Format Run Options Window Help

```
def bisect(f, a, b, tol, n):
    if f(a)*f(b) > 0.0:
        raise ValueError('function has same sign at end points')

    a_ = [a]
    b_ = [b]
    x = [(a+b)/2]

    for i in range(n):
        if f(x[i]) == 0 or (b_[i]-a_[i])/2 < tol:
            print('Bisection method has converged')
            break

        if f(a_[i])*f(x[i]) > 0:
            a_.append(x[i])
            b_.append(b_[i])
        else:
            a_.append(a_[i])
            b_.append(x[i])

        x.append((a_[i+1]+b_[i+1])/2)

    if i == n-1:
        print('Zero not found to desired tolerance')

    k = list(range(len(x)))
    out = list(zip(k, a_, b_, x, [f(i) for i in x]))

    return out
def f(x):
    return x**3 - 2*x - 5

out = bisect(f, 1, 3, 1/2*10**(-4), 50)
print(out)
```

Μέθοδος Διχοτόμησης – Παράδειγμα 1

Να βρεθεί η ρίζα της συνάρτησης

$$f(x) = x^3 - 2x - 5$$

με την μέθοδο Διχοτόμησης, στο διάστημα [1,3] με ακρίβεια 4 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.

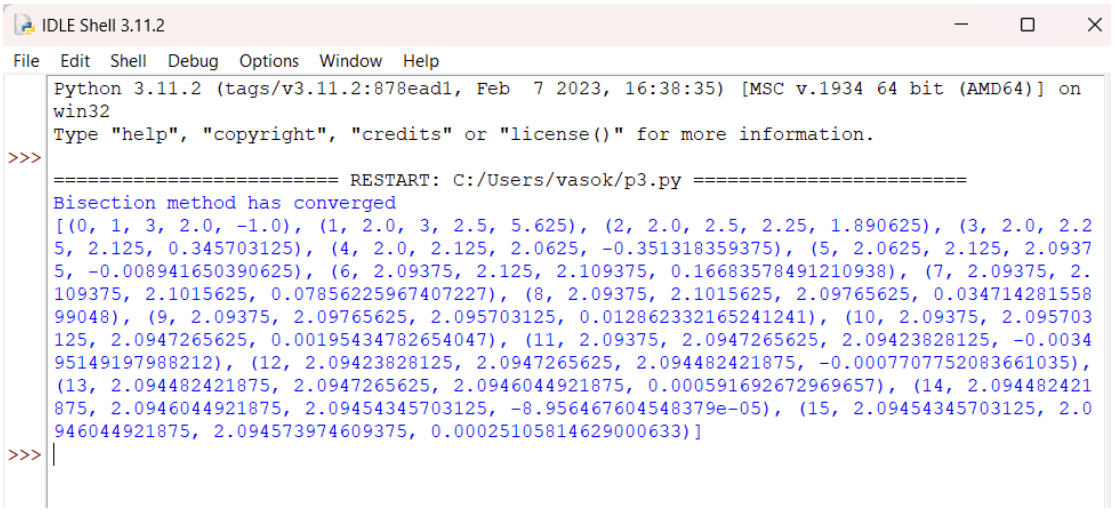
Σε Python θα έχουμε

- Ορίζουμε την συνάρτηση f(x)

```
def f(x):
    return x**3 - 2*x - 5
```

- Καλούμε την συνάρτηση bisect με τα κατάλληλα ορίσματα

```
out = bisect(f, 1, 3, 1/2*10**(-4), 50)
```



Από τον πίνακα out τον οποίο επιστρέφει η συνάρτηση bisect παρατηρούμε τα εξής:

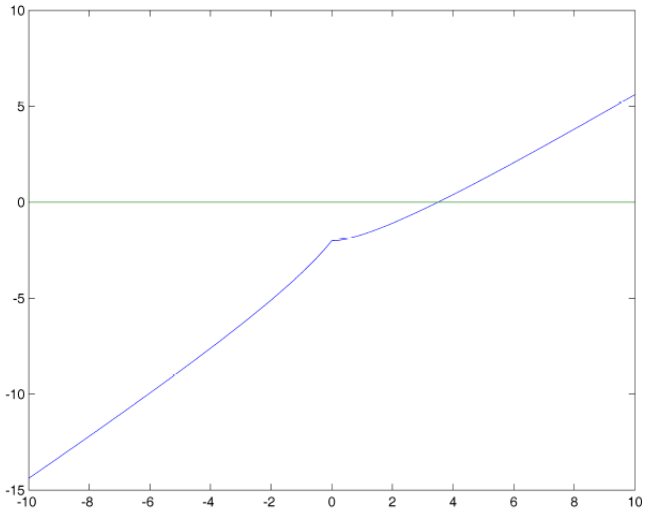
- Για τον υπολογισμό της ρίζας εκτελέστηκαν 16 επαναλήψεις
- Το αριστερό άκρο του διαστήματος $\alpha_{15} = 2.09454345703125$ (δεύτερη στήλη, τελευταία σειρά)
- Το δεξιό άκρο του διαστήματος $b_{15} = 2.0946044921875$ (τρίτη στήλη, τελευταία σειρά)
- Η προσεγγιστική τιμή της ρίζας είναι $\chi_{15} = 2.094573974609375$ (τέταρτη στήλη, τελευταία σειρά)

- Η τιμή της συνάρτησης είναι f(χ_{15})= 0.00025105814629000633 (πέμπτη στήλη, τελευταία σειρά)
- Επομένως, η λύση στο πρόβλημα είναι $\chi_{15} = 2.094573974609375$
- Καλέστε την συνάρτηση με ακρίβεια 8 δεκαδικών ψηφίων
- Καλέστε την συνάρτηση με διάστημα [-30,30]
- Καλέστη την συνάρτηση με διάστημα [10,30]
- Τι παρατηρείτε για τις παραπάνω κλήσεις της συνάρτησης bisect;

Μέθοδος Διχοτόμησης – Παράδειγμα 2

Να βρεθεί η ρίζα της συνάρτησης $f(x) = x - \ln(|x| + 1) - 2$ με τη μέθοδο Διχοτόμησης, με ακρίβεια 4 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.

- Για να βρούμε τη ρίζα της συνάρτησης θα πρέπει πρώτα να βρούμε ένα κατάλληλο διάστημα με την βοήθεια της γραφικής παράστασης της συνάρτησης.
- Σχεδιάζουμε την γραφική παράσταση της συνάρτησης στο διάστημα [-10,10] και επιλέγουμε το κατάλληλο διάστημα **(Στα πλαίσια του μαθήματος, κατάλληλο διάστημα θα θεωρούμε το διάστημα της μορφής** [α,α+1] **με** $a \in Z$**).**
- Από την γραφική παράσταση της συνάρτησης επιλέγουμε το διάστημα [3,4].



Σχήμα: Γραφική παράσταση της συνάρτησης

$f(x) = x - \ln(|x| + 1) - 2$

Σε Python θα έχουμε

- Ορίζουμε την συνάρτηση f(x)

```
import math
def f(x):
    return x-math.log(abs(x)+1)-2
```

- Σχεδιάζουμε την συνάρτηση f(x) ενεργοποιώντας το πλέγμα

```
import numpy as np
import matplotlib.pyplot as plt
x=range(-10,11)
y=[f(xi) for xi in x]
plt.plot(x,y)
plt.grid(True)
plt.show
```

- Καλούμε την συνάρτηση bisect με τα κατάλληλα ορίσματα

```
out = bisect(f, 3, 4, 1/2*10**(-4), 50)
```

- Η λύση είναι $\chi_{15} = 3.50521850585938$

Πλήθος Επαναλήψεων – Ακρίβεια

- Το πλήθος των επαναλήψεων ($n \in N$) της μεθόδου Διχοτόμησης συνδέεται με την ακρίβεια της λύσης σε δεκαδικά ψηφία ($\kappa \in N$) με τον τύπο

$$\frac{b-a}{2^n} < tol \Longrightarrow \frac{b-a}{2^n} < \frac{1}{2} \cdot 10^{-k}$$

- Επομένως, μπορούμε να βρούμε είτε το n γνωρίζοντας το k, είτε το k γνωρίζοντας το n.
- Για να υπολογίσουμε το πλήθος των επαναλήψεων (n) της μεθόδου Διχοτόμησης με ακρίβεια k δεκαδικών ψηφίων στο διάστημα [a,b], λύνουμε ως προς n, επομένως έχουμε

$$\frac{b-a}{2^n} < \frac{1}{2} \cdot 10^{-k} \Longrightarrow \frac{2^n}{b-a} > 2 \cdot 10^{k} \Longrightarrow$$

$$2^n > (b-a) \cdot 2 \cdot 10^{k} \Longrightarrow$$

$$\boxed{n > \log_2\left((b-a) \cdot 2 \cdot 10^{k}\right)}$$

- Ενώ, για να υπολογίσουμε την ακρίβεια των δεκαδικών ψηφίων (k) της μεθόδου Διχοτόμησης αν εκτελεστούν n επαναλήψεις στο διάστημα [a,b], λύνουμε ως προς k, επομένως έχουμε

$$\frac{b-a}{2^n} < \frac{1}{2}\cdot 10^{-k} \Longrightarrow 2\cdot 10^k < \frac{2^n}{b-a} > \Longrightarrow$$

$$10^k < \frac{2^n}{2\cdot(b-a)} \Longrightarrow$$

$$\boxed{k < \log\left(\frac{2^n}{2\cdot(b-a)}\right)}$$

Πλήθος Επαναλήψεων – Παράδειγμα

Να βρεθεί το πλήθος των επαναλήψεων που θα εκτελέσει η μέθοδος Διχοτόμησης στο διάστημα [1,3] με ακρίβεια 4 δεκαδικών ψηφίων.

- Σε Python θα έχουμε

```
>>> import math
>>> a=math.log2((3-1)*2*10**4)
>>> print(a)
15.287712379549449
>>>
```

- Επειδή
  $n > log_2\left((b-a)\cdot 2\cdot 10^k\right) = log_2\left((3-1)\cdot 2\cdot 10^4\right) = 15.2877123795494$ και $n \in N$
- Θα έχουμε n=16.

Ακρίβεια – Παράδειγμα

Να βρεθεί η ακρίβεια σε δεκαδικά ψηφία της προσεγγιστικής λύσης χ_{10} της μεθόδου Διχοτόμησης στο διάστημα [1,3].

- Σε Python θα έχουμε

```
>>> import math
>>> a=math.log10((2**10)/(2*(3-1)))
>>> print(a)
2.4082399653118496
>>>
```

- Επειδή $k < \log\left(\frac{2^n}{2\cdot(b-a)}\right) = \log\left(\frac{2^{10}}{2\cdot(3-1)}\right) = 2.4082399653118496$ και $k \in N$
- Θα έχουμε k=2.

Άσκηση 1

Δίνεται η συνάρτηση

$$f(x) = e^x + 5x - 10$$

- Να βρεθεί η ρίζα της συνάρτησης με τη μέθοδο Διχοτόμησης, στο διάστημα [0,4] με ακρίβεια 6 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.
  - Απάντηση: $x_{23} = 1.28039121627808$
- Να βρεθεί η ακρίβεια σε δεκαδικά ψηφία που έχουν οι προσεγγιστικές λύσεις x_{10} και x_{20} .
  - Απάντηση: 2 και 5 δεκαδικά ψηφία αντίστοιχα
- Να βρεθεί η ακρίβεια σε σημαντικά ψηφία που έχουν οι προσεγγιστικές λύσεις x_{10} και x_{20} .
  - Απάντηση: 3 και 6 σημαντικά ψηφία αντίστοιχα

Άσκηση 2

Να βρεθεί το πλήθος των επαναλήψεων που θα εκτελέσει η μέθοδος Διχοτόμησης στο διάστημα [0,4] με ακρίβεια 10 δεκαδικών ψηφίων.

- Απάντηση: n=37 επαναλήψεις

Άσκηση 3

Να βρεθεί η ακρίβεια σε δεκαδικά ψηφία της προσεγγιστικής λύσης x_{20} της μεθόδου Διχοτόμησης στο διάστημα [0,4].

- Απάντηση: k=5 η ακρίβεια των δεκαδικών ψηφίων

Ενότητα 8

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 8)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής επίλυσης εξισώσεων και ειδικότερα με την επαναληπτική μέθοδο Newton, η οποία υλοποιείται προγραμματιστικά σε Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική επίλυση εξισώσεων
    - Μέθοδος Newton
    - Μέθοδος Newton – Αλγόριθμος
    - Μέθοδος Newton - Υλοποίηση
    - Μέθοδος Newton - Παραδείγματα
    - Ασκήσεις

Μέθοδος Newton

- Υπολογισμός μιας ρίζας της εξίσωσης $f(x) = 0$ με αρχική τιμή x_0 .
- Είσοδος
    - Η συνάρτηση $f(x)$
    - Η παράγωγος συνάρτηση $f'(x)$
    - Η αρχική τιμή x_0
    - Η ακρίβεια σε δεκαδικά ψηφία $(tol = \frac{1}{2}10^{-k})$
    - Ο μέγιστος αριθμός επαναλήψεων
- Έξοδος
    - Η προσεγγιστική ρίζα
    - Μήνυμα αποτυχίας

Μέθοδος Newton – Αλγόριθμος

ΕΙΣΟΔΟΣ: $f(x)$, $f'(x)$, x_0, tol, n

ΒΗΜΑ 1 Θέσε i=2, x(1)= x_0

ΒΗΜΑ 2 Όταν i<=n εκτέλεσε τα βήματα 3 – 5

ΒΗΜΑ 3 Θέσε $x(i) = x(i-1) - \frac{f(x(i-1))}{f'(x(i-1))}$

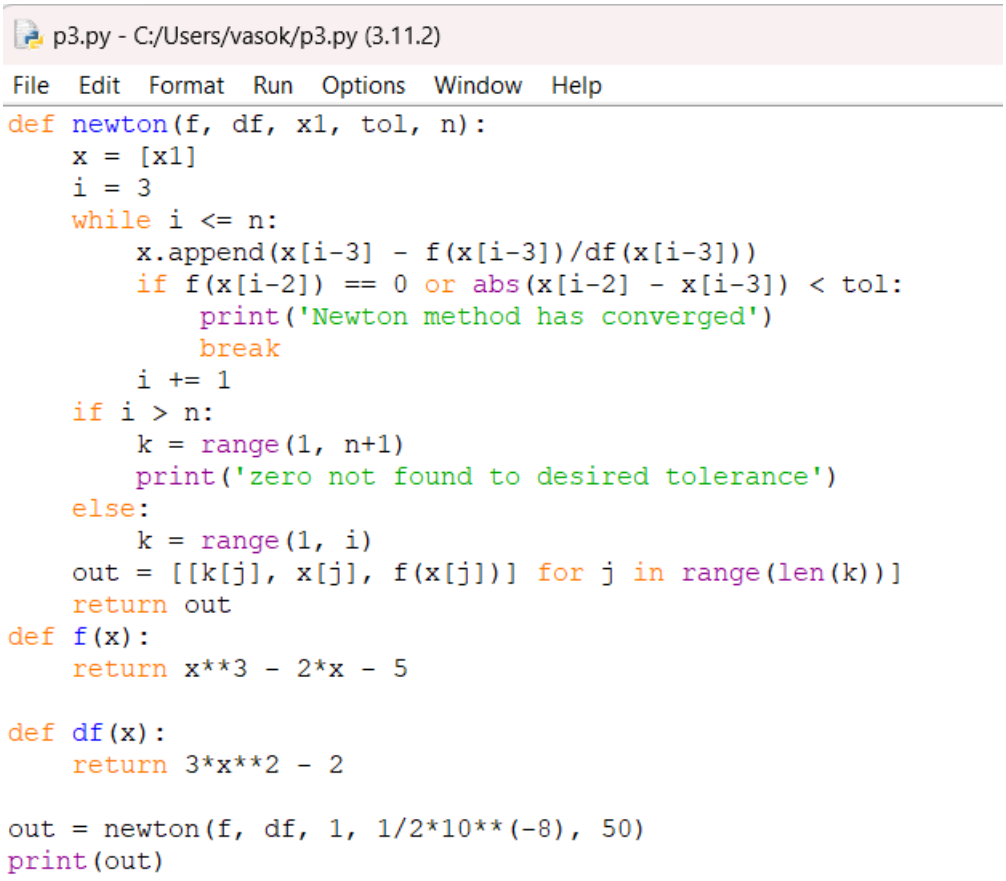
ΒΗΜΑ 4 Αν f(x(i))=0 ή |x(i)-x(i-1)|<tol τότε ΕΞΟΔΟΣ: το x(i) είναι η λύση, τερμάτισε

ΒΗΜΑ 5 Θέσε i=i+1

ΒΗΜΑ 6 ΕΞΟΔΟΣ: Η μέθοδος εξάντλησε όλες τις επαναλήψεις, τερμάτισε

Μέθοδος Newton – Υλοποίηση

- Υλοποίηση της μεθόδου Newton σε συνάρτηση Python

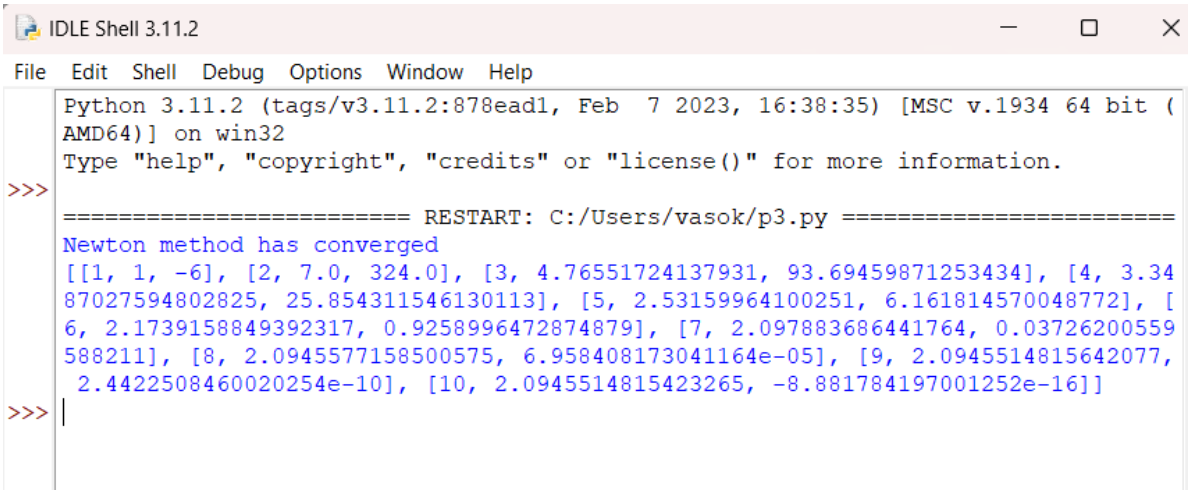


Μέθοδος Newton – Παράδειγμα 1

Να βρεθεί η ρίζα της συνάρτησης $f(x) = x^3 - 2x - 5$ με τη μέθοδο Newton, με αρχική τιμή 1 με ακρίβεια 8 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.

Σε Python θα έχουμε

- Ορίζουμε την συνάρτηση $f(x)$
  def f(x):
  *return x**3 - 2*x - 5*
- Ορίζουμε την παράγωγο συνάρτηση $f'(x)$
  def df(x):
  *return 3*x**2 - 2*
- Καλούμε την συνάρτηση Newton με τα κατάλληλα ορίσματα
  *out = newton(f, df, 1, 1/2*10**(-8), 50)*



Από τον πίνακα out τον οποίο επιστρέφει η συνάρτηση newton παρατηρούμε τα εξής:

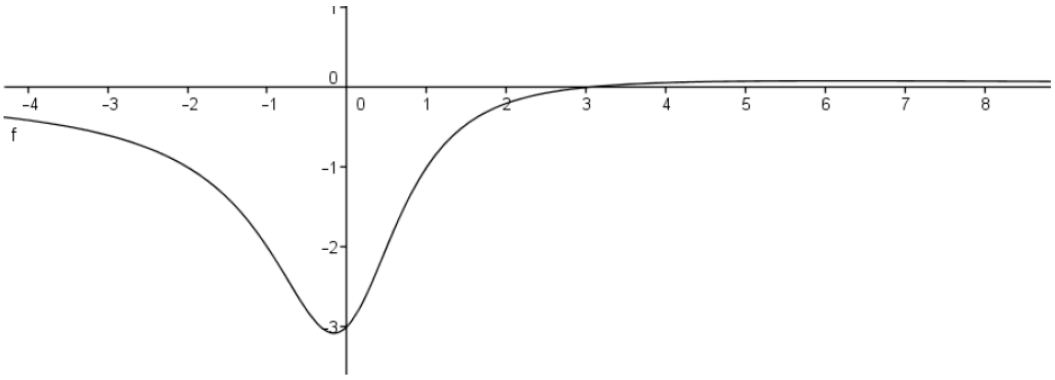
- Για τον υπολογισμό της ρίζας εκτελέστηκαν 10 επαναλήψεις.
- Η προσεγγιστική τιμή της ρίζας είναι $x_{10} = 2.0945514815423265$ (τελευταία γραμμή, δεύτερη στήλη)
- Η τιμή της συνάρτησης είναι $f(x_{10}) = -8.881784197001252e - 16$ (τελευταία γραμμή, τρίτη στήλη)
- Επομένως, η λύση στο πρόβλημα είναι $x_{10} = 2.0945514815423265$

Μέθοδος Newton – Παράδειγμα 2

Ειδική περίπτωση

Να βρεθεί η ρίζα της συνάρτησης $f(x) = \frac{x-3}{x^2+1}$ με τη μέθοδο Newton, με ακρίβεια 8 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.

- Η παραπάνω συνάρτηση έχει προφανή ρίζα το 3
- Έχει ιδιόμορφη γραφική παράσταση



- Πλησιάζει στο μηδέν όταν το $\chi \to \infty$ και $\chi \to -\infty$

Επομένως, σε Python θα έχουμε

- Ορίζουμε την συνάρτηση $f(x)$

```
def f(x):
    return (x-3)/(x**2+1)
```

- Ορίζουμε την παράγωγο συνάρτηση $f'(x)$

```
def df(x):
    return (-x**2+6*x+1)/((x**2+1)**2)
```

- Καλούμε την συνάρτηση newton με τα κατάλληλα ορίσματα
  - Με αρχική τιμή 2

    out = newton(f, df, 2, 1/2*10**(-8), 50)

    μας επιστρέφει $x_7 = 3$
  - Ενώ, με αρχική τιμή 5

    out = newton(f, df, 5, 1/2*10**(-8), 50)

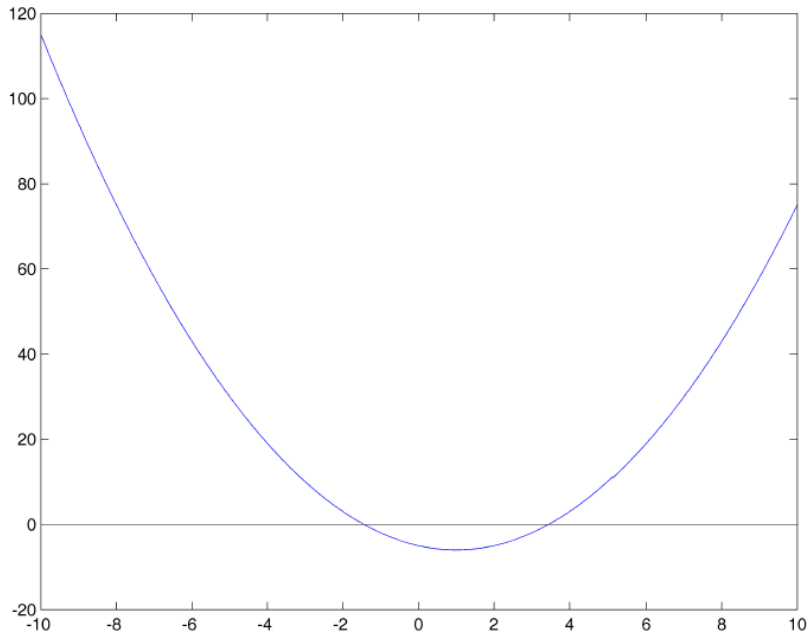
    μας επιστρέφει $x_{50} = -660095034734379$, δηλαδή, εκτελέστηκε ο μέγιστος αριθμός των επαναλήψεων

Μέθοδος Newton – Παράδειγμα 3

Να βρεθεί η θετική ρίζα της συνάρτησης $f(x) = x^2 - 2x - 5$ με την μέθοδο Newton, με ακρίβεια 8 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.

- Για να βρούμε την θετική ρίζα της συνάρτησης θα πρέπει πρώτα να βρούμε μια κατάλληλη αρχική τιμή με την βοήθεια της γραφικής παράστασης της συνάρτησης.

- Σχεδιάζουμε την γραφική παράσταση της συνάρτησης στο διάστημα [-10,10] και επιλέγουμε την κατάλληλη αρχική τιμή **(Στα πλαίσια του μαθήματος, κατάλληλη αρχική τιμή θα θεωρούμε τη κοντινότερη στη ρίζα τιμή)**.
- Από την γραφική παράσταση της συνάρτησης επιλέγουμε αρχική τιμή το 4.



Σχήμα: Γραφική παράσταση της συνάρτησης $f(x) = x^2 - 2x - 5$

Σε Python θα έχουμε

- Ορίζουμε την συνάρτηση $f(x)$

```
def f(x):
    return x**2-2*x-5
```

- Σχεδιάζουμε την συχνότητα $f(x)$ ενεργοποιώντας το πλέγμα

```
import numpy as np
import matplotlib.pyplot as plt
x=range(-10,11)
y=[f(xi) for xi in x]
plt.plot(x,y)
plt.grid(True)
plt.show
```

- Ορίζουμε την παράγωγο συνάρτηση $f'(x)$
  def df(x):
    return 2*x-2
- Καλούμε την συνάρτηση newton με τα κατάλληλα ορίσματα
  out = newton(f, df, 4, 1/2*10**(-8), 50)
- Η θετική ρίζα είναι $x_6 = 3.44948974278318$

Άσκηση 1

Δίνεται η συνάρτηση

$$f(x) = e^x + 5x - 10$$

- Να βρεθεί η ρίζα της συνάρτησης με τη μέθοδο Newton, με αρχική τιμή $x_1 = 0$ με ακρίβεια 12 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.
  - Απάντηση: $x_7 = 1.28039085035359$
- Να βρεθεί η ακρίβεια σε δεκαδικά ψηφία που έχουν οι προσεγγιστικές λύσεις x_6 και x_7 .
  - Απάντηση: 9 και 15 δεκαδικά ψηφία αντίστοιχα
- Να βρεθεί η ακρίβεια σε σημαντικά ψηφία που έχουν οι προσεγγιστικές λύσεις x_6 και x_7 .
  - Απάντηση: 10 και 16 σημαντικά ψηφία αντίστοιχα

Ενότητα 9

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 9)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της παρεμβολής και τις διάφορες μεθόδους υλοποίησης της μέσα από την Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Πολυωνυμική παρεμβολή

- Γενική μέθοδος
- Παρεμβολή – Συνάρτηση interp1d
- Παρεμβολή – Συνάρτηση np.polyfit(x,y,n)

2. Polynomial Functions

Πολυωνυμική Παρεμβολή

- Υπολογισμός πολυωνύμου παρεμβολής n βαθμού που διέρχεται από n+1 σημεία.
- Είσοδος
  - Τα n+1 σημεία
- Έξοδος
  - Το πολυώνυμο παρεμβολής n βαθμού

Γενική Μέθοδος

Έστω τα σημεία

$$(x_i, y_i) \quad i = 0, 1, \ldots, n$$

- Υπολογισμός του πίνακα Vandermonde

$$V = \begin{bmatrix} x_0^n & x_0^{n-1} & \cdots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ \vdots & \vdots & \ddots & & \vdots \\ x_n^n & x_n^{n-1} & \cdots & x_n & 1 \end{bmatrix}$$

- Υπολογισμός των συντελεστών του πολυωνύμου με βάση τον τύπο

$$p = V^{-1} \cdot y$$

όπου y το διάνυσμα στήλη των τεταγμένων των σημείων.

Γενική Μέθοδος – Παράδειγμα

Έστω τα σημεία Α(1,1), Β(2,3) και Γ(3,2). Να βρεθεί το πολυώνυμο παρεμβολής το οποίο διέρχεται από τα παραπάνω σημεία (με την γενική μέθοδο).

- Υπολογισμός του πίνακα Vandermonde

$$V = \begin{bmatrix} 1^2 & 1^1 & 1 \\ 2^2 & 2^1 & 1 \\ 3^2 & 3^1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{bmatrix}$$

- Υπολογισμός των συντελεστών του πολυωνύμου, σε Python θα έχουμε

p3.py - C:/Users/vasok/p3.py (3.11.2)

File Edit Format Run Options Window Help

```
import numpy as np

y = np.array([[1], [3], [2]])
V = np.array([[1, 1, 1], [4, 2, 1], [9, 3, 1]])
p = np.linalg.inv(V)@y
print(p)
```

- Οι παραπάνω εντολές μας επιστρέφουν

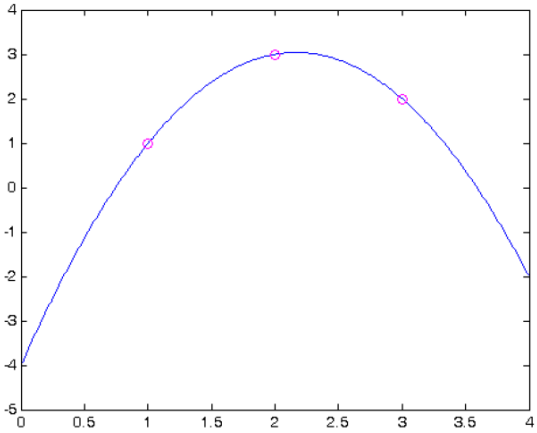
IDLE Shell 3.11.2

File Edit Shell Debug Options Win

```
Python 3.11.2 (tags/v3.11.
AMD64)] on win32
Type "help", "copyright",
>>>
==========================
[[-1.5]
 [ 6.5]
 [-4. ]]
>>>
```

- Τα οποία είναι οι συντελεστές του πολυωνύμου, δηλαδή

$$p(x) = -1.5x^2 + 6.5x - 4$$



Σχήμα: Πολυωνυμική Παρεμβολή

Γενική Μέθοδος – Άσκηση

Έστω τα σημεία Α(0,-4), Β(1,-2), Γ(3,14) και Δ(4,40).

- Να βρεθεί το πολυώνυμο παρεμβολής το οποίο διέρχεται από τα παραπάνω σημεία (με τη γενική μέθοδο).
  - Απάντηση:

$$\left(V = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 27 & 9 & 3 & 1 \\ 64 & 16 & 4 & 1 \end{bmatrix}, \quad p(x) = x^3 - 2x^2 + 3x - 4 \right)$$

Παρεμβολή – Συνάρτηση interp1d

- Η πολυωνυμική παρεμβολή μπορεί να υλοποιηθεί με τη συνάρτηση interp1d.
- f=interp1d (x,y)
  x_new=1.5
  y_new=f(x_new)
  - x,y οι τετμημένες και οι τεταγμένες των δοθέντων σημείων
  - x_new η παρεμβαλόμενη τιμή
  - y_new τελική τιμή με παρεμβαλόμενη τιμή
  - η παραπάνω σύνταξη υλοποιεί γραμμική παρεμβολή
- Η συνάρτηση np.polyfit(x,y,3) βρίσκει τους συντελεστές του πολυωνύμου και η np.poly1d() υλοποιεί παρεμβολή με πολυώνυμα τρίτου βαθμού (κυβική παρεμβολή)
- Η συνάρτηση CubicSpiline(x,y) της βιβλιοθήκης scipy.interpolate υλοποιεί παρεμβολή με κυβικά splines

Συνάρτηση interp1 – Παράδειγμα

Έστω τα σημεία Α(1,1), Β(2,3) και Γ(3,2). Να βρεθούν οι τιμές του y

- Για χ=1.5 με γραμμική παρεμβολή

- Για x=2.9 με γραμμική παρεμβολή, με κυβική παρεμβολή και με παρεμβολή με κυβικά splines

Σε Python θα έχουμε

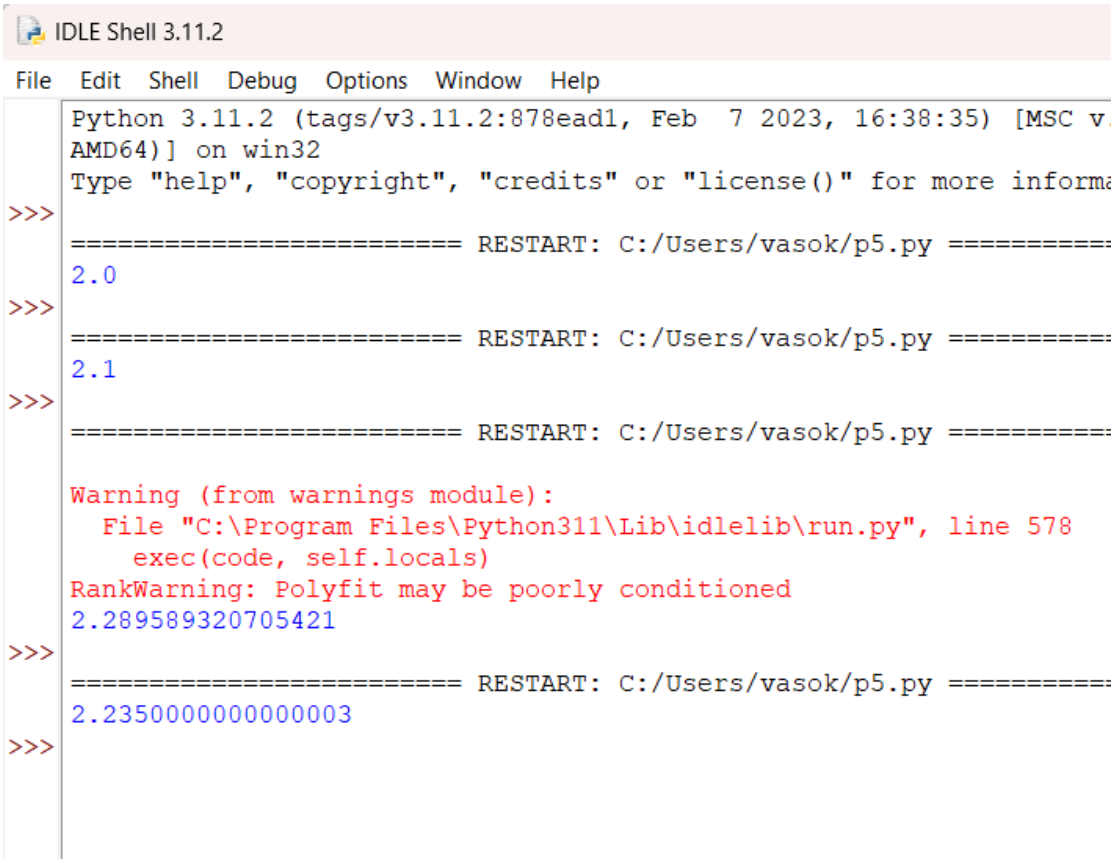
```
import numpy as np
x=np.array([1,2,3])
y=np.array([1,3,2])
xi=1.5
yi=np.interp(xi,x,y)
print(yi)
```

```
import numpy as np
x=np.array([1,2,3])
y=np.array([1,3,2])
xi=2.9
yi=np.interp(xi,x,y)
print(yi)
```

```
import numpy as np
x=np.array([1,2,3])
y=np.array([1,3,2])
f=np.polyfit(x,y,3)
p=np.poly1d(f)
xi=2.9
yi=p(xi)
print(yi)
```

```
import numpy as np
from scipy.interpolate import CubicSpline
x=np.array([1,2,3])
y=np.array([1,3,2])
f=CubicSpline(x,y)
y=f(2.9)
print(y)
```

Εκτελούμε και στο Shell έχουμε



Συνάρτηση interp1d – Άσκηση

Δίνεται ο παρακάτω πίνακας τιμών

x	2	3	4	5	6
y	-1	3	5	2	7

Να βρεθούν οι τιμές του y

- Για x=2.7 με γραμμική παρεμβολή
  - Απάντηση: (y=1.8)
- Για x=5.8 με γραμμική παρεμβολή και με κυβική παρεμβολή
  - Απάντηση: (y=5.9 και y= 5.394)

Παρεμβολή – Συνάρτηση np.polyfit(x,y,n)

- Εύρεση πολυωνύμου παρεμβολής με την συνάρτηση np.polyfit(x,y,n) .
- np.polyfit(x,y,n)
  - x,y οι τετμημένες και οι τεταγμένες των δοθέντων σημείων
  - n ο βαθμός του πολυωνύμου παρεμβολής
  - Στην περίπτωση που ο βαθμός n δεν είναι ίσος με το πλήθος των σημείων -1, τότε η συνάρτηση υλοποιεί προσέγγιση.

Συνάρτηση np.polyfit(x,y,n) – ΠΑΡΑΔΕΙΓΜΑ

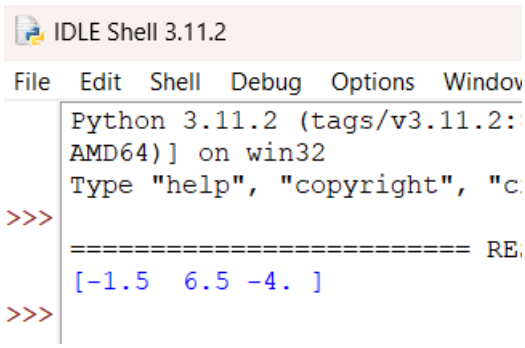
Έστω τα σημεία Α(1,2), Β(2,3) και Γ(3,2).

Να βρεθεί το πολυώνυμο παρεμβολής που διέρχεται από τα παραπάνω σημεία.

Σε Python θα έχουμε

```
import numpy as np
x=np.array([1,2,3])
y=np.array([1,3,2])
f=np.polyfit(x,y,2)
print(f)
```

- Οι παραπάνω εντολές μας επιστρέφουν



- Τα οποία είναι οι συντελεστές του πολυωνύμου, δηλαδή,

$$p(x) = -1.5x^2 + 6.5x - 4$$

Polynomial Functions

- np.polyval(p,x) υπολογισμός της τιμής του πολυωνύμου p για την δοθείσα τιμή x
- np.polyder(p) υπολογισμός της παραγώγου συνάρτησης (πολυώνυμο) του πολυωνύμου p
- np.polyint(p) υπολογισμός της αρχικής συνάρτησης (πολυώνυμο) του πολυωνύμου p
- np.polymul(p1,p2) υπολογισμός του γινομένου των πολυωνύμων p1 και p2

Παράδειγμα

Για το πολυώνυμο του προηγούμενου παραδείγματος να βρεθούν οι παρακάτω παραστάσεις

- p(1.7)
- p'(2)

Σε Python θα έχουμε

- από το προηγούμενο παράδειγμα

p6.py - C:/Users/vasok/p6.py (3.11.2)

File Edit Format Run Options Windov

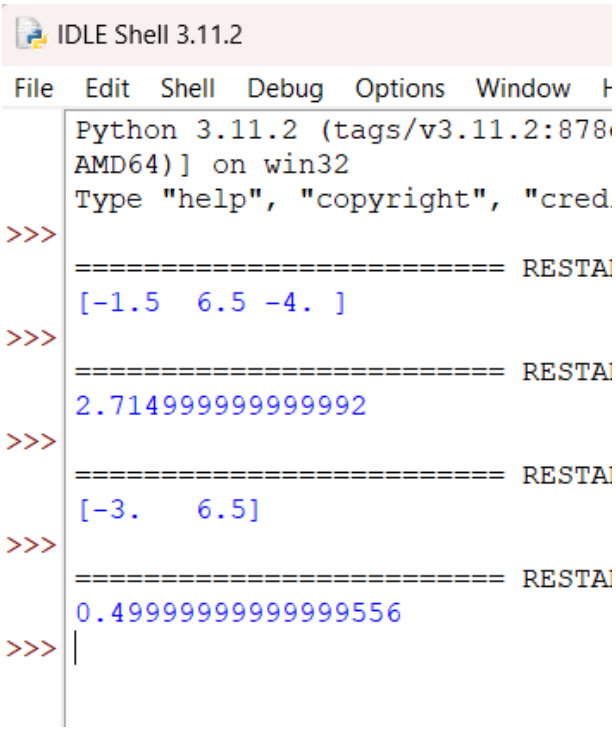
```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([1, 3, 2])


p1 = np.polyfit(x, y, 2)

print(p1)
```

```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([1, 3, 2])

p1 = np.polyfit(x, y, 2)
val=np.polyval(p1,1.7)

print(val)
```

```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([1, 3, 2])

p1 = np.polyfit(x, y, 2)
val=np.polyder(p1)

print(val)
```

```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([1, 3, 2])

p1 = np.polyfit(x, y, 2)
val=np.polyder(p1)
g=np.polyval(val,2)

print(g)
```

- Οι παραπάνω εντολές μας επιστρέφουν



Άσκηση

Έστω τα σημεία Α(0,-4), Β(1,-2), Γ(3,14) και Δ(4,40).

- Να βρεθεί το πολυώνυμο παρεμβολής το οποίο διέρχεται από τα παραπάνω σημεία.
  - Απάντηση: $p(x) = x^3 - 2x^2 + 3x - 4$
- Να υπολογιστεί η τιμή p''(2).
  - Απάντηση: p''(2)=8

Ενότητα 10

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 10)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής παραγώγισης και τις διάφορες μεθόδους υλοποίησης της μέσα

από την Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική παραγώγιση
    - Διαφορές
    - Πρώτη παράγωγος με προς τα εμπρός διαφορές
    - Πρώτη παράγωγος με προς τα πίσω διαφορές

Αριθμητική Παραγώγιση

- Υπολογισμός αριθμητικής τιμής της παραγώγου σε σημείο χ_0 .
- Είσοδος
    - Τα δοθέντα σημεία
- Έξοδος
    - Η αριθμητική τιμή της παραγώγου στο σημείο χ_0 .

Διαφορές

- Δίνεται ο πίνακας τιμών

x	0	1	2	3	4
y	-3	2	3	6	17

Έχουμε 5 σημεία άρα θα υπολογίσουμε διαφορές μέχρι τέταρτης τάξης.

Δημιουργούμε πίνακα τιμών των διαφορών

x_i	f_i	$1^{\eta\varsigma}$ τάξης	$2^{\eta\varsigma}$ τάξης	$3^{\eta\varsigma}$ τάξης	$4^{\eta\varsigma}$ τάξης
0	-3				
		5			
1	2		-4		
		1		6	
2	3		2		0
		3		6	
3	6		8		
		11			
4	17				

- Υπολογισμός των διαφορών, σε Python θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window

```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])
d1 = np.diff(y)
print(d1)
```

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Wind

```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])
d1 = np.diff(y,2)
print(d1)
```

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Windo

```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])
d1 = np.diff(y,3)
print(d1)
```

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window

```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])
d1 = np.diff(y,4)
print(d1)
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Window

```
Python 3.11.2 (tags/v3.11.2:8
AMD64)] on win32
Type "help", "copyright", "cr
>>>
========== RES
[ 5  1  3 11]
>>>
========== RES
[-4  2  8]
>>>
========== RES
[6 6]
>>>
========== RES
[0]
>>>
```

Πρώτη παράγωγος (Δ - ΝG)

Για s=0 υπολογίζουμε την πρώτη παράγωγο στο $\chi = \chi_0$. Επομένως, με βάση το n έχουμε

- Για n=1

$$f'(x_0) \simeq \frac{1}{h}\Delta f_0$$

- Για n=2

$$f'(x_0) \simeq \frac{1}{h}\left[\Delta f_0 - \frac{1}{2}\Delta^2 f_0\right]$$

- Για n=3

$$f'(x_0) \simeq \frac{1}{h}\left[\Delta f_0 - \frac{1}{2}\Delta^2 f_0 + \frac{1}{3}\Delta^3 f_0\right]$$

- Για n=k, με k περιττό

$$f'(x_0) \simeq \frac{1}{h}\left[\Delta f_0 - \frac{1}{2}\Delta^2 f_0 + \frac{1}{3}\Delta^3 f_0 - \ldots + \frac{1}{k}\Delta^k f_0\right]$$

ή ισοδύναμα, με k άρτιο

$$f'(x_0) \simeq \frac{1}{h}\left[\Delta f_0 - \frac{1}{2}\Delta^2 f_0 + \frac{1}{3}\Delta^3 f_0 - \ldots - \frac{1}{k}\Delta^k f_0\right]$$

- Η αριθμητική παραγώγιση με το πολυώνυμο Δ-ΝG χρησιμοποιείται στα αρχικά σημεία παρεμβολής.

Πρώτη Παράγωγος – Παράδειγμα

Με το πολυώνυμο Δ-ΝG έχουμε

- Για $\chi_0 = 0$ έχουμε μέχρι Δ^4 διαφορές, άρα

$$f'(0) \simeq \frac{1}{h}\left[\Delta f_0 - \frac{1}{2}\Delta^2 f_0 + \frac{1}{3}\Delta^3 f_0 - \frac{1}{4}\Delta^4 f_0\right] = 5 - \frac{1}{2}(-4) + \frac{1}{3}6 - \frac{1}{4}0$$

- Σε Python θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])

y0 = y[:5]
d1 = np.diff(y0)
d2 = np.diff(y0,2)
d3 = np.diff(y0,3)
d4 = np.diff(y0,4)

res = 1/1 * (d1[0] - 1/2 * (d2[0]) + 1/3 * (d3[0]) - 1/4 * (d4[0]))
print(res)
```

IDLE Shell 3.11.2

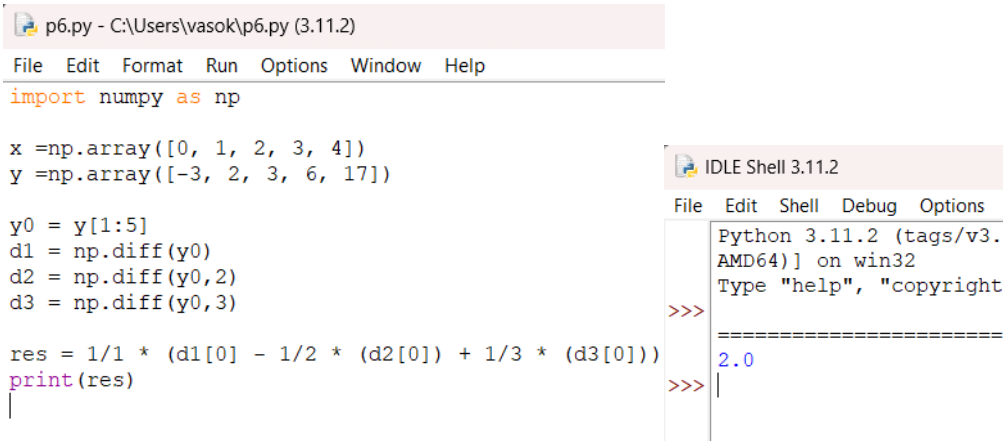
File Edit Shell Debug Options

```
Python 3.11.2 (tags/v3
AMD64)] on win32
Type "help", "copyrigh
>>>
==========
9.0
>>>
```

- Για $\chi_0 = 1$ έχουμε μέχρι Δ^3 διαφορές, άρα

$$f'(1) \simeq \frac{1}{h}\left[\Delta f_0 - \frac{1}{2}\Delta^2 f_0 + \frac{1}{3}\Delta^3 f_0\right] = 1 - \frac{1}{2}2 + \frac{1}{3}6 = 2$$

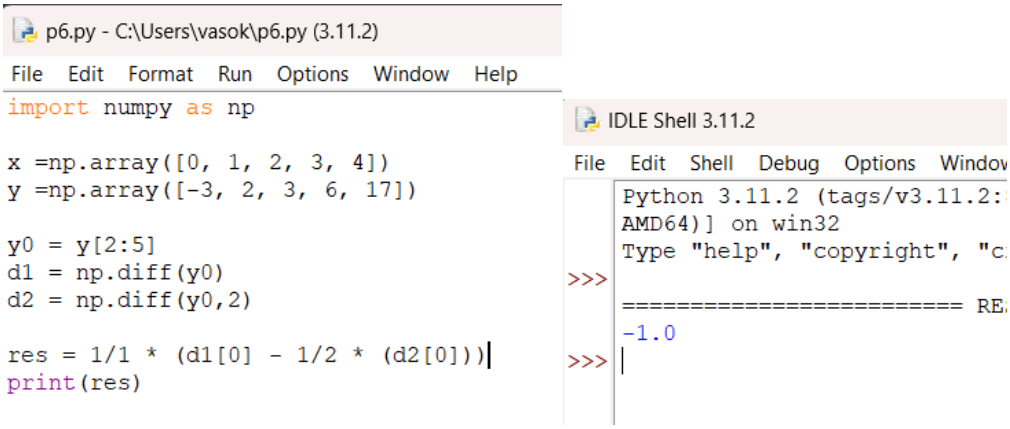
- Σε Python θα έχουμε



- Για $\chi_0 = 2$ έχουμε μέχρι Δ^2 διαφορές, άρα

$$f'(2) \simeq \frac{1}{h}\left[\Delta f_0 - \frac{1}{2}\Delta^2 f_0\right] = 3 - \frac{1}{2}8 = -1$$

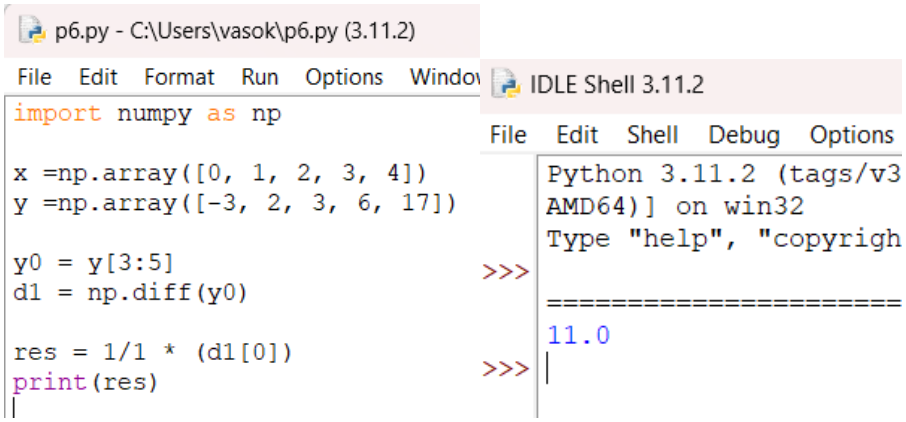
- Σε Python θα έχουμε



- Για $\chi_0 = 3$ έχουμε μέχρι Δ^1 διαφορές, άρα

$$f'(3) \simeq \frac{1}{h}\Delta f_0 = 11$$

- Σε Python θα έχουμε



- Για $\chi_0 = 4$ δεν έχουμε προς τα εμπρός διαφορές και δεν μπορούμε να υπολογίσουμε την παράγωγο.

Πρώτη Παράγωγος $(\nabla - NG)$

Για s=0 υπολογίζουμε την πρώτη παράγωγο στο $\chi = \chi_n$.

Επομένως, με βάση το n έχουμε

- Για n=1

$$f'(x_n) \simeq \frac{1}{h}\nabla f_n$$

- Για n=2

$$f'(x_n) \simeq \frac{1}{h}\left[\nabla f_n + \frac{1}{2}\nabla^2 f_n\right]$$

- Για n=3

$$f'(x_n) \simeq \frac{1}{h}\left[\nabla f_n + \frac{1}{2}\nabla^2 f_n + \frac{1}{3}\nabla^3 f_n\right]$$

- Για n=k

$$f'(x_n) \simeq \frac{1}{h}\left[\nabla f_n + \frac{1}{2}\nabla^2 f_n + \frac{1}{3}\nabla^3 f_n + \ldots + \frac{1}{k}\nabla^k f_n\right]$$

- Η αριθμητική παραγώγιση με το πολυώνυμο $\nabla - NG$ χρησιμοποιείται στα τελευταία σημεία παρεμβολής.

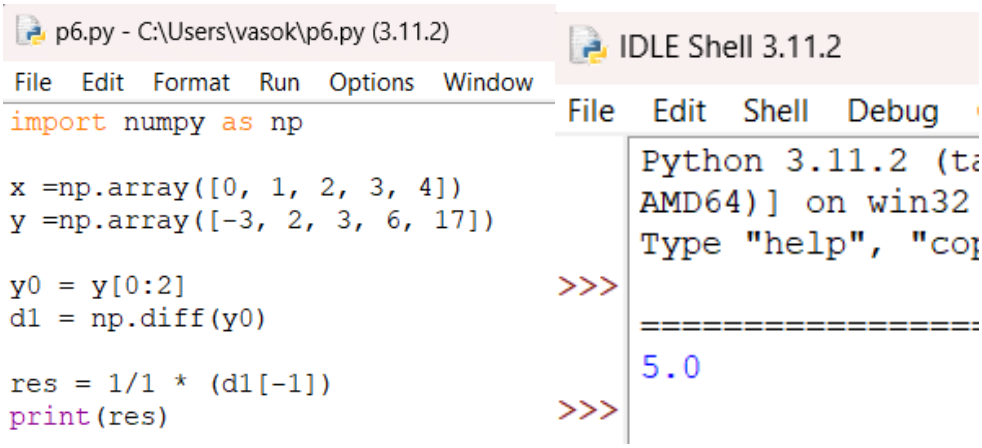
Πρώτη Παράγωγος - Παράδειγμα

Με το πολυώνυμο $\nabla - NG$ έχουμε

- Για $\chi_n = 0$ δεν έχουμε προς τα πίσω διαφορές και δεν μπορούμε να υπολογίσουμε την παράγωγο.
- Για $\chi_n = 1$ έχουμε μέχρι ∇^1 διαφορές, άρα

$$f'(1) \simeq \frac{1}{h}\nabla f_n = 5$$

- Σε Python θα έχουμε



- Για $\chi_n = 2$ έχουμε μέχρι ∇^2 διαφορές, άρα

$$f'(2) \simeq \frac{1}{h}\left[\nabla f_n + \frac{1}{2}\nabla^2 f_n\right] = 1 + \frac{1}{2}(-4) = -1$$

- Σε Python θα έχουμε

```
p6.py - C:\Users\vasok\p6.py (3.11.2)
File  Edit  Format  Run  Options  Window
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])

y0 = y[0:3]
d1 = np.diff(y0)
d2 = np.diff(y0,2)

res = 1/1 * (d1[-1]+1/2*(d2[-1]))
print(res)
```

```
IDLE Shell 3.11.2
File  Edit  Shell  Debug  Options  Wi
Python 3.11.2 (tags/v3.11
AMD64)] on win32
Type "help", "copyright",
>>>
==========================
-1.0
>>>
```

- Για $\chi_n = 3$ έχουμε μέχρι ∇^3 διαφορές, άρα

$$f'(3) \simeq \frac{1}{h}\left[\nabla f_n + \frac{1}{2}\nabla^2 f_n + \frac{1}{3}\nabla^3 f_n\right] = 3 + \frac{1}{2}2 + \frac{1}{3}6 = 6$$

- Σε Python θα έχουμε

```
p6.py - C:\Users\vasok\p6.py (3.11.2)
File  Edit  Format  Run  Options  Window  Help
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])

y0 = y[0:4]
d1 = np.diff(y0)
d2 = np.diff(y0,2)
d3 = np.diff(y0,3)

res = 1/1 * (d1[-1]+1/2*(d2[-1])+1/3*(d3[-1]))
print(res)
```

```
IDLE Shell 3.11.2
File  Edit  Shell  Debug  Optio
Python 3.11.2 (tags/
AMD64)] on win32
Type "help", "copyri
>>>
==========
6.0
>>>
```

- Για $\chi_n = 4$ έχουμε μέχρι ∇^4 διαφορές, άρα

$$f'(4) \simeq \frac{1}{h}\left[\nabla f_n + \frac{1}{2}\nabla^2 f_n + \frac{1}{3}\nabla^3 f_n + \frac{1}{4}\nabla^4 f_n\right] = 11 + \frac{1}{2}8 + \frac{1}{3}6 + \frac{1}{4}0 =$$

- Σε Python θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])

y0 = y[0:5]
d1 = np.diff(y0)
d2 = np.diff(y0,2)
d3 = np.diff(y0,3)
d4 = np.diff(y0,4)

res = 1/1 * ((d1[-1]+1/2*(d2[-1])+1/3*(d3[-1])))+1/4*(d4[-1])
print(res)
```

IDLE Shell 3.11.2

File Edit Shell Debug Options

```
Python 3.11.2 (tags/v3
AMD64)] on win32
Type "help", "copyrigh
>>>
==========
17.0
>>>
```

Άσκηση

Δίνεται ο πίνακας τιμών

x	0	2	4	6	8	10
y	−3	1	3	5	7	9

- Να υπολογιστούν οι τιμές f'(4), f'(6), f'(2), f'(8).

Ενότητα 11

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 11)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής ολοκλήρωσης και τις διάφορες μεθόδους υλοποίησης της αριθμητικής ολοκλήρωσης μέσα από την Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική ολοκλήρωση
2. Κανόνας τραπεζίου
    - Κανόνας τραπεζίου – υλοποίηση
    - Κανόνας τραπεζίου - παραδείγματα
3. Κανόνας Simpson
    - Κανόνας Simpson - υλοποίηση
4. Συνδυαστική Άσκηση

Αριθμητική Ολοκλήρωση

- Με την αριθμητική ολοκλήρωση υπολογίζεται η τιμή ενός ορισμένου ολοκληρώματος.
- Η αριθμητική ολοκλήρωση μπορεί να εφαρμοστεί είτε σε συνάρτηση είτε σε ένα σύνολο σημείων που προέρχονται από δειγματοληψία.

Στην πρώτη προσέγγιση έχουμε

- Είσοδος
    - Τη συνάρτηση
    - Το διάστημα
    - Το πλήθος των υποδιαστημάτων που θα εφαρμοστεί η αριθμητική ολοκλήρωση
- Έξοδος
    - Η τιμή του ορισμένου ολοκληρώματος

Στη δεύτερη προσέγγιση έχουμε

- Είσοδος
    - Τις τετμημένες και τις τεταγμένες των σημείων
- Έξοδος
    - Η τιμή του ορισμένου ολοκληρώματος

Κανόνας Τραπεζίου

Έστω μια συνάρτηση f. Η τιμή του ορισμένου ολοκληρώματος

$$I = \int_a^b f(x)dx$$

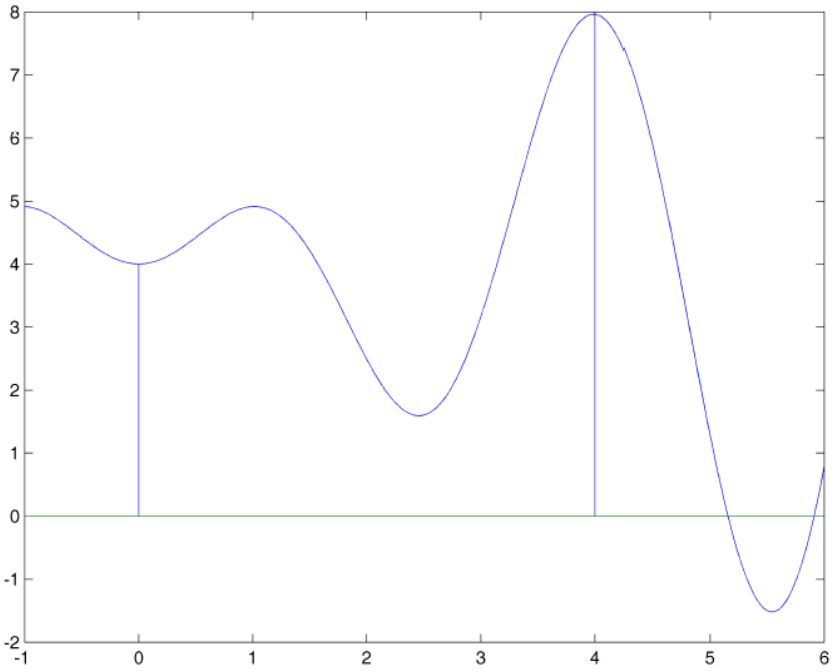
προσεγγίζεται από τον κανόνα του τραπεζίου σύμφωνα με τον τύπο

$$I = \frac{h}{2}(f_0 + 2 \cdot f_1 + \ldots + 2 \cdot f_{n-1} + f_n) = \frac{h}{2}\left(f_0 + 2 \cdot \sum_{i=1}^{n-1} f_i + f_n\right)$$

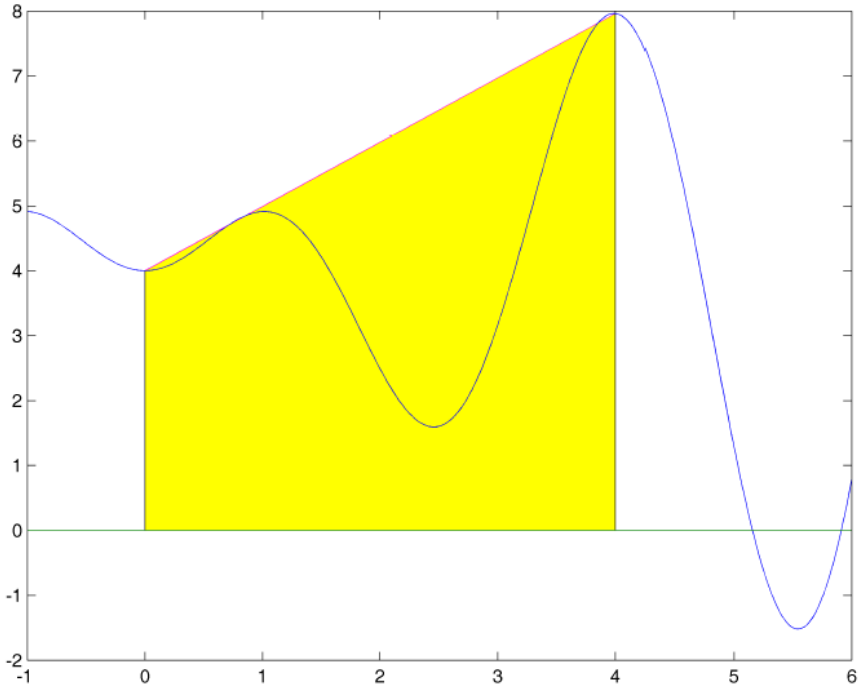
όπου n το πλήθος των υποδιαστημάτων που θα εφαρμοστεί ο τύπος και

$$h = \frac{b-a}{n}$$

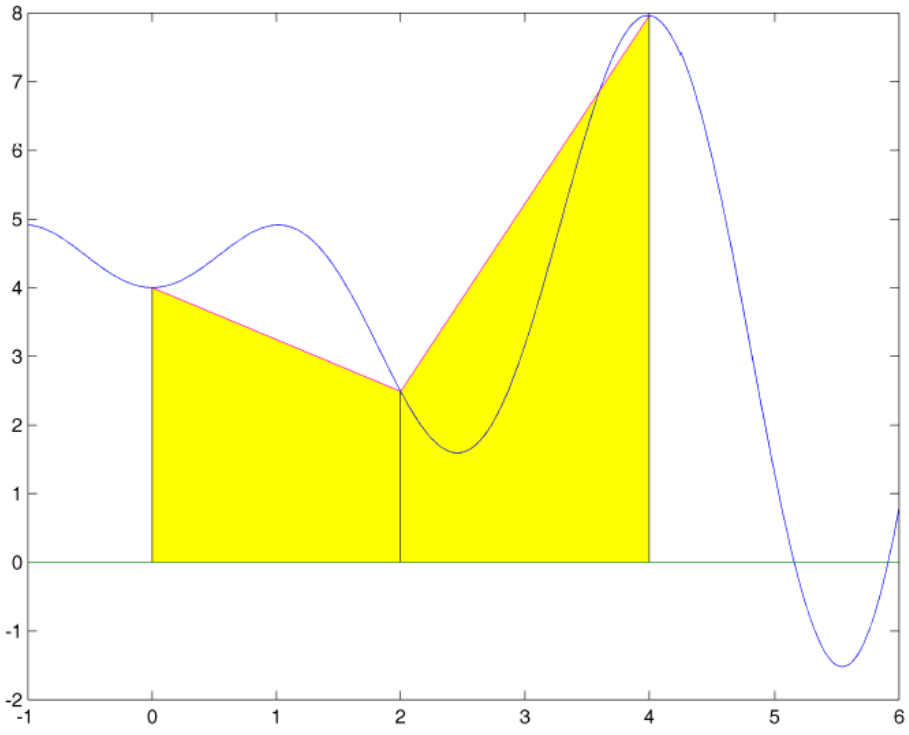
το βήμα των ισαπεχόντων σημείων.



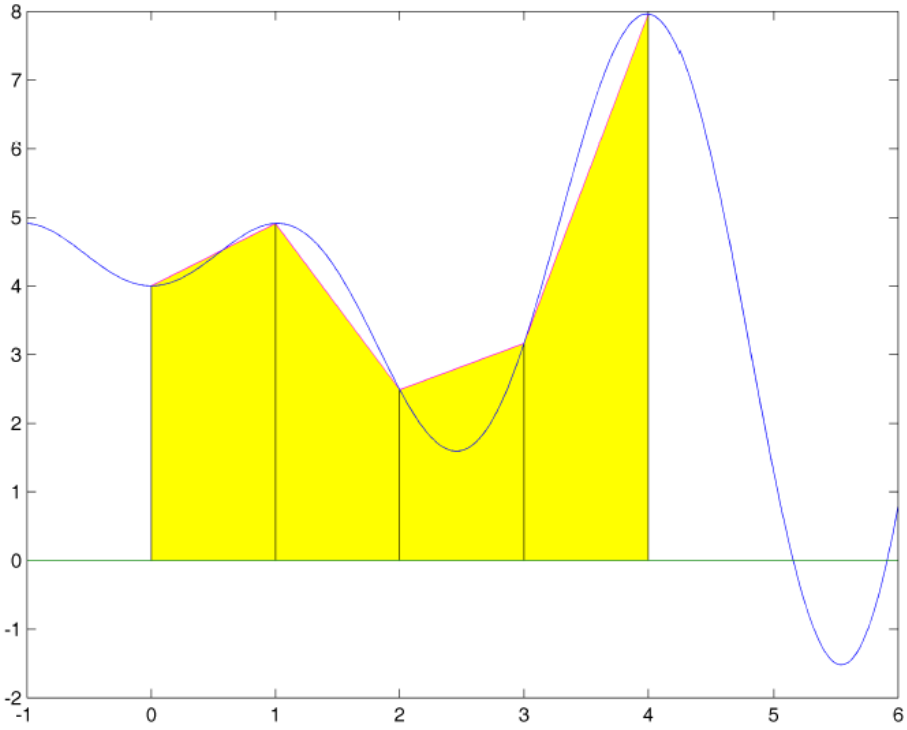
Σχήμα: Υπολογισμός του ολοκληρώματος $\int_0^4 f(x)dx$



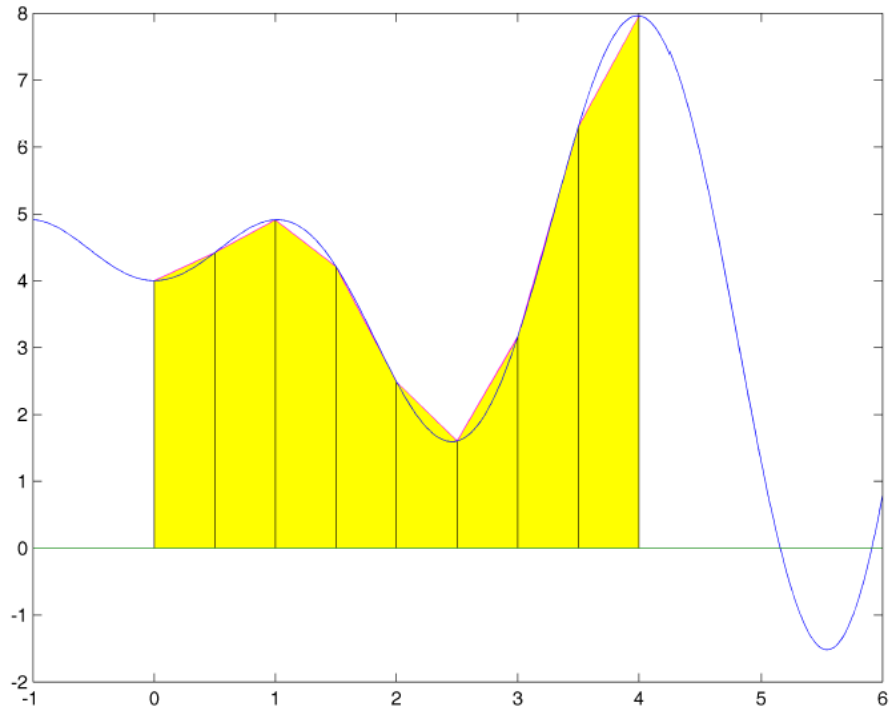
Σχήμα: Κανόνας τραπεζίου για n=1



Σχήμα: Κανόνας τραπεζίου για n=2



Σχήμα: Κανόνας τραπεζίου για n=4



Σχήμα: Κανόνας τραπεζίου για n=8

Κανόνας Τραπεζίου – Υλοποίηση

- Υλοποίηση πρώτης προσέγγισης σε συνάρτηση Python

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

```
import numpy as np
def trapeziou(f, a, b, n):
    h = (b - a) / n
    S = f(a)
    for i in range(1, n):
        x = a + h*i
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/2
    return I
```

Κανόνας Τραπεζίου – Παράδειγμα 1

Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του τραπεζίου για n=1, n=2, n=4, n=100, n=1000.

Σε Python θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

```
import numpy as np
def trapeziou(f, a, b, n):
    h = (b - a) / n
    S = f(a)
    for i in range(1, n):
        x = a + h*i
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/2
    return I

def f(x):
    return x*np.sin(2*x)+4
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Window Help

```
Python 3.11.2 (tags/v3.11.2:878ead
AMD64)] on win32
Type "help", "copyright", "credits

========== RESTART:
>>> I=trapeziou(f, 0, 4, 1)
>>> print(I)
23.914865972987055
>>> I=trapeziou(f, 0, 4, 2)
>>> print(I)
16.930223005261816
>>> I=trapeziou(f, 0, 4, 4)
>>> print(I)
16.53616243485981
>>> I=trapeziou(f, 0, 4, 100)
>>> print(I)
16.53831636933605
>>> I=trapeziou(f, 0, 4, 1000)
>>> print(I)
16.5383393964196
>>>
```

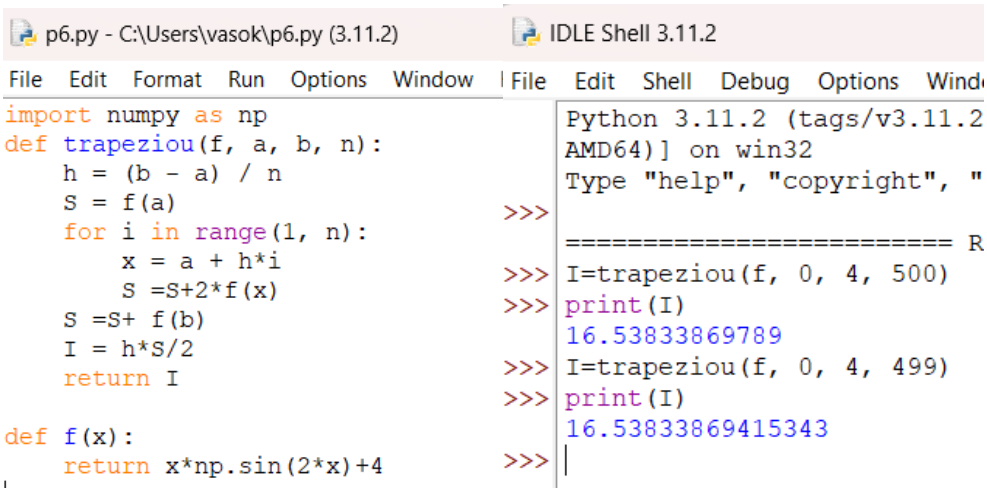
Κανόνας Τραπεζίου – Παράδειγμα 2

Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του τραπεζίου για n=500 και να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.

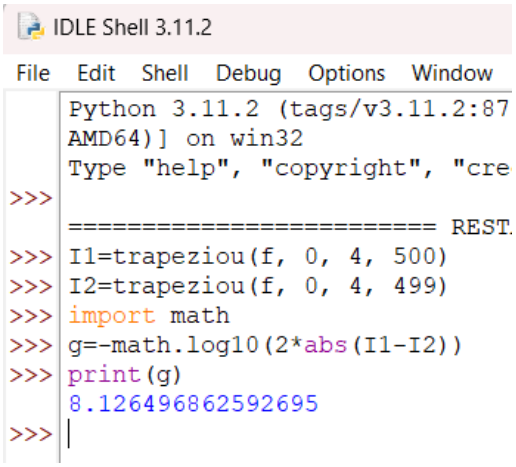
- Σε Python θα έχουμε



- Συγκρίνουμε τις τιμές ως προς τα δεκαδικά ψηφία τους σύμφωνα με τον τύπο

$$|x_n - x_{n-1}| < \frac{1}{2}10^{-k} \Longrightarrow k < -\log(2 \cdot |x_n - x_{n-1}|)$$

όπου k το πλήθος των δεκαδικών ψηφίων.



επειδή

$$k < -log(2 \cdot |I_{500} - I_{499}|) = 8.126496862592695$$

θα έχουμε

k=8

Επομένως, η προσεγγιστική λύση I_{500} έχει ακρίβεια 8 δεκαδικών ψηφίων.

Κανόνας Τραπεζίου – Άσκηση

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^3 e^{x^2} dx$$

με τον κανόνα του τραπεζίου για n=100.

  - Απάντηση: (1448.1892158659284)
- Να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.
  - Απάντηση: (0 δεκαδικά ψηφία)

Κανόνας Simpson

Έστω μια συνάρτηση f. Η τιμή του ορισμένου ολοκληρώματος

$$I = \int_a^b f(x)dx$$

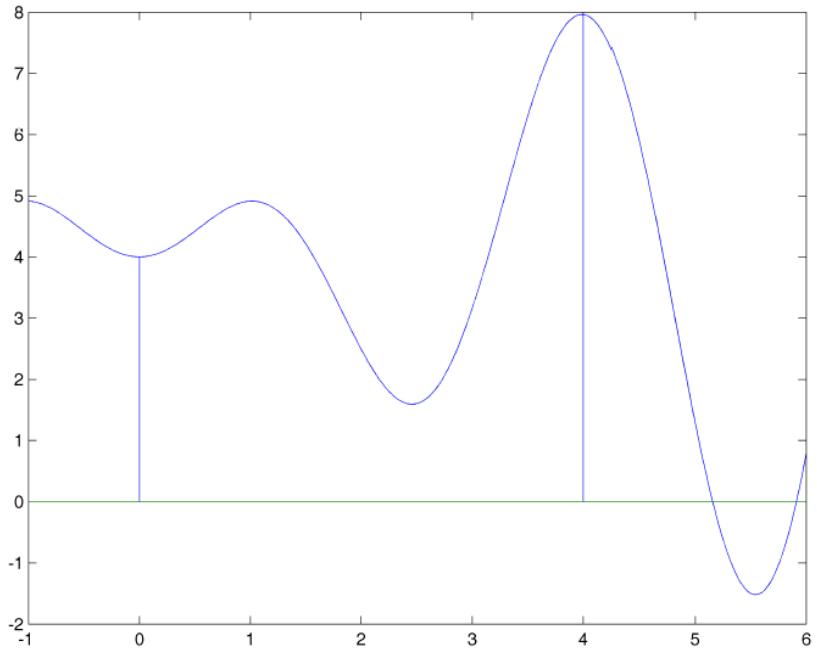
προσεγγίζεται από τον κανόνα του Simpson σύμφωνα με τον τύπο

$$\begin{aligned} I &= \frac{h}{3}(f_0 + 4 \cdot f_1 + 2 \cdot f_2 + \ldots + 2 \cdot f_{2n-2} + 4 \cdot f_{2n-1} + f_n) \\ &= \frac{h}{3}\left(f_0 + 4 \cdot \sum_{i=1}^{n} f_{2i-1} + 2 \cdot \sum_{i=1}^{n-1} f_{2i} + f_n\right) \end{aligned}$$

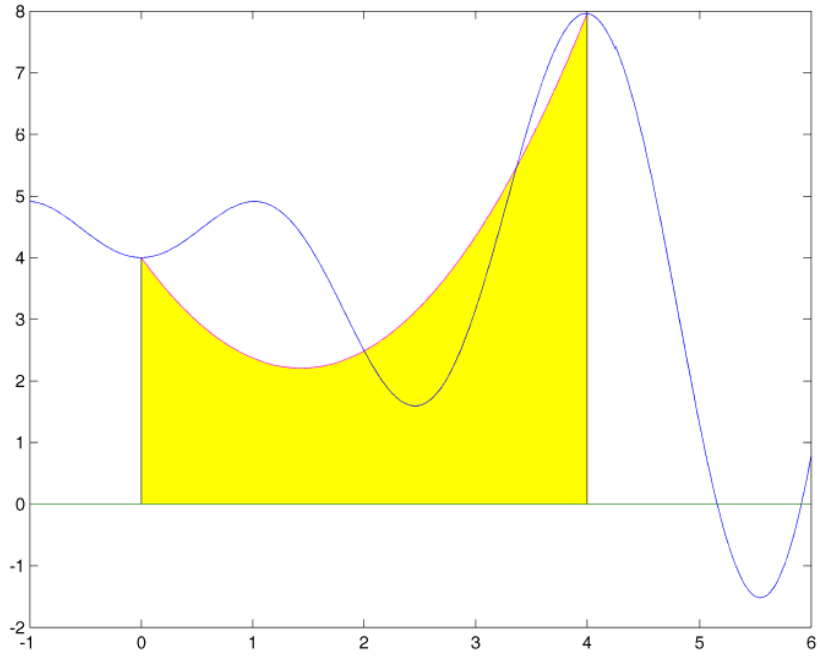
όπου n το πλήθος των υποδιαστημάτων που θα εφαρμοστεί ο τύπος και

$$h = \frac{b-a}{2n}$$

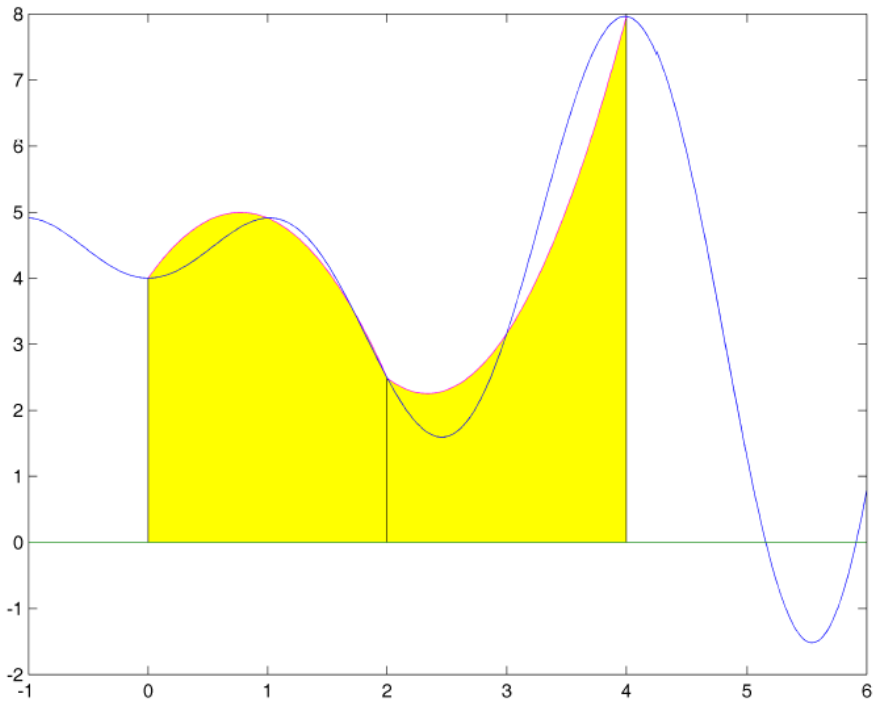
το βήμα των ισαπέχοντων σημείων.



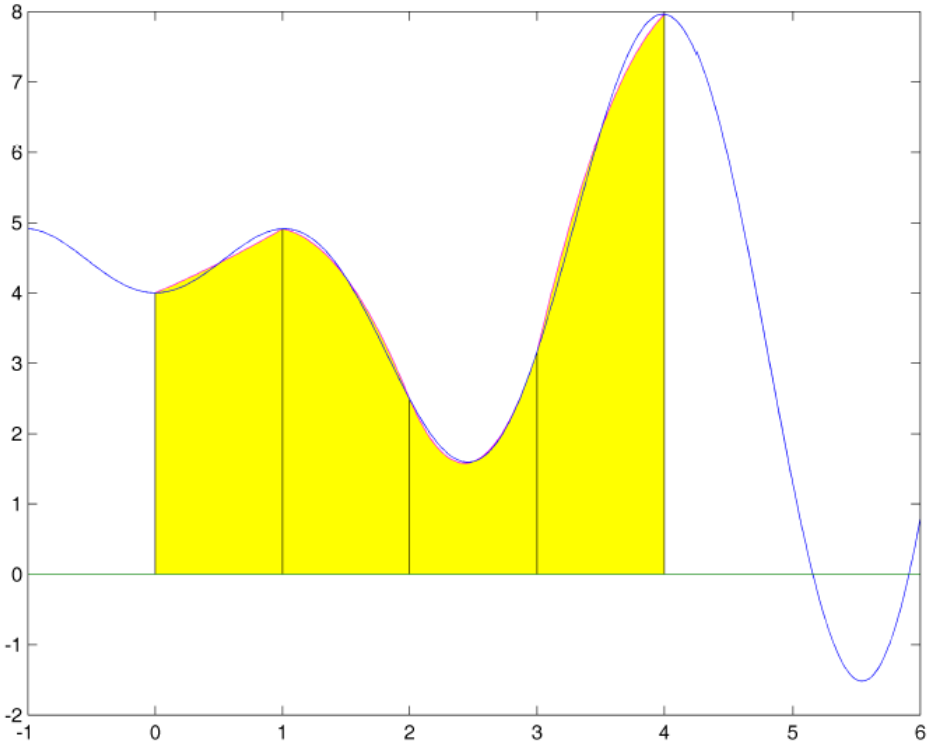
Σχήμα: Υπολογισμός του ολοκληρώματος $\int_0^4 f(x)dx$



Σχήμα: Κανόνας Simpson για n=1



Σχήμα: Κανόνας Simpson για n=2



Σχήμα: Κανόνας Simpson για n=4

Κανόνας Simpson – Υλοποίηση

- Υλοποίηση πρώτης προσέγγισης σε συνάρτηση Python

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Wind

```
import numpy as np
import math
def simpson(f, a, b, n):
    h = (b - a) / (2 * n)
    S = f(a)
    for i in range(1, n+1):
        x = a + h*(2*i-1)
        S =S+4*f(x)
    for i in range(1, n):
        x = a + h*(2*i)
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/3
    return I
```

Κανόνας Simpson – Παράδειγμα 1

Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του Simpson για n=1, n=2, n=4, n=100, n=1000.

Σε Python θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Wind

```
import numpy as np
import math
def simpson(f, a, b, n):
    h = (b - a) / (2 * n)
    S = f(a)
    for i in range(1, n+1):
        x = a + h*(2*i-1)
        S =S+4*f(x)
    for i in range(1, n):
        x = a + h*(2*i)
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/3
    return I
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Window He

```
Python 3.11.2 (tags/v3.11.2:878e
AMD64)] on win32
Type "help", "copyright", "credi
>>>
========================= RESTAR
>>> g=simpson(f, 0, 4, 1)
>>> print(g)
14.602008682686735
>>> g=simpson(f, 0, 4, 2)
>>> print(g)
16.404808911392475
>>> g=simpson(f, 0, 4, 4)
>>> print(g)
16.535092753854382
>>> g=simpson(f, 0, 4, 100)
>>> print(g)
16.538339622856025
>>> g=simpson(f, 0, 4, 1000)
>>> print(g)
16.53833962927242
>>>
```

Κανόνας Simpson – Παράδειγμα 2

Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του Simpson για n=500 και να βρεθεί ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.

Σε Python θα έχουμε

- Συγκρίνουμε τις τιμές ως προς τα δεκαδικά ψηφία τους σύμφωνα με τον τύπο

$$|x_n - x_{n-1}| < \frac{1}{2}10^{-k} \Longrightarrow k < -\log(2 \cdot |x_n - x_{n-1}|)$$

όπου k το πλήθος των δεκαδικών ψηφίων.

```
>>> import math
>>> g=-math.log10(2*abs(I1-I2))
>>> print(g)
12.671288541487455
>>>
```

επειδή

$$k < -log(2 \cdot |I_{500} - I_{499}|) = 12.671288541487455$$

Θα έχουμε

k=12

Επομένως, η προσεγγιστική λύση I_{500} έχει ακρίβεια 12 δεκαδικών ψηφίων.

Κανόνας Simpson – Άσκηση

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^3 e^{x^2} dx$$

με τον κανόνα του Simpson για n=100.

  - Απάντηση: (1444.5456964394289)

- Να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.
  - Απάντηση: (4 δεκαδικά ψηφία)

Συνδυαστική Άσκηση

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^3 e^{x^2} dx$$

με τον κανόνα του τραπεζίου και του Simpson για n=500.

  - Απάντηση:
    - Μέθοδος τραπεζίου: 1444.6909747280079
    - Μέθοδος Simpson: 1444.5451238115556

- Να βρεθεί η ακρίβεια των αποτελεσμάτων σε δεκαδικά ψηφία.
  - Απάντηση:
    - Μέθοδος τραπεζίου: 2.93170469768348 (2 δεκαδικά ψηφία)
    - Μέθοδος Simpson: 7.830546951121842 (7 δεκαδικά ψηφία)
- Ποια είναι η πιο ακριβής μέθοδος;
  - Απάντηση: Η μέθοδος Simpson

Ενότητα 12

Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός (Εργαστήριο 12)

Στόχος του εργαστηρίου είναι η γνωριμία του φοιτητή με την έννοια της αριθμητικής ολοκλήρωσης και τις διάφορες μεθόδους υλοποίησης της μέσα από την Python. Ειδικότερα, ο φοιτητής θα ασχοληθεί με τα παρακάτω αντικείμενα:

1. Αριθμητική ολοκλήρωση
2. Κανόνας ορθογωνίου
    - Κανόνας ορθογωνίου - παραδείγματα
3. Κανόνας τραπεζίου
    - Κανόνας τραπεζίου - παραδείγματα
4. Κανόνας Simpson
    - Κανόνας Simpson - παραδείγματα
5. Συνδυαστική άσκηση

Αριθμητική Ολοκλήρωση

- Με την αριθμητική ολοκλήρωση υπολογίζεται η τιμή ενός ορισμένου ολοκληρώματος.
- Η αριθμητική ολοκλήρωση μπορεί να εφαρμοστεί είτε σε συνάρτηση είτε σε ένα σύνολο σημείων που προέρχονται από δειγματοληψία.
- Οι κανόνες αριθμητικής ολοκλήρωσης που θα υλοποιηθούν είναι¨
    - Κανόνας Ορθογωνίου
    - Κανόνας Τραπεζίου
    - Κανόνας Simpson

Κανόνας Ορθογωνίου

Έστω μια συνάρτηση f. Η τιμή του ορισμένου ολοκληρώματος

$$I = \int_a^b f(x)dx$$

προσεγγίζεται από τον κανόνα του ορθογωνίου σύμφωνα με τον τύπο

$$I = h(f_0 + f_1 + \ldots + f_{n-1}) = h\sum_{i=0}^{n-1} f_i$$

όπου n το πλήθος των υποδιαστημάτων που θα εφαρμοστεί ο τύπος και

$$h = \frac{b-a}{n}$$

το βήμα των ισαπεχόντων σημείων.

Τα σημεία $\chi_0, \chi_1, \chi_2, \ldots, \chi_{n-1}$ δίνονται από τον τύπο

$$x_i = x_0 + i \cdot h \quad με \quad i = 0,1,2,\ldots,n-1 \quad και\, x_0 = a + \frac{h}{2}$$

Κανόνας Ορθογωνίου – Παράδειγμα 1

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του Ορθογωνίου για n=4, n=200.

- Σε Python για n=4 θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

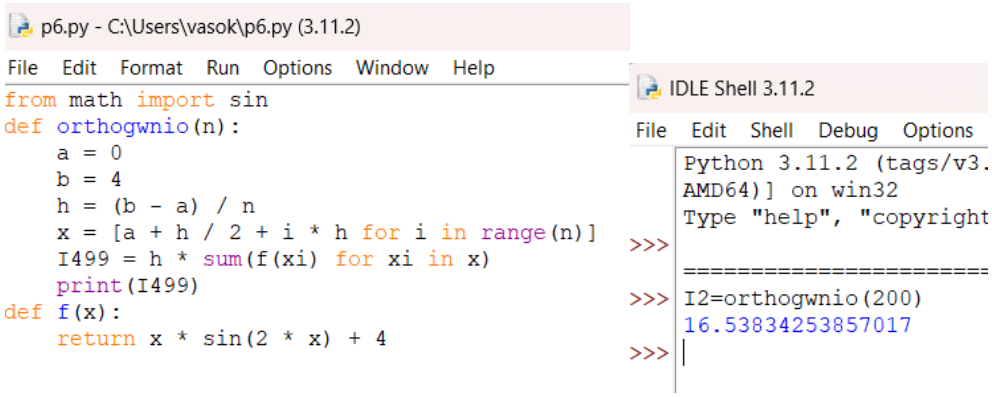
```
from math import sin
def orthogwnio(n):
    a = 0
    b = 4
    h = (b - a) / n
    x = [a + h / 2 + i * h for i in range(n)]
    I499 = h * sum(f(xi) for xi in x)
    print(I499)
def f(x):
    return x * sin(2 * x) + 4
```

IDLE Shell 3.11.2

File Edit Shell Debug Options

```
Python 3.11.2 (tags/v3.1
AMD64)] on win32
Type "help", "copyright"
>>>
==========
>>> I1=orthogwnio(4)
16.534557913351666
>>>
```

- Σε Python για n=200 θα έχουμε



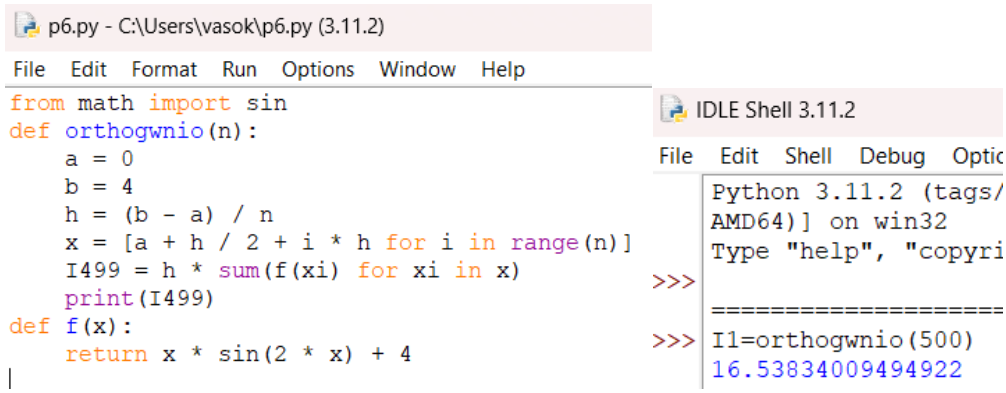
Κανόνας Ορθογωνίου – Παράδειγμα 2

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του Ορθογωνίου για n=500 και να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.

- Σε Python για n=500



- Σε Python για n=499

```
p6.py - C:\Users\vasok\p6.py (3.11.2)
File  Edit  Format  Run  Options  Window  Help
from math import sin
def orthogwnio(n):
    a = 0
    b = 4
    h = (b - a) / n
    x = [a + h / 2 + i * h for i in range(n)]
    I499 = h * sum(f(xi) for xi in x)
    print(I499)
def f(x):
    return x * sin(2 * x) + 4
```

```
>>> I2=orthogwnio(499)
16.538340096817368
>>>
```

- Συγκρίνουμε τις τιμές ως προς τα δεκαδικά ψηφία τους σύμφωνα με τον τύπο

$$|x_n - x_{n-1}| < \frac{1}{2}10^{-k} \Longrightarrow k < -\log\left(2 \cdot |x_n - x_{n-1}|\right)$$

όπου k το πλήθος των δεκαδικών ψηφίων.

- Στην Python θα έχουμε

```
p6.py - C:\Users\vasok\p6.py (3.11.2)
File  Edit  Format  Run  Options  Window  Help
from math import sin
import math

def f(x):
    return x * sin(2 * x) + 4

a = 0
b = 4
n = 500
n1=499
h = (b - a) / n
x = [a + h / 2 + i * h for i in range(n)]
I500 = h * sum(f(xi) for xi in x)
x = [a + h / 2 + i * h for i in range(n1)]
I499 = h * sum(f(xi) for xi in x)
I1=print(I500)
I2=print(I499)
g=-math.log10(2*abs(I500-I499))
print(g)
```

```
======================
16.53834009494922
16.474676092241776
0.8950760640644286
>>>
```

επειδή

$$k < -log(2 \cdot |I_{500} - I_{499}|) = 0.8950760640644286$$

Θα έχουμε

k=0

Επομένως, η προσεγγιστική λύση I_{500} έχει ακρίβεια 0 δεκαδικών ψηφίων.

Κανόνας Ορθογωνίου – Άσκηση

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^3 e^{x^2} dx$$

με τον κανόνα του Ορθογωνίου για n=100.

  - Απάντηση: (1442.7239367261789)

- Να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.
  - Απάντηση: (2 δεκαδικά ψηφία)

Κανόνας Τραπεζίου

Έστω μια συνάρτηση f. Η τιμή του ορισμένου ολοκληρώματος

$$I = \int_a^b f(x) dx$$

προσεγγίζεται από τον κανόνα του τραπεζίου σύμφωνα με τον τύπο

$$I = \frac{h}{2}(f_0 + 2 \cdot f_1 + \ldots + 2 \cdot f_{n-1} + f_n) = \frac{h}{2}\left(f_0 + 2 \cdot \sum_{i=1}^{n-1} f_i + f_n\right)$$

όπου n το πλήθος των υποδιαστημάτων που θα εφαρμοστεί ο τύπος και

$$h = \frac{b-a}{n}$$

το βήμα των ισαπεχόντων σημείων.

Τα σημεία $\chi_0, \chi_1, \chi_2, \ldots, \chi_n$ δίνονται από τον τύπο

$$x_i = x_0 + i \cdot h \quad με \quad i = 0,1,2,\ldots,n \quad και \; x_0 = a, \; x_n = b$$

Κανόνας Τραπεζίου – Παράδειγμα 1

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του τραπεζίου για n=4, n=100.

- Σε Python για n=4 θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

```
import numpy as np
def trapezioy(f,a,b,n):
    h=(b-a)/n
    S=f(a)
    for i in range(1,n):
        x=a+h*i
        S=S+2*f(x)
    S=S+f(b)
    I=h*S/2
    return I

def f(x):
    return x*np.sin(2*x)+4
```

IDLE Shell 3.11.2

File Edit Shell Debug Options W

```
Python 3.11.2 (tags/v3.11
AMD64)] on win32
Type "help", "copyright",
>>>
==========================
>>> a=trapezioy(f,0,4,4)
>>> print(a)
16.53616243485981
>>>
```

- Σε Python για n=100 θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

```
import numpy as np
def trapezioy(f,a,b,n):
    h=(b-a)/n
    S=f(a)
    for i in range(1,n):
        x=a+h*i
        S=S+2*f(x)
    S=S+f(b)
    I=h*S/2
    return I

def f(x):
    return x*np.sin(2*x)+4
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Wi

```
Python 3.11.2 (tags/v3.11
AMD64)] on win32
Type "help", "copyright",
>>>
==========================
>>> a=trapezioy(f,0,4,100)
>>> print(a)
16.53831636933605
>>>
```

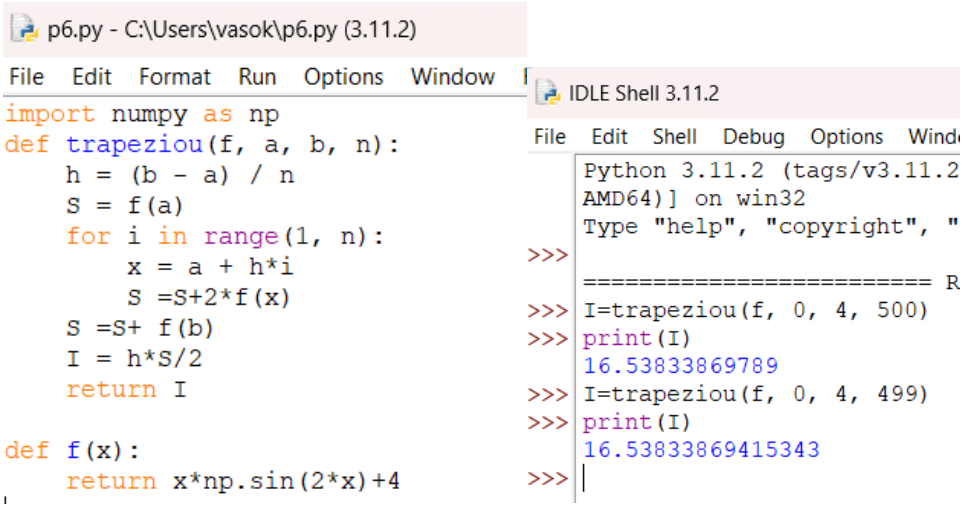
Κανόνας Τραπεζίου – Παράδειγμα 2

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

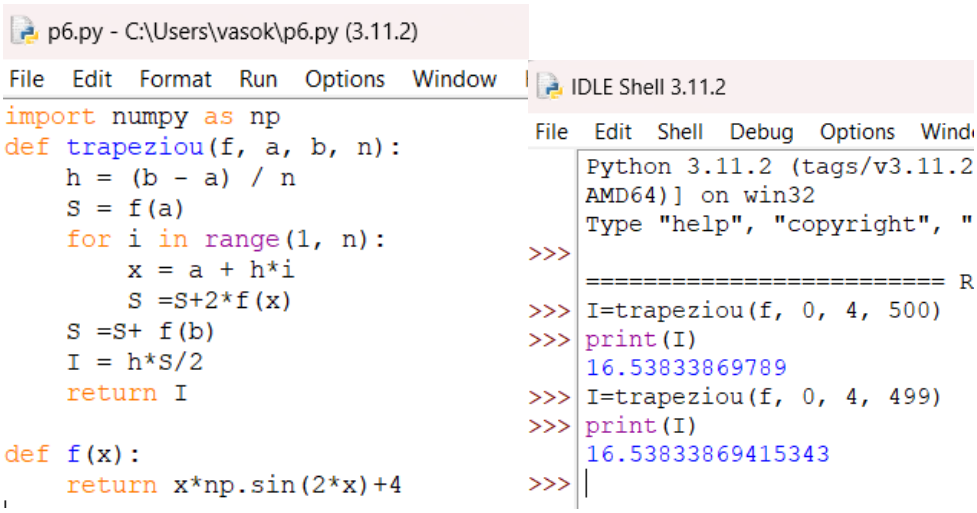
$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του τραπεζίου για n=500 και να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.

- Σε Python για n=500 θα έχουμε



- Σε Python για n=499 θα έχουμε

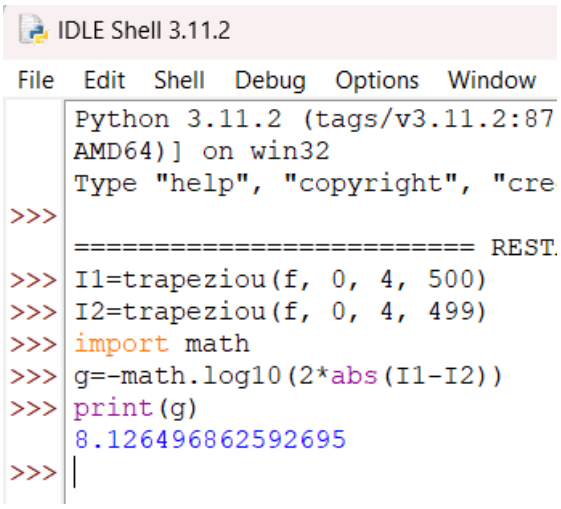


- Συγκρίνουμε τις τιμές ως προς τα δεκαδικά ψηφία τους σύμφωνα με τον τύπο

$$|x_n - x_{n-1}| < \frac{1}{2}10^{-k} \Longrightarrow k < -\log\left(2 \cdot |x_n - x_{n-1}|\right)$$

όπου k το πλήθος των δεκαδικών ψηφίων.

- Σε Python θα έχουμε



επειδή

$$k < -log(2 \cdot |I_{500} - I_{499}|) = 8.126496862592695$$

θα έχουμε

k=8

Επομένως, η προσεγγιστική λύση I_{500} έχει ακρίβεια 8 δεκαδικών ψηφίων.

Κανόνας Τραπεζίου – Άσκηση

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^3 e^{x^2} dx$$

με τον κανόνα του τραπεζίου για n=100.

  - Απάντηση: (1448.1892158659284)
- Να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.
  - Απάντηση: (0 δεκαδικά ψηφία)

Κανόνας Simpson

Έστω μια συνάρτηση f. Η τιμή του ορισμένου ολοκληρώματος

$$I = \int_a^b f(x)dx$$

προσεγγίζεται από τον κανόνα του Simpson σύμφωνα με τον τύπο

$$\begin{aligned} I &= \frac{h}{3}(f_0 + 4 \cdot f_1 + 2 \cdot f_2 + \ldots + 2 \cdot f_{2n-2} + 4 \cdot f_{2n-1} + f_{2n}) \\ &= \frac{h}{3}\left(f_0 + 4 \cdot \sum_{i=1}^{n} f_{2i-1} + 2 \cdot \sum_{i=1}^{n-1} f_{2i} + f_{2n}\right) \end{aligned}$$

όπου n το πλήθος των υποδιαστημάτων που θα εφαρμοστεί ο τύπος και

$$h = \frac{b-a}{2n}$$

το βήμα των ισαπέχοντων σημείων.

Τα σημεία $\chi_0, \chi_1, \chi_2, \ldots, \chi_{2n}$ δίνονται από τον τύπο

$$x_i = x_0 + i \cdot h \quad με \quad i = 0,1,2,\ldots,2n \quad και \; x_0 = a, \; x_{2n} = b$$

Κανόνας Simpson – Παράδειγμα 1

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του Simpson για n=4, n=1000.

- Σε Python για n=4 θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Wind

```
import numpy as np
import math
def simpson(f, a, b, n):
    h = (b - a) / (2 * n)
    S = f(a)
    for i in range(1, n+1):
        x = a + h*(2*i-1)
        S =S+4*f(x)
    for i in range(1, n):
        x = a + h*(2*i)
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/3
    return I
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Window He

```
Python 3.11.2 (tags/v3.11.2:878e
AMD64)] on win32
Type "help", "copyright", "credi
>>>
========================= RESTAR
>>> g=simpson(f, 0, 4, 1)
>>> print(g)
14.602008682686735
>>> g=simpson(f, 0, 4, 2)
>>> print(g)
16.404808911392475
>>> g=simpson(f, 0, 4, 4)
>>> print(g)
16.535092753854382
>>> g=simpson(f, 0, 4, 100)
>>> print(g)
16.538339622856025
>>> g=simpson(f, 0, 4, 1000)
>>> print(g)
16.53833962927242
>>>
```

- Σε Python για n=1000 θα έχουμε

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Wind

```
import numpy as np
import math
def simpson(f, a, b, n):
    h = (b - a) / (2 * n)
    S = f(a)
    for i in range(1, n+1):
        x = a + h*(2*i-1)
        S =S+4*f(x)
    for i in range(1, n):
        x = a + h*(2*i)
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/3
    return I
```

IDLE Shell 3.11.2

File Edit Shell Debug Options Window He

```
Python 3.11.2 (tags/v3.11.2:878e
AMD64)] on win32
Type "help", "copyright", "credi
>>>
========================= RESTAR
>>> g=simpson(f, 0, 4, 1)
>>> print(g)
14.602008682686735
>>> g=simpson(f, 0, 4, 2)
>>> print(g)
16.404808911392475
>>> g=simpson(f, 0, 4, 4)
>>> print(g)
16.535092753854382
>>> g=simpson(f, 0, 4, 100)
>>> print(g)
16.538339622856025
>>> g=simpson(f, 0, 4, 1000)
>>> print(g)
16.53833962927242
>>>
```

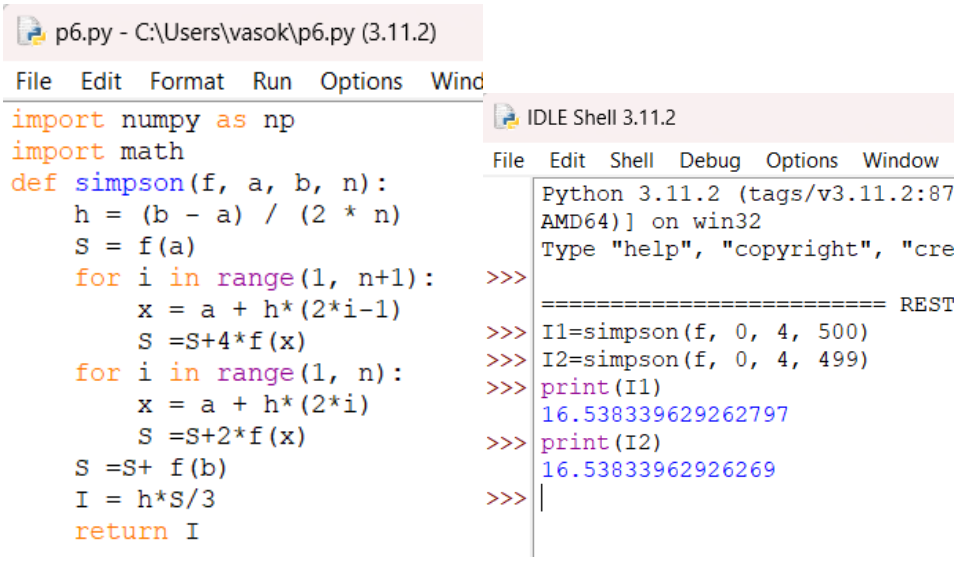
Κανόνας Simpson – Παράδειγμα 2

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

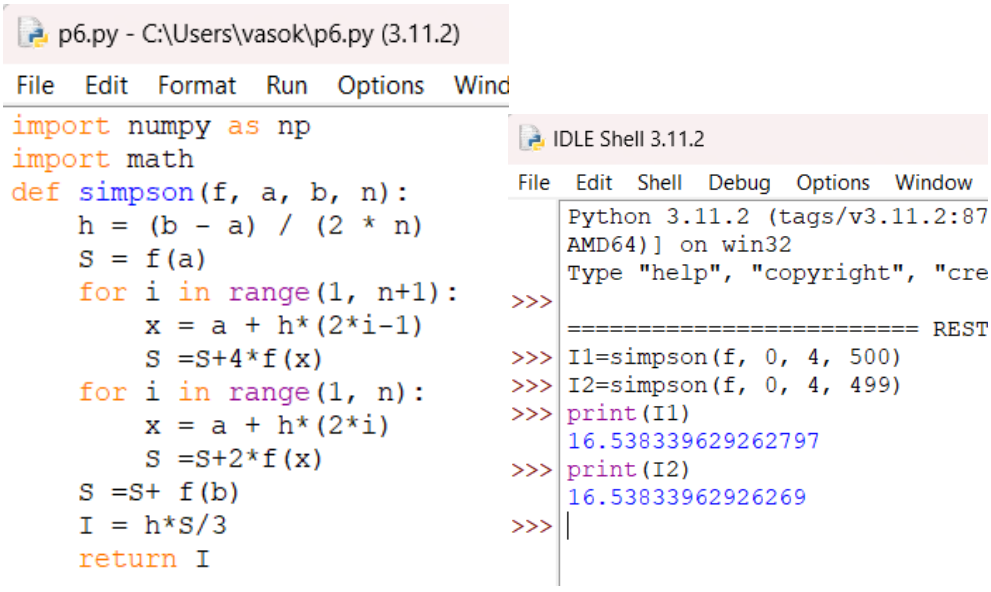
$$I = \int_0^4 (x \cdot \sin(2x) + 4)dx$$

με τον κανόνα του Simpson για n=500 και να βρεθεί ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.

- Σε Python για n=500 θα έχουμε



- Σε Python για n=499 θα έχουμε



- Συγκρίνουμε τις τιμές ως προς τα δεκαδικά ψηφία τους σύμφωνα με τον τύπο

$$|x_n - x_{n-1}| < \frac{1}{2}10^{-k} \Longrightarrow k < -\log\left(2 \cdot |x_n - x_{n-1}|\right)$$

όπου k το πλήθος των δεκαδικών ψηφίων.

- Σε Python θα έχουμε

```
>>> import math
>>> g=-math.log10(2*abs(I1-I2))
>>> print(g)
12.671288541487455
>>>
```

επειδή

$$k < -log(2 \cdot |I_{500} - I_{499}|) = 12.671288541487455$$

θα έχουμε

k=12

Επομένως, η προσεγγιστική λύση I_{500} έχει ακρίβεια 12 δεκαδικών ψηφίων.

Κανόνας Simpson – Άσκηση

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^3 e^{x^2} dx$$

με τον κανόνα του Simpson για n=100.

  - Απάντηση: (1444.5456964394289)
- Να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.
  - Απάντηση: (4 δεκαδικά ψηφία)

Συνδυαστική Άσκηση

- Να υπολογιστεί η τιμή του ορισμένου ολοκληρώματος

$$I = \int_0^3 e^{x^2} dx$$

με τον κανόνα του τραπεζίου και του Simpson για n=500.

  - Απάντηση:
    - Μέθοδος τραπεζίου: 1444.6909747280079
    - Μέθοδος Simpson: 1444.5451238115556
- Να βρεθεί η ακρίβεια των αποτελεσμάτων σε δεκαδικά ψηφία.
  - Απάντηση:
    - Μέθοδος τραπεζίου: 2.93170469768348 (2 δεκαδικά ψηφία)
    - Μέθοδος Simpson: 7.830546951121842 (7 δεκαδικά ψηφία)
- Ποια είναι η πιο ακριβής μέθοδος;
  - Απάντηση: Η μέθοδος Simpson

Ανάλυση Αλγορίθμων

Αλγόριθμος Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή

Ο αλγόριθμος **Μετατροπή** $([X])_b \rightarrow (\cdot)_{10}$ – **Πρώτη Εκδοχή** μετατρέπει το ακέραιο μέρος του αριθμού *x* από το αριθμητικό σύστημα με βάση *b* σε δεκαδικό σύστημα.

```
def b2decM(a,b):
    n=len(a)
    a=a[::-1] #Αντιστροφή Πίνακα
    y=0
    for i in range(n):
        y+=a[i]*b**(i)
    return y
```

Αλγόριθμος Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – **Πρώτη Εκδοχή**

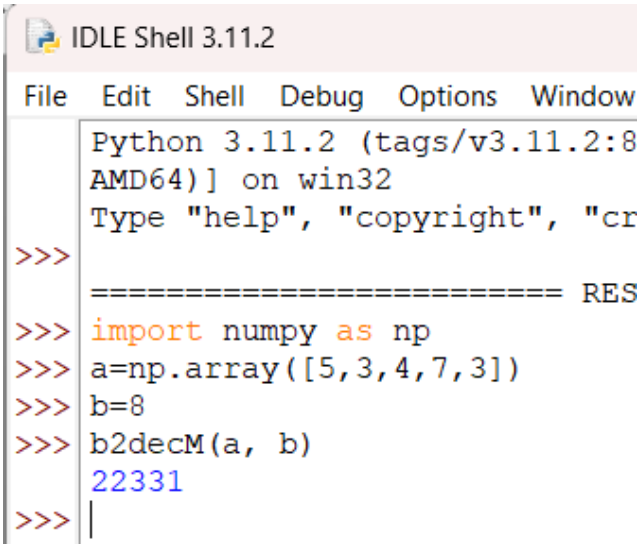
- Η συνάρτηση "**b2decM(a,b):**" μετατρέπει τον αριθμό "**a**" που δίνεται με βάση "**b**" σε δεκαδικό αριθμό. Η συνάρτηση δέχεται δύο ορίσματα, έναν ακέραιο αριθμό "**a**" σε μία βάση "**b**", όπου κάθε ψηφίο του "**a**" αντιπροσωπεύεται από έναν ακέραιο αριθμό μικρότερο ή ίσο του "**b**", και έναν ακέραιο αριθμό "**b**" που αντιπροσωπεύει τη βάση του αριθμού "**a**". Η μεταβλητή "**a**" είναι ένας πίνακας με στοιχεία τα ψηφία του αριθμού "**X**".
- Η εντολή "**n=len(a)**" υπολογίζει το μήκος του αριθμού "**a**", αποθηκεύοντας το στη μεταβλητή "**n**".
- Η εντολή "**a=a[::-1]**" αντικαθιστά την αρχική λίστα "**a**" με όλα τα στοιχεία της "**a**" σε αντίστροφη σειρά, ώστε να αναλύσει κάθε ψηφίο από τα δεξιά προς τα αριστερά.
- Η εντολή "**y=0**" αρχικοποιεί την μεταβλητή "**y**" με την τιμή μηδέν.
- Ο βρόγχος "**for i in range(n):**"
  "**y+=a[i]*b**(i)**"
  για κάθε ψηφίο του "**a**" από τα δεξιά προς τα αριστερά, αναλύει το ψηφίο σε δυνάμεις της βάσης "**b**" και πολλαπλασιάζει το κάθε ψηφίο με

την αντίστοιχη δύναμη της βάσης "**b**" και το προσθέτει στη μεταβλητή "**y**".

- Η εντολή "**return** **y**", επιστρέφει το "**y**", που είναι ο δεκαδικός αριθμός που αντιστοιχεί στον αριθμό "**a**" στη βάση "**b**".

Τρόπος Εκτέλεσης Αλγόριθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή

Για να εκτελεστεί ο αλγόριθμος " **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση Αλγόριθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή

- Η εντολή "**import** **numpy** **as** **np**" εισάγει την βιβλιοθήκη ***numpy*** στο πρόγραμμα και την ορίζει με το όνομα ***np*** για να μπορεί να την αναφέρει. Στη συνέχεια, μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις και τα αντικείμενα της ***numpy*** με την μορφή "**np.function()**" ή "**np.object**".
- Η εντολή "**a=np.array([5,3,4,7,3])**" δημιουργεί έναν πίνακα με τα στοιχεία 5, 3, 4, 7, 3 χρησιμοποιώντας τη συνάρτηση "**array()**" της βιβλιοθήκης **numpy**.
- Η εντολή "**b=8**" ορίζει την μεταβλητή "**b**" με την τιμή "**8**" για να χρησιμοποιηθεί στο υπόλοιπο του προγράμματος.

- Η εντολή "**b2decM(a,b)**" επιστρέφει το αποτέλεσμα του υπολογισμού για τα ορίσματα "**a**" και "**b**" που δίνουμε, δηλαδή τον αριθμό στο δεκαδικό σύστημα.

Αλγόριθμος Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Δεύτερη Εκδοχή

Ο αλγόριθμος **Μετατροπή** $([X])_b \rightarrow (\cdot)_{10}$ – **Δεύτερη Εκδοχή** μετατρέπει το ακέραιο μέρος του αριθμού *x* από το αριθμητικό σύστημα με βάση *b* σε δεκαδικό σύστημα.

```
def b2dec(x,b):
    xc=str(x) #Μετατροπή αριθμού σε συμβολοσειρά
    n=len(xc)
    a=[int(xc[n-i-1]) for i in range(n)] #Δημιουργία πίνακα α με αντίστροφη σειρά
    #και μετατροπή με int ε ακέραιο αριθμό
    y=0
    for i in range(n):
        y+=a[i]*b**(i)
    return y
```

Αλγόριθμος Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ **– Δεύτερη Εκδοχή**

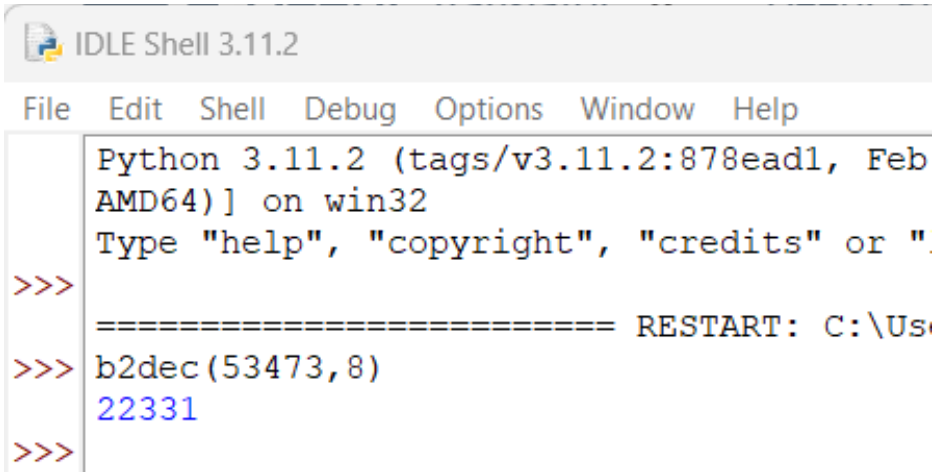
- Η συνάρτηση "**b2dec(x,b):**" μετατρέπει τον αριθμό "**x**" που δίνεται με βάση "**b**" σε δεκαδικό αριθμό. Η συνάρτηση δέχεται δύο ορίσματα, έναν ακέραιο αριθμό "**a**" σε μία βάση "**b**". Στα ορίσματα της συνάρτησης έχουμε τον αριθμό "**x**" ο οποίος χωρίζεται στα ψηφία του μέσα στη συνάρτηση.
- Η εντολή "**xc=str(x)**" μετατρέπει τον αριθμό "**x**" σε μια συμβολοσειρά "**xc**", ώστε να είναι εφικτή η πρόσβαση σε κάθε ψηφίο του αριθμού.
- Η εντολή "**n=len(xc)**" υπολογίζει το πλήθος των ψηφίων του αριθμού "**x**".
- Η εντολή "**a=[int(xc[n-i-1]) for i in range(n)]**" δημιουργεί μια λίστα "**a**" από τα ψηφία του "**x**", αντιστρέφοντας την σειρά τους. Η σύνταξη "**int(xc[n-i-1])**" αντλεί το i-οστό ψηφίο από το τέλος του αριθμού, χρησιμοποιώντας τον αντίστροφο δείκτη "**[n-i-1]**".
- Η εντολή "**y=0**" αρχικοποιεί την μεταβλητή "**y**" με την τιμή μηδέν.
- Ο βρόγχος "**for i in range(n):**"
  "**y+=a[i]*b**(i)**"

υπολογίζει τον δεκαδικό αριθμό που αντιστοιχεί στον ακέραιο αριθμό "**x**" στη βάση "**b**". Πιο συγκεκριμένα, επαναλαμβάνεται για κάθε ψηφίο του "**x**" στη βάση "**b**", ξεκινώντας από το λιγότερο σημαντικό ψηφίο (το "**a[0]**") μέχρι το πιο σημαντικό ψηφίο (το "**a[n-1]**"). Για κάθε ψηφίο στη βάση "**b**", υπολογίζεται η αντίστοιχη δύναμη της βάσης "**b**" (δηλαδή "**b******i**") και πολλαπλασιάζεται με την τιμή του ψηφίου "**a[i]**". Οι παραγόμενοι όροι προστίθενται μεταξύ τους σε μια μεταβλητή "**y**", η οποία και επιστρέφεται σαν τον δεκαδικό αντίστοιχο του αριθμού "**x**" στη βάση "**b**".

- Η εντολή "**return y**", επιστρέφει το "**y**", που είναι ο δεκαδικός αριθμός που αντιστοιχεί στον αριθμό "**x**" στη βάση "**b**".

Τρόπος Εκτέλεσης Αλγόριθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Δεύτερη Εκδοχή

Για να εκτελεστεί ο αλγόριθμος " **Μετατροπή** $([X])_b \rightarrow (\cdot)_{10}$ – **Δεύτερη Εκδοχή**" πρέπει να εκτελεστεί η παρακάτω εντολή στο IDLE Shell της Python.



Εκτέλεση Αλγόριθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – **Δεύτερη Εκδοχή**

- Η εντολή "**b2dec(x,b)**" επιστρέφει το αποτέλεσμα του υπολογισμού για τα ορίσματα "**x**" και "**b**" που δίνουμε, δηλαδή τον αριθμό στο δεκαδικό σύστημα.

Αλγόριθμος Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Πρώτη Εκδοχή)

Ο αλγόριθμος **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Πρώτη Εκδοχή)** μετατρέπει το ακέραιο μέρος του αριθμού *x* από το αριθμητικό σύστημα με βάση *b* σε δεκαδικό σύστημα ακολουθώντας την τακτική του σχήματος Horner, ώστε να μειωθεί ο αριθμός των πράξεων.

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def b2decMH(a, b):
    n = len(a)
    a = a[::-1]
    y = a[n-1]
    for i in range(n-2, -1,-1):
        y = a[i] + y*b
    return y
```

Αλγόριθμος Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ **– Σχήμα Horner (Πρώτη Εκδοχή)**

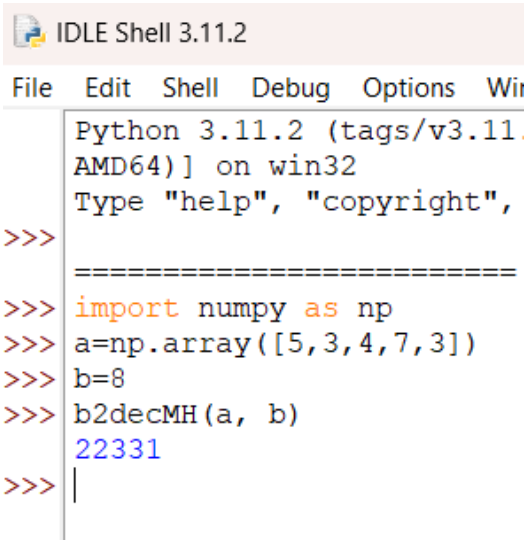
- Η συνάρτηση "**b2decMH(a,b):**" μετατρέπει τον αριθμό "**a**" που δίνεται με βάση "**b**" σε δεκαδικό αριθμό. Η συνάρτηση δέχεται δύο ορίσματα, έναν ακέραιο αριθμό "**a**" σε μία βάση "**b**". Η μεταβλητή "**a**" είναι ένας πίνακας με στοιχεία τα ψηφία του αριθμού "**X**".
- Η εντολή "**n=len(a)**" υπολογίζει το μήκος του αριθμού "**a**", αποθηκεύοντας το στη μεταβλητή "**n**".
- Η εντολή "**a=a[::-1]**" αντικαθιστά την αρχική λίστα "**a**" με όλα τα στοιχεία της "**a**" σε αντίστροφη σειρά. Έτσι, η μεταβλητή "**a**" περιέχει τα στοιχεία της αρχικής λίστας σε αντίστροφη σειρά μετά από αυτή την εντολή.
- Η εντολή "**y = a[n-1]**" αποθηκεύει στη μεταβλητή "**y**" το τελευταίο στοιχείο του πίνακα "**a**".
- Ο βρόγχος "**for i in range(n-2,-1,-1):**"
  "**y = a[i] + y*b**"
  εκτελεί επαναλήψεις από το "**n-2**" έως το "**0**", με βήμα "**-1**". Σε κάθε επανάληψη, αυξάνει το "**y**" κατά τον πολλαπλασιασμό του τρέχοντος ψηφίου του δεκαδικού αριθμού με την κατάλληλη δύναμη της βάσης "**b**". Στη συνέχεια προσθέτει το αποτέλεσμα "**y**" ώστε να ενημερωθεί για τα

επόμενα ψηφία. Στο τέλος, η μεταβλητή "**y**" θα περιέχει το αποτέλεσμα του υπολογισμού.

- Η εντολή "**return** **y**", επιστρέφει το "**y**", που είναι ο δεκαδικός αριθμός που αντιστοιχεί στον αριθμό "**X**" στη βάση "**b**".

Τρόπος Εκτέλεσης Αλγόριθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Πρώτη Εκδοχή)

Για να εκτελεστεί ο αλγόριθμος " **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Πρώτη Εκδοχή)**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση Αλγορίθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Πρώτη Εκδοχή)

- Οι εντολές εκτέλεσης του αλγόριθμου " **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Πρώτη Εκδοχή)**" είναι ίδιες με την εκτέλεση του αλγόριθμου " **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή** ", οι οποίες αναλύονται παραπάνω στην υπό-ενότητα "**Τρόπος εκτέλεσης αλγόριθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή** "

Αλγόριθμος Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Δεύτερη Εκδοχή)

Ο αλγόριθμος **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Δεύτερη Εκδοχή)** μετατρέπει το ακέραιο μέρος του αριθμού *x* από το αριθμητικό σύστημα με βάση *b* σε δεκαδικό σύστημα ακολουθώντας την τακτική του σχήματος Horner, ώστε να μειωθεί ο αριθμός των πράξεων.

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

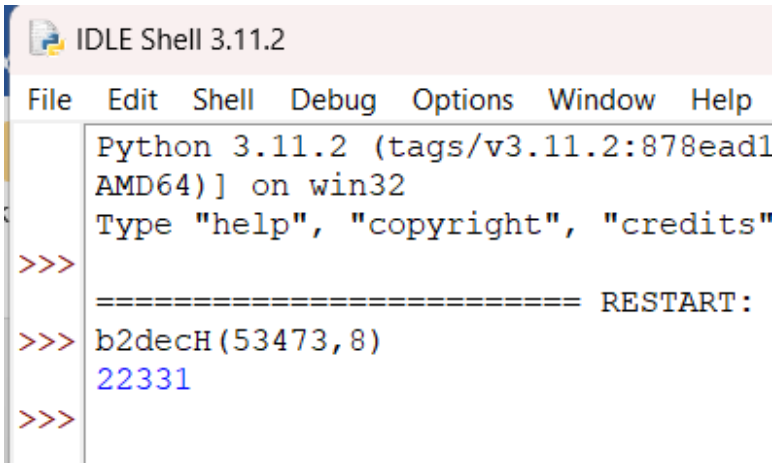
```
def b2decH(x,b):
    xc = str(x)
    n = len(xc)
    a = [int(xc[n-i-1]) for i in range(n)]
    y = a[n-1]
    for i in range(n-2,-1,-1):
        y = a[i] + b*y
    return y
```

Αλγόριθμος Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ **– Σχήμα Horner (Δεύτερη Εκδοχή)**

- Η συνάρτηση "**b2decH(x,b):**" μετατρέπει τον αριθμό "**x**" που δίνεται με βάση "**b**" σε δεκαδικό αριθμό. Η συνάρτηση δέχεται δύο ορίσματα, έναν ακέραιο αριθμό "**x**" σε μία βάση "**b**". Στα ορίσματα της συνάρτησης έχουμε τον αριθμό "**x**" ο οποίος χωρίζεται στα ψηφία του μέσα στη συνάρτηση.
- Οι εντολές "**xc=str(x)**", "**n=len(xc)**" και "**a=[int(xc[n-i-1]) for i in range(n)]**" αναλύθηκαν παραπάνω στην υπό-ενότητα " **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Δεύτερη Εκδοχή**".
- Οι εντολές "**y = a[n-1]**", "**for i in range(n-2,-1,-1):**" , "**y = a[i] + y*b**" και "**return y**" αναλύθηκα παραπάνω στην υπό-ενότητα Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – **Σχήμα Horner (Πρώτη Εκδοχή)**.

Τρόπος Εκτέλεσης Αλγόριθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Δεύτερη Εκδοχή)

Για να εκτελεστεί ο αλγόριθμος " **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Δεύτερη Εκδοχή)**" πρέπει να εκτελεστεί η παρακάτω εντολή στο IDLE Shell της Python.



Εκτέλεση Αλγορίθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Σχήμα Horner (Δεύτερη Εκδοχή)

- Η εντολή "**b2decH(x,b)**" επιστρέφει το αποτέλεσμα του υπολογισμού για τα ορίσματα "**x**" και "**b**" που δίνουμε, δηλαδή τον αριθμό στο δεκαδικό σύστημα.

Αλγόριθμος Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή

Ο αλγόριθμος **Μετατροπή** $(X-[X])_b \rightarrow (\cdot)_{10}$ **– Πρώτη Εκδοχή** μετατρέπει το κλασματικό μέρος του αριθμού *x* από το αριθμητικό σύστημα με βάση *b* σε δεκαδικό σύστημα.

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window He

```
def b2decFMH(a, b):
    n = len(a)
    y = 0
    for i in range(n-1, -1, -1):
        y = (a[i] + y) / b
    return y
```

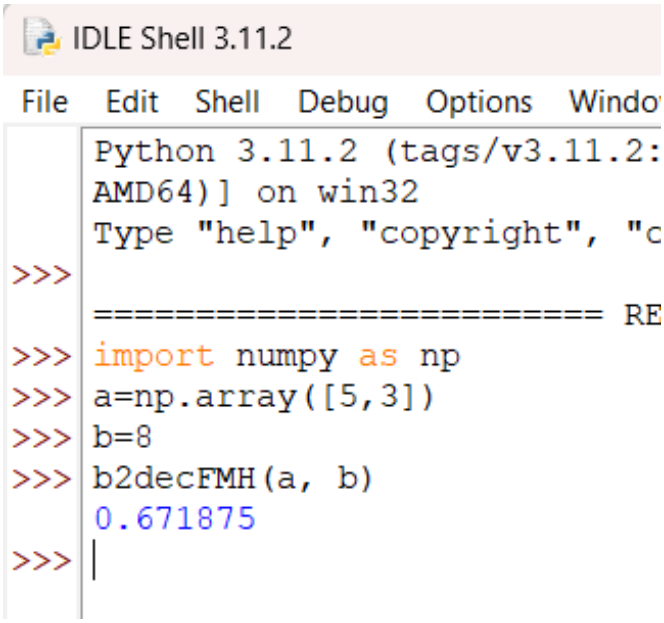
Αλγόριθμος Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ **– Πρώτη Εκδοχή**

- Η συνάρτηση "**b2decFMH(a,b):**" μετατρέπει το κλασματικό μέρος του αριθμού "**a**" που δίνεται, από αριθμητικό σύστημα με βάση "**b**" σε δεκαδικό σύστημα. Η συνάρτηση δέχεται δύο ορίσματα, έναν ακέραιο αριθμό "**a**" σε μία βάση "**b**" και έναν ακέραιο αριθμό "**b**" που αντιπροσωπεύει τη βάση του αριθμού "**a**". Η μεταβλητή "**a**" είναι ένας πίνακας με στοιχεία τα ψηφία του αριθμού "**X**".
- Οι εντολές "**n=len(a)**" και "**y=0**" αναλύθηκα παραπάνω στην υπό-ενότητα Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ **– Σχήμα Horner (Πρώτη Εκδοχή).**
- Ο βρόγχος "**for i in range(n-1, -1, -1):**
  "**y = (a[i] + y) / b**"
  "**return y**"
  εκτελεί επαναλήψεις από το "**n-1**" έως το "**0**", με βήμα "**-1**". Ξεκινάει από το τελευταίο ψηφίο του αριθμού "**a**" και προχωράει προς τα αριστερά, διαιρώντας κάθε ψηφίο με την βάση "**b**" και προσθέτοντας το υπόλοιπο στο αποτέλεσμα "**y**" . Μόλις ολοκληρωθεί ο βρόγχος, η

μεταβλητή "**y**" θα περιέχει την δεκαδική αναπαράσταση του αριθμού "**a**" στην βάση "**b**".

Τρόπος Εκτέλεσης Αλγόριθμου Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή

Για να εκτελεστεί ο αλγόριθμος " **Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση Αλγόριθμου Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή

- Οι εντολές εκτέλεσης του αλγόριθμου " **Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή** " είναι ίδιες με την εκτέλεση του αλγόριθμου " **Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή** ", οι οποίες αναλύονται παραπάνω στην υπό-ενότητα "**Τρόπος εκτέλεσης αλγόριθμου Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Πρώτη Εκδοχή** "

Αλγόριθμος Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Δεύτερη Εκδοχή

Ο αλγόριθμος **Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Δεύτερη Εκδοχή** μετατρέπει το κλασματικό μέρος του αριθμού *x* από το αριθμητικό σύστημα με βάση *b* σε δεκαδικό σύστημα.

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def b2decFH(x, b):
    xc = str(x)
    n = len(xc) - 2
    a = [int(xc[i + 2]) for i in range(n)]
    y = 0
    for i in range(n - 1, -1, -1):
        y = (a[i] + y) / b
    return y
```

Αλγόριθμος Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Δεύτερη Εκδοχή

- Η συνάρτηση "**b2decFH(x,b):**" μετατρέπει το κλασματικό μέρος του αριθμού "**x**" που δίνεται, από αριθμητικό σύστημα με βάση "**b**" σε δεκαδικό σύστημα. Η συνάρτηση δέχεται δύο ορίσματα, έναν ακέραιο αριθμό "**x**" σε μία βάση "**b**" και έναν ακέραιο αριθμό "**b**" που αντιπροσωπεύει τη βάση του αριθμού "**a**". Στα ορίσματα της συνάρτησης έχουμε τον αριθμό "**x**" ο οποίος χωρίζεται στα ψηφία του μέσα στη συνάρτηση.
- Η εντολή "**xc=str(x)**", αναλύθηκε παραπάνω στην υπό-ενότητα "**Μετατροπή $([X])_b \rightarrow (\cdot)_{10}$ – Δεύτερη Εκδοχή**".
- Η εντολή "**n=len(xc) - 2**" υπολογίζει το μήκος του αλφαριθμητικού "**xc**" και αφαιρεί 2 από το μήκος, καθώς το πρώτο ψηφίο από τα αριστερά αποτελεί το ακέραιο μέρος του αριθμού που δεν χρειάζεται να υπολογίζεται ξανά.
- Η εντολή "**a = [int(xc[i + 2]) for i in range(n)]**" δημιουργεί μια λίστα "**a**" από τα αριθμητικά ψηφία του αριθμού "**x**", αγνοώντας τα δύο πρώτα ψηφία (καθώς αντιστοιχούν στο πρόθεμα του αριθμού). Για κάθε θέση "**i**" της λίστας "**a**", προσθέτει το αντίστοιχο ψηφίο του "**x**"

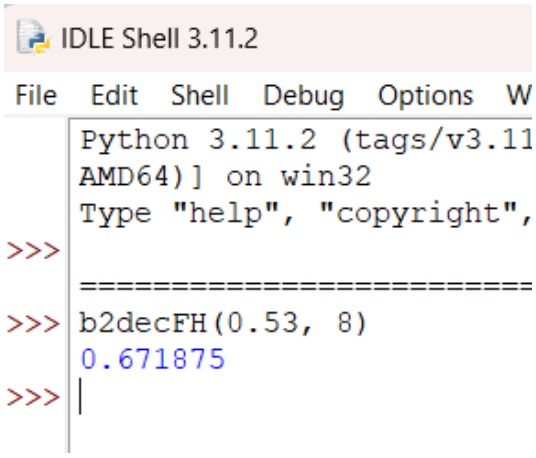
στην θέση "**i+2**" και το μετατρέπει σε ακέραιο αριθμό. Στο τέλος, η λίστα "**a**" περιέχει τα ψηφία του αριθμού "**x**", εκτός από τα δύο πρώτα.

- Η εντολή "**y=0**" αναλύθηκε παραπάνω στην υπό-ενότητα **Μετατροπή** $([X])_b \rightarrow (\cdot)_{10}$ – **Σχήμα Horner (Πρώτη Εκδοχή)**.
- Ο βρόγχος "**for i in range(n-1, -1, -1):**
  "**y = (a[i] + y) / b**"
  "**return y**"

  Αναλύθηκε παραπάνω στην υπό-ενότητα **Μετατροπή** $(X-[X])_b \rightarrow (\cdot)_{10}$ – **Πρώτη Εκδοχή**.

Τρόπος Εκτέλεσης Αλγόριθμου Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – Δεύτερη Εκδοχή

Για να εκτελεστεί ο αλγόριθμος " **Μετατροπή** $(X-[X])_b \rightarrow (\cdot)_{10}$ – **Δεύτερη Εκδοχή**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση Αλγόριθμου Μετατροπή $(X-[X])_b \rightarrow (\cdot)_{10}$ – **Δεύτερη Εκδοχή**

- Η εντολή "**b2decFH(x,b)**" επιστρέφει το αποτέλεσμα του υπολογισμού για τα ορίσματα "**x**" και "**b**" που δίνουμε, δηλαδή το ακέραιο μέρος του αριθμού "**x**" στο δεκαδικό σύστημα.

Αλγόριθμος Μετατροπή $([X])_{10} \rightarrow (\cdot)_b$ – (Αλγόριθμος Ευκλείδειας Διαίρεσης)

Ο αλγόριθμος **Μετατροπή $([X])_{10} \rightarrow (\cdot)_b$ – (Αλγόριθμος Ευκλείδειας Διαίρεσης)** *μετατρέπει τον ακέραιο αριθμό x από το δεκαδικό σύστημα σε βάση b.*

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def dec2b(x, b):
    a = []
    i = 1
    while x != 0:
        a.append(x % b)
        x = round(x // b * 1000) / 1000
        i += 1
    return a[::-1]
```

Αλγόριθμος Μετατροπή $([X])_{10} \rightarrow (\cdot)_b$ – (Αλγόριθμος Ευκλείδειας Διαίρεσης)

- Η συνάρτηση "**dec2b(x,b):**" μετατρέπει τον ακέραιο αριθμό "**x**" από το δεκαδικό σύστημα που δίνεται στην βάση "**b**". Η συνάρτηση δέχεται δύο ορίσματα, έναν ακέραιο αριθμό "**x**" του δεκαδικού συστήματος και έναν ακέραιο αριθμό "**b**" που αντιπροσωπεύει τη βάση στην οποία θέλουμε να μετατραπεί ο αριθμός "**a**". Η μεταβλητή "**a**" είναι τα ψηφία του αριθμού "**x**" που μετατράπηκε.
- Η εντολή "**a = []**" δημιουργεί μια κενή λίστα με όνομα "**a**".
- Η εντολή "**i = 1**" δηλώνει μια νέα μεταβλητή "**i**" και αρχικοποιεί την τιμή της σε 1. Αυτή η εντολή χρησιμοποιείται στο επόμενο βήμα του βρόγχου για τον έλεγχο της συνθήκης επανάληψης.
- Ο βρόγχος "**while x != 0:**"
  "**a.append(x % b)**"
  "**x = round(x // b * 1000) / 1000**"
  "**i += 1**"

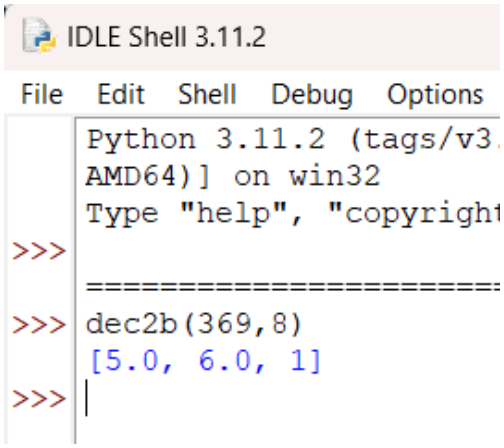
  επαναλαμβάνει το κομμάτι του κώδικα που βρίσκεται μέσα στον βρόγχο μέχρι η τιμή της μεταβλητής "**x**" να γίνει μηδέν. Κάθε φορά, θα προστίθεται το υπόλοιπο της διαίρεσης του "**x**" με την βάση "**b**" στη

λίστα "**a**", και μετά θα ανανεώνει το "**x**" ως το ακέραιο μέρος της διαίρεσης του παλιού "**x**" με τη βάση "**b**", συν το 1/1000 του δεκαδικού μέρους της διαίρεσης του παλιού "**x**" με τη βάση "**b**", σε στρογγυλοποίηση στο τρίτο δεκαδικό ψηφίο, ώστε να αποφευχθούν στρογγυλοποιητικά σφάλματα. Η μεταβλητή "**i**" χρησιμοποιείται για καθαρά λειτουργικούς λόγους και δεν επηρεάζει το αποτέλεσμα του κώδικα.

- Η εντολή " **return a[::-1]**", επιστρέφει την λίστα "**a**", η οποία αντιστρέφεται με το "**[::-1]**" για να εμφανιστεί στη σωστή σειρά των ψηφίων της νέας βάσης.

Τρόπος Εκτέλεσης Αλγόριθμου Μετατροπή $([X])_{10} \rightarrow (\cdot)_b$ – (Αλγόριθμος Ευκλείδειας Διαίρεσης)

Για να εκτελεστεί ο αλγόριθμος " **Μετατροπή $([X])_{10} \rightarrow (\cdot)_b$ – (Αλγόριθμος Ευκλείδειας Διαίρεσης)**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση Αλγόριθμου Μετατροπή $([X])_{10} \rightarrow (\cdot)_b$ – (Αλγόριθμος Ευκλείδειας Διαίρεσης)

- Η εντολή "**dec2b(x,b)**" επιστρέφει το αποτέλεσμα του υπολογισμού για τα ορίσματα "**x**" και "**b**" που δίνουμε, δηλαδή το ακέραιο μέρος του αριθμού "**x**" στο σύστημα βάσης "**b**" που δόθηκε .

Αλγόριθμος Μετατροπή $(X-[X])_{10} \rightarrow (\cdot)_b$

Ο αλγόριθμος **Μετατροπή** $(X-[X])_{10} \rightarrow (\cdot)_b$ *μετατρέπει τον κλασματικό αριθμό x από το δεκαδικό σύστημα σε βάση b.*

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def dec2bF(x, b):
    y = b * x
    a = []
    i = 1
    while y != 0:
        a.append(round(int(y) * 1000) / 1000)
        y = (y - int(y)) * b
        i += 1
    return a
```

Αλγόριθμος Μετατροπή $(X-[X])_{10} \rightarrow (\cdot)_b$

- Η συνάρτηση "**dec2bF(x,b):**" μετατρέπει τον κλασματικό αριθμό "**x**" από το δεκαδικό σύστημα που δίνεται στην βάση "**b**". Η συνάρτηση δέχεται δύο ορίσματα, έναν κλασματικό αριθμό "**x**" του δεκαδικού συστήματος και έναν ακέραιο αριθμό "**b**" που αντιπροσωπεύει τη βάση στην οποία θέλουμε να μετατραπεί ο αριθμός "**a**". Η μεταβλητή "**a**" είναι τα ψηφία του αριθμού "**x**" που μετατράπηκε.
- Η εντολή "**y = b * x**" πολλαπλασιάζει τον κλασματικό αριθμό "**x**" με την βάση "**b**" και την αποθηκεύει στην μεταβλητή "**y**".
- Η εντολή " **a = []**" και η " **i = 1**" αναλύθηκε παραπάνω στην υπό-ενότητα " **Μετατροπή** $([X])_{10} \rightarrow (\cdot)_b$ **- (Αλγόριθμος Ευκλείδειας Διαίρεσης)**".
- *Ο* βρόγχος "**while y != 0:**"
  "**a.append(round(int(y) * 1000) / 1000)**"
  "**y = (y - int(y)) * b**"
  "**i += 1**"

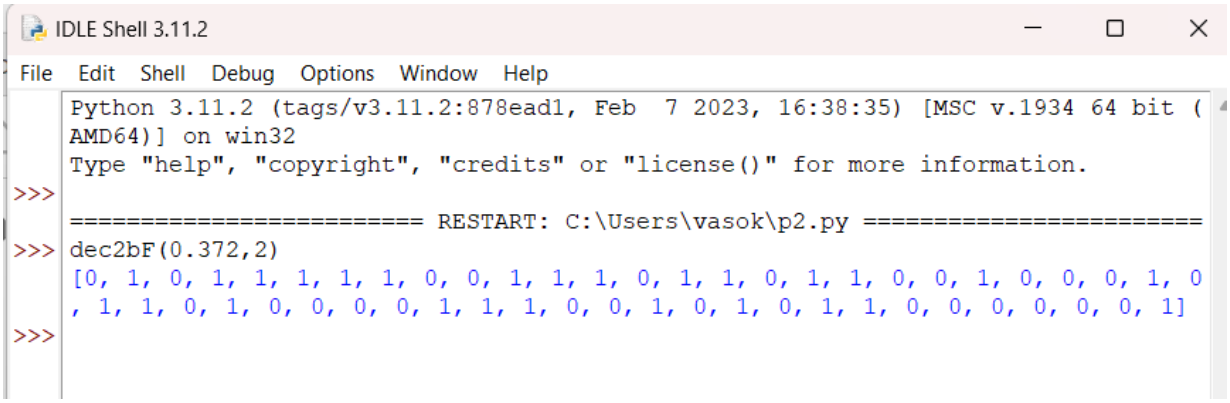
  επαναλαμβάνει το κομμάτι του κώδικα που βρίσκεται μέσα στον βρόγχο μέχρι η τιμή της μεταβλητής "**y**" να γίνει μηδέν. *Ο* αριθμός "**y**" που υπολογίστηκε σε προηγούμενο βήμα αποθηκεύεται στον πίνακα "**a**" το

ακέραιο μέρος του "**y**" (δηλαδή το ψηφίο που αντιστοιχεί στην τρέχουσα θέση στον αριθμό που μετατρέπουμε). Έπειτα, υπολογίζεται ο νέος αριθμός "**y**" ως το υπόλοιπο του "**y**" μετά την αφαίρεση του ακέραιου μέρους του, πολλαπλασιασμένο με τη βάση "**b**".

- Η εντολή "**return a**" επιστρέφει τον πίνακα "**a**", ο οποίος περιέχει τα ψηφία του αριθμού "**x**" στη βάση "**b**".

Τρόπος Εκτέλεσης Αλγόριθμου Μετατροπή $(X-[X])_{10} \rightarrow (\cdot)_b$

Για να εκτελεστεί ο αλγόριθμος " **Μετατροπή** $(X-[X])_{10} \rightarrow (\cdot)_b$" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.

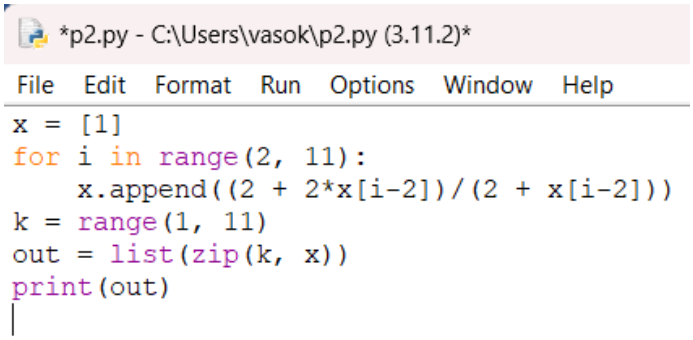


Εκτέλεση Αλγόριθμου Μετατροπή $(X-[X])_{10} \rightarrow (\cdot)_b$

- Η εντολή "**dec2bF(x,b)**" επιστρέφει το αποτέλεσμα του υπολογισμού για τα ορίσματα "**x**" και "**b**" που δίνουμε, δηλαδή τον πίνακα "**a**", ο οποίος περιέχει τα ψηφία του αριθμού "**x**" που μετατράπηκε στη βάση "**b**" που δόθηκε.

Αλγόριθμος Επαναληπτική Μέθοδος

Ο αλγόριθμος **Επαναληπτική Μέθοδος** υπολογίζει την ρίζα της εξίσωσης $x^2 - 2 = 0$ με χρήση του αναδρομικού τύπου $x_n = \frac{2+2x_{n-1}}{2+x_{n-1}}$ με αρχική τιμή $x_1 = 1$ και βρίσκει την τιμή του x_{10} .

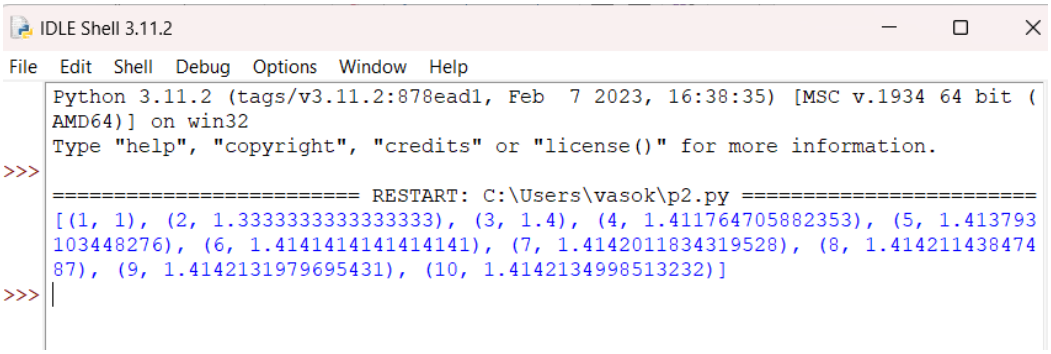


Αλγόριθμος Επαναληπτική Μέθοδος

- Η εντολή "**x = [1]**" δημιουργεί την λίστα "**x**" με μόνο ένα στοιχείο την τιμή 1.
- Ο βρόγχος "**for i in range(2, 11):**"
  "**x.append((2 + 2*x[i-2])/(2 + x[i-2]))**"
  για κάθε αριθμό "**i**" στο εύρος "**2 έως 11**", υπολογίζει τον αριθμό "**x[i]**" ως "**(2 + 2*x[i-2])/(2 + x[i-2])**" και τον προσθέτει στην λίστα "**x**".
- Η εντολή "**k = range(1, 11)**" δημιουργεί την λίστα "**k**" με τις τιμές "**1 έως 10**".
- Η εντολή "**out = list(zip(k, x))**" χρησιμοποιώντας την συνάρτηση "**zip()**", με την οποία συνδυάζονται οι λίστες "**k**" και "**x**" σε ζεύγη (το πρώτο στοιχείο της "**k**" με το πρώτο στοιχείο της "**x**", κ.ο.κ.), και αποθηκεύονται στην λίστα "**out**".
- Η εντολή "**print(out)**" εκτυπώνει στην οθόνη την λίστα "**out**".

Τρόπος Εκτέλεσης Αλγόριθμου Επαναληπτική Μέθοδος

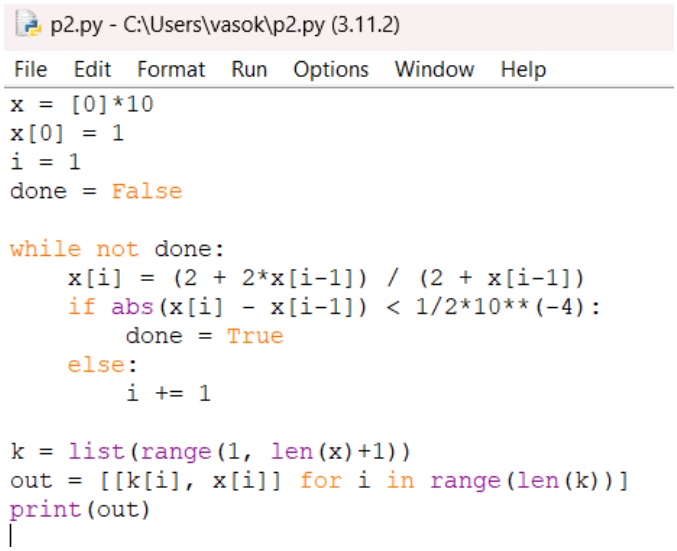
Για να εκτελεστεί ο αλγόριθμος " **Επαναληπτική Μέθοδος**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εκτυπώνεται η λίστα "**out**" όπως φαίνεται παρακάτω.



Εκτέλεση Αλγόριθμου Επαναληπτική Μέθοδος

Αλγόριθμος Επαναληπτική Μέθοδος με κριτήριο τερματισμού

Ο αλγόριθμος **Επαναληπτική Μέθοδος με κριτήριο τερματισμού** υπολογίζει την ρίζα της εξίσωσης $x^2 - 2 = 0$ με χρήση του αναδρομικού τύπου $x_n = \frac{2+2x_{n-1}}{2+x_{n-1}}$ με αρχική τιμή $x_1 = 1$ και βρίσκει τις τιμές του x_n μέχρι να ισχύει το κριτήριο τερματισμού $|x_n - x_{n-1}| < \frac{1}{2}10^{-4}$ δηλαδή, η προσεγγιστική λύση να έχει ακρίβεια τεσσάρων δεκαδικών ψηφίων.



Αλγόριθμος Επαναληπτική Μέθοδος με κριτήριο τερματισμού

- Η εντολή "**x = [0]*10**" δημιουργεί την λίστα "**x**" με μέγεθος 10 και αρχικοποιεί όλα τα στοιχεία της με την τιμή 0.
- Η εντολή "**x[0] = 1**" ορίζει το πρώτη στοιχείο της λίστα με την τιμή 1 έτσι ώστε να ξεκινήσει η αναδρομή από το 1.
- Η εντολή "**i = 1**" ορίζει την μεταβλητή "**i**" με την τιμή 1 και η εντολή "**done = False**" ορίζει την μεταβλητή "**done**" ως False για να ξεκινήσει η επανάληψη.
- Ο βρόγχος "**while not done:**"

  "**x[i] = (2 + 2*x[i-1]) / (2 + x[i-1])**"

  "**if abs(x[i] - x[i-1]) < 1/2*10**(-4):**"

  "**done = True**"

  "**else:**"

  "**i += 1**"

  υπολογίζει την τρέχουσα τιμή "**x[i]**" και αποθηκεύεται στην αντίστοιχη θέση στην λίστα "**x**". Στην συνέχεια, γίνεται έλεγχος αν ισχύει το κριτήριο τερματισμού. Αν ισχύει, τότε τερματίζει και το "**done**" γίνεται True. Διαφορετικά, αυξάνετε η μεταβλητή "**i**" κατά 1 και συνεχίζεται η επανάληψη μέχρι να ισχύει το κριτήριο τερματισμού.
- Η εντολή " **k = list(range(1, len(x)+1))**" δημιουργεί μια λίστα "**k**" η οποία περιέχει τους ακέραιους από 1 έως 10.
- Η εντολή "**out = [[k[i], x[i]] for i in range(len(k))]**" δημιουργεί μια λίστα "**out**", που περιέχει τα στοιχεία της λίστας "**k**" και της λίστας που δημιουργήθηκαν προηγουμένως στον κώδικα. Συγκεκριμένα, δημιουργείτε μια νέα λίστα που περιέχει λίστες με δύο στοιχεία, όπου το πρώτο στοιχείο είναι ο αριθμός του στοιχείου στη λίστα "**k**" και το δεύτερο στοιχείο είναι το αντίστοιχο στοιχείο στη λίστα "**x**".
- Η εντολή "**print(out)**" εκτυπώνει στην οθόνη την λίστα "**out**".

Τρόπος Εκτέλεσης Αλγόριθμου Επαναληπτική Μέθοδος με κριτήριο τερματισμού

Για να εκτελεστεί ο αλγόριθμος " **Επαναληπτική Μέθοδος με κριτήριο τερματισμού**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εκτυπώνεται η λίστα "**out**" όπως φαίνεται παρακάτω.

```
>>>
========================= RESTART: C:\Users\vasok\p2.py =========================
[[1, 1], [2, 1.3333333333333333], [3, 1.4], [4, 1.411764705882353], [5, 1.413793
103448276], [6, 1.4141414141414141], [7, 1.4142011834319528], [8, 1.414211438474
87], [9, 0], [10, 0]]
>>>
```

Εκτέλεση Αλγόριθμου Επαναληπτική Μέθοδος με κριτήριο τερματισμού

Αλγόριθμος Μέθοδος Διχοτόμησης - Υλοποίηση

Ο αλγόριθμος **Μέθοδος Διχοτόμησης - Υλοποίηση** υπολογίζει την ρίζα της εξίσωσης $f(x) = 0$ στο διάστημα [a,b] όπου ισχύει $f(a) \cdot f(b) < 0$.

p2.py - C:\Users\vasok\p2.py (3.11.2)

File Edit Format Run Options Window Help

```
def bisection(f, a, b, tol, n):
    if f(a)*f(b) > 0.0:
        raise ValueError('function has same sign at end points')
    i = 1
    while i <= n:
        x = (a + b) / 2
        if f(x) == 0 or (b - a) / 2 < tol:
            print('The solution found')
            print(x)
            break
        if f(a)*f(x) > 0:
            a = x
        else:
            b = x
        i = i + 1
```

Αλγόριθμος Μέθοδος Διχοτόμησης - Υλοποίηση

- Η συνάρτηση "**bisection(f, a, b, tol, n):**" εφαρμόζει την μέθοδο της διχοτόμησης για τον εντοπισμό της ρίζας μιας συνάρτησης στο διάστημα "**[a,b]**". Τα ορίσματα της συνάρτησης είναι:
  - "**f**" : μια συνάρτηση "**f**" που αντιστοιχεί στην εξίσωση που θέλουμε να λύσουμε
  - "**[a,b]**" : ένα διάστημα "**[a,b]**"
  - "**tol**" : η ακρίβεια σε δεκαδικά ψηφία
  - "**n**" : ο μέγιστος αριθμός επαναλήψεων
- Στον έλεγχο "**if f(a)*f(b) > 0.0:**"

  "**raise ValueError('function has same sign at end points')**"

  ελέγχεται αν οι τιμές της συνάρτησης "**f**" έχουν το ίδιο πρόσημο στα άκρα "**a**" και "**b**". Αν ισχύει, τότε η συνάρτηση δεν μπορεί να έχει ρίζα μέσα στο διάστημα "**[a,b]**" και επομένως το πρόγραμμα αναγκάζεται να τερματίσει και να εμφανίσει ένα μήνυμα σφάλματος.
- Η εντολή "**i = 1**" ορίζει την μεταβλητή "**i**" με την τιμή 1.
- Ο βρόγχος "**while i <= n:**"

  "**x = (a + b) / 2**"

  "**if f(x) == 0 or (b - a) / 2 < tol:**"

  "**print('The solution found')**"

  "**print(x)**"

  "**break**"

  "**if f(a)*f(x) > 0:**"

  "**a=x**"

  "**else:**"

  "**b = x**"

  "**i = i + 1**"

  επαναλαμβάνεται όσο η τιμή του "**i**" είναι μικρότερη ίση με την τιμή των επαναλήψεων "**n**" και κάνει τι ακόλουθες ενέργειες. Υπολογίζει την τιμή του "**x**", η οποία είναι το μέσο του διαστήματος "**[a,b]**". Ελέγχει αν το "**f(x)**" είναι μηδέν ή αν το "**(b - a) / 2** " είναι μικρότερο από την ακρίβεια των δεκαδικών ψηφίων και αν είναι τότε εμφανίζει την λύση "**x**" και σταματάει τις επαναλήψεις. Στην συνέχεια, ελέγχεται αν η συνάρτηση έχει το ίδιο πρόσημο στα άκρα του διαστήματος και αν έχει εκχωρεί την τιμή του "**x**" στο "**a**", αλλιώς εκχωρεί την τιμή του "**x**" στο "**b**". Τέλος, αυξάνει τον μετρητή "**i**" κατά 1.

Αλγόριθμος Μέθοδος Διχοτόμησης – Παράδειγμα 1

Ο αλγόριθμος **Μέθοδος Διχοτόμησης – Παράδειγμα 1** υπολογίζει την ρίζα της εξίσωσης $f(x) = x^3 - 2x - 5$ στο διάστημα [1,3] με ακρίβεια 4 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.

p3.py - C:/Users/vasok/p3.py (3.11.2)

File Edit Format Run Options Window Help

```
def bisect(f, a, b, tol, n):
    if f(a)*f(b) > 0.0:
        raise ValueError('function has same sign at end points')

    a_ = [a]
    b_ = [b]
    x = [(a+b)/2]

    for i in range(n):
        if f(x[i]) == 0 or (b_[i]-a_[i])/2 < tol:
            print('Bisection method has converged')
            break

        if f(a_[i])*f(x[i]) > 0:
            a_.append(x[i])
            b_.append(b_[i])
        else:
            a_.append(a_[i])
            b_.append(x[i])

        x.append((a_[i+1]+b_[i+1])/2)

    if i == n-1:
        print('Zero not found to desired tolerance')

    k = list(range(len(x)))
    out = list(zip(k, a_, b_, x, [f(i) for i in x]))

    return out
def f(x):
    return x**3 - 2*x - 5

out = bisect(f, 1, 3, 1/2*10**(-4), 50)
print(out)
```

Αλγόριθμος Μέθοδος Διχοτόμησης – Παράδειγμα 1

- Η συνάρτηση "**bisect(f, a, b, tol, n):**" και ο έλεγχος "**if f(a)*f(b) > 0.0:**"
  "**raise ValueError('function has same sign at end points')**" αναλύθηκαν παραπάνω στην υπό-ενότητα **Αλγόριθμος Μέθοδος Διχοτόμησης – Υλοποίηση**.

- Η εντολή "**a_ = [a]**" θέτει στην μεταβλητή "**a_**" την αρχή του διαστήματος "**[a,b]**" , η εντολή "**b_ = [b]**" θέτει στη μεταβλητή "**b_**" το τέλος του διαστήματος "**[a,b]**" και η εντολή "**x = [(a+b)/2]**" θέτει στη μεταβλητή "**x**" το μέσο του διαστήματος "**[a,b]**" για να χρησιμοποιηθούν αργότερα στο πρόγραμμα.
- Στον βρόγχο "**for i in range(n):**"

```
"if f(x[i]) == 0 or (b_[i]-a_[i])/2 < tol:"
    "print('Bisection method has converged')"
    "break"

"if f(a_[i])*f(x[i]) > 0:"
    "a_.append(x[i])"
    "b_.append(b_[i])"
"else:"
    "a_.append(a_[i])"
    "b_.append(x[i])"

"x.append((a_[i+1]+b_[i+1])/2)"
```

γίνεται επανάληψη για τον μέγιστο αριθμό επαναλήψεων "**n**". Σε κάθε επανάληψη ελέγχεται αν η τιμή της συνάρτησης στο "**x[i]**" είναι μηδέν ή αν η διαφορά " **b_[i]-a_[i])/2** " είναι μικρότερη από την ακρίβεια σε δεκαδικά ψηφία. Αν ισχύει είτε το ένα είτε το άλλο τότε εμφανίζει κατάλληλο μήνυμα και σταματάει τις επαναλήψεις. Αν δεν ισχύουν τότε υπολογίζονται οι νέες τιμές για "**a**", "**b**" και "**x**" με βάση το αποτέλεσμα του γινομένου "**f(a_[i])*f(x[i])**". Ελέγχεται αν η συνάρτηση έχει το ίδιο πρόσημο στα άκρα του διαστήματος και αν έχει εκχωρεί την τιμή του "**x[i]**" στο "**a**" και την τιμή "**b_[i]**" στο "**b**", αλλιώς εκχωρεί την τιμή του "**a[i]**" στο "**a**" και την τιμή "**x[i]**" στο "**b**". Τέλος, υπολογίζεται η νέα τιμή του "**x**".

- Στον έλεγχο "**if i == n-1:**"

```
"print('Zero not found to desired tolerance')"
```

αν ο βρόγχος για τις επαναλήψεις στην συνάρτηση " **bisect(f, a, b, tol, n):**" φτάσει στο τέλος και δεν βρεθεί κάποια ρίζα για την επιθυμητή ακρίβεια δεκαδικών, τότε εμφανίζεται κατάλληλο μήνυμα. Αυτό σημαίνει πως η επιθυμητή ακρίβεια δεκαδικών δεν επιτυγχάνεται σε αυτό τον αριθμό επαναλήψεων "**n**".

- Η εντολή "**k = list(range(len(x)))**" δημιουργεί μια λίστα "**k**" η οποία περιέχει τους δείκτες των στοιχείων της "**x**".

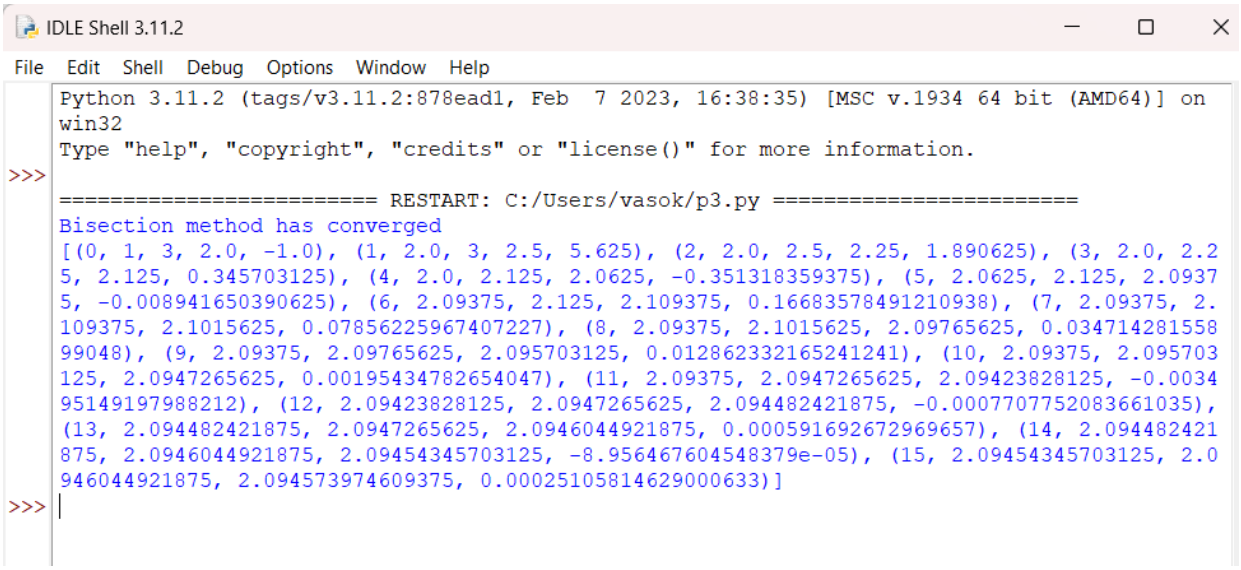
- Στη εντολή "**out = list(zip(k, a_, b_, x, [f(i) for i in x]))**", η "**zip()**" ενώνει την λίστα "**k**" με μία λίστα από τιμές "**a_**", μία λίστα από τιμές "**b_**", μια λίστα από τιμές "**x**" και μία λίστα από τιμές της συνάρτησης "**f()**" που εκτιμώνται στο σημείο κάθε στοιχείου της λίστας "**x**". Η "**list()**" δημιουργεί μια λίστα από την έξοδο της "**zip()**".
- Η εντολή "**return out**" επιστρέφει την μεταβλητή "**out**".
- Η συνάρτηση "**def f(x):**"

  "**return x**3 - 2*x - 5**"

  ορίζει την συνάρτηση "**f(x)**" που θέλουμε να υπολογιστεί.
- Η εντολή "**out = bisect(f, 1, 3, 1/2*10**(-4), 50)**" καλεί την συνάρτηση "**bisect**" για τις τιμές ορισμάτων που επιθυμούμε.
- Η εντολή "**print(out)**" εμφανίζει τον πίνακα "**out**" με τα αποτελέσματα.

Τρόπος Εκτέλεσης Αλγόριθμου Μέθοδος Διχοτόμησης – Παράδειγμα 1

Για να εκτελεστεί ο αλγόριθμος " **Μέθοδος Διχοτόμησης – Παράδειγμα 1**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εκτυπώνεται η λίστα "**out**" όπως φαίνεται παρακάτω.



Εκτέλεση αλγόριθμου Μέθοδος Διχοτόμησης – Παράδειγμα 1

Αλγόριθμος Μέθοδος Newton – Υλοποίηση – Παράδειγμα 1

Ο αλγόριθμος **Μέθοδος Newton – Υλοποίηση – Παράδειγμα 1** υπολογίζει την ρίζα της εξίσωσης $f(x) = x^3 - 2x - 5$ με αρχική τιμή 1, με ακρίβεια 8 δεκαδικών ψηφίων και με μέγιστο αριθμό επαναλήψεων 50.

p3.py - C:/Users/vasok/p3.py (3.11.2)

File Edit Format Run Options Window Help

```
def newton(f, df, x1, tol, n):
    x = [x1]
    i = 3
    while i <= n:
        x.append(x[i-3] - f(x[i-3])/df(x[i-3]))
        if f(x[i-2]) == 0 or abs(x[i-2] - x[i-3]) < tol:
            print('Newton method has converged')
            break
        i += 1
    if i > n:
        k = range(1, n+1)
        print('zero not found to desired tolerance')
    else:
        k = range(1, i)
    out = [[k[j], x[j], f(x[j])] for j in range(len(k))]
    return out
def f(x):
    return x**3 - 2*x - 5

def df(x):
    return 3*x**2 - 2

out = newton(f, df, 1, 1/2*10**(-8), 50)
print(out)
```

Αλγόριθμος Μέθοδος Newton – Υλοποίηση – Παράδειγμα 1

- Η συνάρτηση "**newton(f, df, x1, tol, n):**" εφαρμόζει την μέθοδο newton για την εύρεση της ρίζας μιας συνάρτησης "**f(x)**". Τα ορίσματα της συνάρτησης είναι:
  - "**f**" : μια συνάρτηση "**f**" που αντιστοιχεί στην εξίσωση που θέλουμε να λύσουμε
  - "**df**" : την παράγωγο της "**f**"
  - "**x1**" : την αρχική τιμή

- "**tol**" : η ακρίβεια σε δεκαδικά ψηφία
- "**n**" : ο μέγιστος αριθμός επαναλήψεων

- Η εντολή "**x = [x1]**" δημιουργεί μια λίστα που περιέχει την αρχική τιμή "**x1**".
- Η εντολή "**i = 3**" θα χρησιμοποιηθεί ως μετρητής για να αναφέρει την τρέχουσα επανάληψη του αλγόριθμου.
- Ο βρόγχος "**while i <= n:**"

```
"x.append(x[i-3] - f(x[i-3])/df(x[i-3]))"
"if f(x[i-2]) == 0 or abs(x[i-2] - x[i-3]) < tol:"
    "print('Newton method has converged')"
    "break"
"i += 1"
```

υλοποιεί την μέθοδο Newton. Η επανάληψη ξεκινάει από το "**i = 3**" μέχρι το "**i**" να γίνει μεγαλύτερο από το μέγιστο αριθμό επαναλήψεων "**n**". Στη συνέχεια, υπολογίζεται ο επόμενος όρος της ακολουθίας της μεθόδου Newton, προσθέτετε στο τέλος της λίστας "**x**" και ελέγχεται αν είναι μηδέν ή αν η απόλυτη τιμή του είναι μικρότερη από την ακρίβεια τον δεκαδικών ψηφίων. Αν ισχύει ένα από τα δύο η επανάληψη σταματάει και εμφανίζει κατάλληλο μήνυμα, αλλιώς αυξάνεται ο δείκτης κατά ένα και συνεχίζεται ο έλεγχος. Στο τέλος, η συνάρτηση επιστρέφει το αποτέλεσμα σε μορφή λίστας.

- Στην συνθήκη "**if i > n:**"

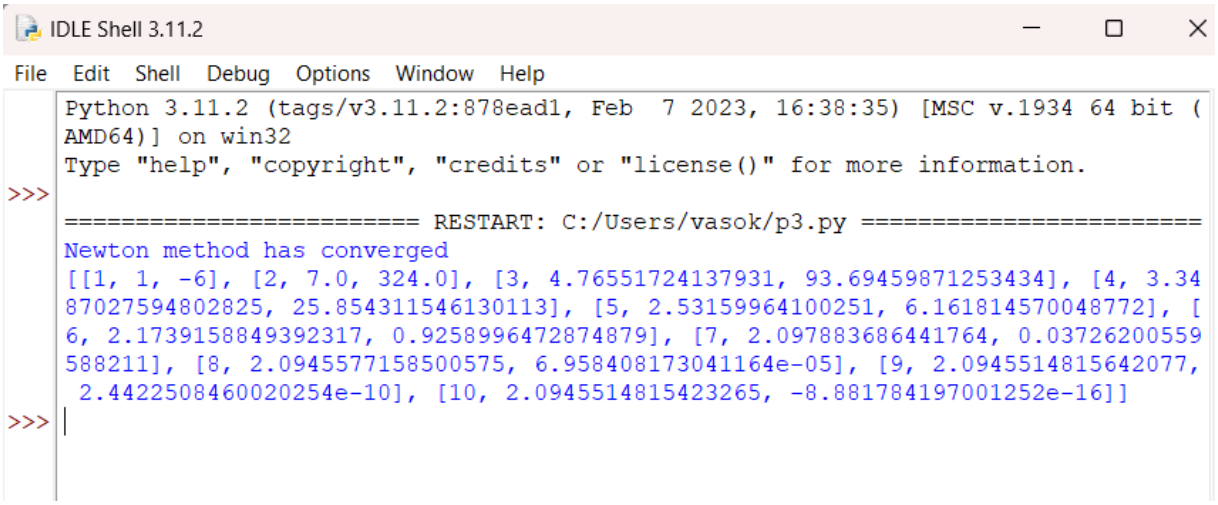
```
"k = range(1, i)"
"print('zero not found to desired tolerance')"
"else:"
    "k = range(1, i)"
"out = [[k[j], x[j], f(x[j])] for j in range(len(k))]"
"return out"
```

γίνεται έλεγχος για το αν το βήμα της επανάληψης "**i**" είναι μεγαλύτερο από τον αριθμό των επαναλήψεων "**n**" και εμφανίζει το ανάλογο μήνυμα πως δεν βρέθηκε ρίζα με αυτή την ακρίβεια δεκαδικών ψηφίων. Αντίθετα, αν το βήμα της επανάληψης "**i**" είναι μικρότερο ή ίσο από τον αριθμό των επαναλήψεων "**n**", τότε η συνάρτηση θέτει τη λίστα "**k**" ίση με την σειρά των αριθμών "**1 έως i-1**". Στη συνέχεια, δημιουργεί μια λίστα "**out**" η οποία περιέχει την θέση "**k[j]** " στην οποία βρέθηκε η ρίζα της συνάρτησης, την τιμή της ρίζας "**x[j]**" και την τιμή της συνάρτησης στο σημείο " **f(x[j])**". Τέλος, η συνάρτηση επιστρέφει την λίστα "**out**".

- Η συνάρτηση "**def f(x):**"
  "**return x**3 - 2*x - 5**"
  ορίζει την συνάρτηση "**f(x)**" που θέλουμε να υπολογιστεί.
- Η συνάρτηση "**def df(x):**"
  "**return 3*x**2 - 2**"
  ορίζει την παράγωγο συνάρτηση "**df(x)**" της "**f(x)**".
- Η εντολή "**out = newton(f, df, 1, 1/2*10**(-8), 50)**" καλεί την συνάρτηση "**newton**" για τις τιμές ορισμάτων που επιθυμούμε.
- Η εντολή "**print(out)**" εμφανίζει τον πίνακα "**out**" με τα αποτελέσματα.

Τρόπος Εκτέλεσης Αλγόριθμου Μέθοδος Newton – Υλοποίηση – Παράδειγμα 1

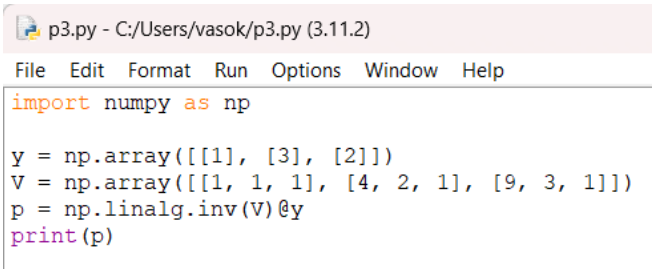
Για να εκτελεστεί ο αλγόριθμος " **Μέθοδος Newton – Υλοποίηση – Παράδειγμα 1**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εκτυπώνεται η λίστα "**out**" όπως φαίνεται παρακάτω.



Εκτέλεση αλγόριθμου Μέθοδος Newton - Υλοποίηση – Παράδειγμα 1

Αλγόριθμος Πολυωνυμική Παρεμβολή – Γενική Μέθοδος - Παράδειγμα

Ο αλγόριθμος **Πολυωνυμική Παρεμβολή – Γενική Μέθοδος – Παράδειγμα** βρίσκει τους συντελεστές του πολυωνύμου το οποίο διέρχεται από τα σημεία Α(1,1), Β(2,3) και Γ(3,2).

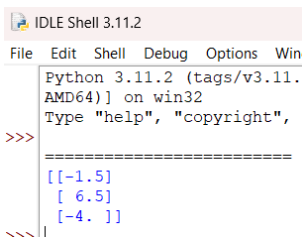


Αλγόριθμος Πολυωνυμική Παρεμβολή – Γενική Μέθοδος – Παράδειγμα

- Η εντολή "**import numpy as np**" εισάγει την βιβλιοθήκη Numpy για να μπορέσουμε να υλοποιήσουμε τις μαθηματικές πράξεις που χρειάζεται ο αλγόριθμος.
- Η εντολή "**y = np.array([[1], [3], [2]])**" εισάγει το διάνυσμα στήλη των τεταγμένων των σημείων στην μεταβλητή "**y**".
- Η εντολή "**V = np.array([[1, 1, 1], [4, 2, 1], [9, 3, 1]])**" εισάγει τον πίνακα Vandermonde στην μεταβλητή "**V**".
- Η εντολή "**p = np.linalg.inv(V)@y**" αντιστρέφει τον πίνακα "**V**" με την συνάρτηση " **np.linalg.inv(V)**" και στην συνέχεια πολλαπλασιάζει τον αντίστροφο πίνακα "**V**" με το διάνυσμα "**y**" χρησιμοποιώντας το σύμβολο "**@**". Στην μεταβλητή "**p**" βρίσκονται οι συντελεστές του πολυωνύμου.

Τρόπος Εκτέλεσης Αλγόριθμου Πολυωνυμική Παρεμβολή – Γενική Μέθοδος – Παράδειγμα

Για να εκτελεστεί ο αλγόριθμος " **Πολυωνυμική Παρεμβολή – Γενική Μέθοδος - Παράδειγμα** " πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εκτυπώνεται ο πίνακας "**p**" όπως φαίνεται παρακάτω.



Εκτέλεση αλγορίθμου Πολυωνυμική Παρεμβολή – Γενική Μέθοδος – Παράδειγμα

Αλγόριθμος Συνάρτηση interp1 – Παράδειγμα (Αλγόριθμοι Γραμμικών Παρεμβολών, Κυβικών Παρεμβολών και με Κυβικά Splines)

Ο αλγόριθμος **Συνάρτηση interp1 – Παράδειγμα** βρίσκει τις τιμές του "**y**" για "**x=1.5**" με γραμμική παρεμβολή και για "**x=2.9**" με γραμμική παρεμβολή, με κυβική παρεμβολή και με παρεμβολή με κυβικά splines για τα σημεία Α(1,1), Β(2,3) και Γ(3,2).

```
import numpy as np
x=np.array([1,2,3])
y=np.array([1,3,2])
xi=1.5
yi=np.interp(xi,x,y)
print(yi)
```

Αλγόριθμος Συνάρτηση interp1 – Παράδειγμα για x=1.5 με γραμμική παρεμβολή

- Η εντολή "**import numpy as np**" εισάγει την βιβλιοθήκη Numpy για να μπορέσουμε να υλοποιήσουμε τις μαθηματικές πράξεις που χρειάζεται ο αλγόριθμος.
- Η εντολή "**x=np.array([1,2,3])**" εισάγει το διάνυσμα των τετμημένων των σημείων στην μεταβλητή "**x**".
- Η εντολή "**y=np.array([1,3,2])**" εισάγει το διάνυσμα των τεταγμένων των σημείων στην μεταβλητή "**y**".
- Η εντολή "**xi=1.5**" ορίζει στην μεταβλητή "**xi**" την τιμή 1.5 της παρεμβαλόμενης τιμής.
- Η εντολή "**yi=np.interp(xi,x,y)**" υπολογίζει την γραμμική παρεμβολή των "**x**" και "**y**" για την τιμή "**xi**" και την εκχωρεί στην μεταβλητή "**yi**".
- Η εντολή "**print(yi)**" εμφανίζει την τιμή του "**yi**".

```
import numpy as np
x=np.array([1,2,3])
y=np.array([1,3,2])
xi=2.9
yi=np.interp(xi,x,y)
print(yi)
```

Αλγόριθμος Συνάρτηση interp1 – Παράδειγμα για x=2.9 με γραμμική παρεμβολή

- Η εντολή "**import numpy as np**", η "**x=np.array([1,2,3])**", η "**y=np.array([1,3,2])**", η "**yi=np.interp(xi,x,y)**" και η "**print(yi)**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Συνάρτηση interp1 – Παράδειγμα για χ=1.5 με γραμμική παρεμβολή**.
- Η εντολή "**xi=2.9**" ορίζει στην μεταβλητή "**xi**" την τιμή 2.9 της παρεμβαλόμενης τιμής.

```
import numpy as np
x=np.array([1,2,3])
y=np.array([1,3,2])
f=np.polyfit(x,y,3)
p=np.poly1d(f)
xi=2.9
yi=p(xi)
print(yi)
```

Αλγόριθμος Συνάρτηση interp1 – Παράδειγμα για χ=2.9 με κυβική παρεμβολή

- Η εντολή "**import numpy as np**", η "**x=np.array([1,2,3])**", η "**y=np.array([1,3,2])**", η "**xi=2.9**" και η "**print(yi)**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Συνάρτηση interp1 – Παράδειγμα για χ=1.5 με γραμμική παρεμβολή**.
- Η εντολή "**f=np.polyfit(x,y,3)**" επιστρέφει τους συντελεστές ενός πολυωνύμου τρίτου βαθμού για τα "**x**" και "**y**".
- Η εντολή "**p=np.poly1d(f)**" δημιουργεί ένα πολυώνυμο τρίτου βαθμού με βάση τους συντελεστές που βρέθηκαν προηγουμένος.
- Η εντολή "**yi=p(xi)**" υπολογίζει την τιμή "**yi**" του πολυωνύμου για "**xi**".

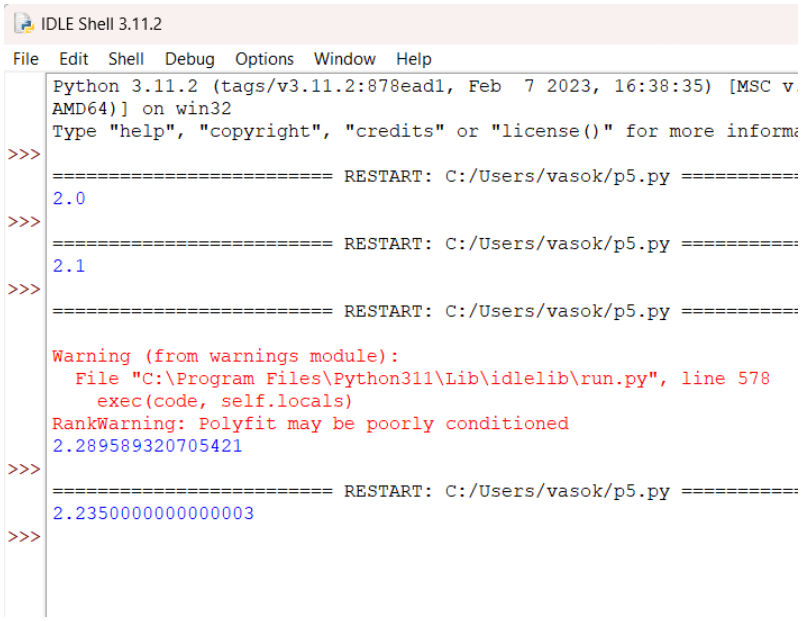
```
import numpy as np
from scipy.interpolate import CubicSpline
x=np.array([1,2,3])
y=np.array([1,3,2])
f=CubicSpline(x,y)
y=f(2.9)
print(y)
```

Αλγόριθμος Συνάρτηση interp1 – Παράδειγμα για χ=2.9 με παρεμβολή με κυβικά splines

- Η εντολή "**import numpy as np**", η "**x=np.array([1,2,3])**" και η "**y=np.array([1,3,2])**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Συνάρτηση interp1 – Παράδειγμα για χ=1.5 με γραμμική παρεμβολή**.
- Η εντολή "**from scipy.interpolate import CubicSpline**" εισάγει την κλάση "**CubicSpline**" από την βιβλιοθήκη "**scipy.interpolate**" για να μπορέσουμε να υλοποιήσουμε την παρεμβολή με κυβικά splines.
- Η εντολή "**f=CubicSpline(x,y)**" δημιουργεί την παρεμβολή με κυβικά splines και την εκχωρεί στην μεταβλητή "**f**".
- Η εντολή "**y=f(2.9)**" υπολογίζει την τιμή "**y**" της συνάρτησης στο σημείο 2.9 με την χρήση της "**CubicSpline** " της "**f**".
- Η εντολή "**print(y)**" εμφανίζει την τιμή του "**y**" για "**f(2.9)**".

Τρόπος Εκτέλεσης Αλγόριθμων Συνάρτηση interp1 – Παράδειγμα (Αλγόριθμοι Γραμμικών Παρεμβολών, Κυβικών Παρεμβολών και με Κυβικά Splines)

Για να εκτελεστούν οι αλγόριθμοι "**Γραμμικών Παρεμβολών, Κυβικών Παρεμβολών και με Κυβικά Splines**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζονται τα αποτελέσματα.



Εκτέλεση αλγορίθμου Συνάρτηση interp1 – Παράδειγμα για χ=1.5 με γραμμική παρεμβολή

Εκτέλεση αλγορίθμου Συνάρτηση interp1 – Παράδειγμα για χ=2.9 με γραμμική παρεμβολή

Εκτέλεση αλγορίθμου Συνάρτηση interp1 – Παράδειγμα για χ=2.9 με κυβική παρεμβολή

Εκτέλεση αλγορίθμου Συνάρτηση interp1 – Παράδειγμα για χ=2.9 με παρεμβολή με κυβικά splines

Αλγόριθμος Συνάρτηση polyfit – ΠΑΡΑΔΕΙΓΜΑ

Ο αλγόριθμος **Συνάρτηση polyfit – ΠΑΡΑΔΕΙΓΜΑ** βρίσκει το πολυώνυμο παρεμβολής που διέρχεται από τα σημεία Α(1,1), Β(2,3) και Γ(3,2).

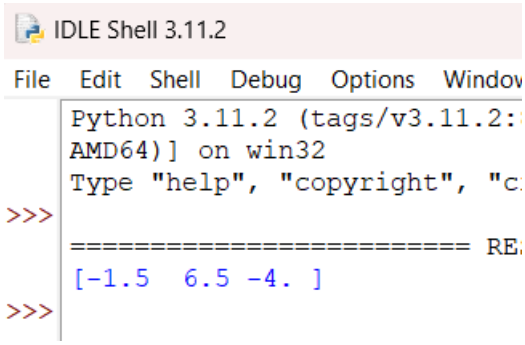
```
import numpy as np
x=np.array([1,2,3])
y=np.array([1,3,2])
f=np.polyfit(x,y,2)
print(f)
```

Αλγόριθμος Συνάρτηση polyfit – ΠΑΡΑΔΕΙΓΜΑ

- Η εντολή "**import numpy as np**", η "**x=np.array([1,2,3])**" και η "**y=np.array([1,3,2])**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Συνάρτηση interp1 – Παράδειγμα για χ=1.5 με γραμμική παρεμβολή**.
- Η εντολή "**f=np.polyfit(x,y,2)**" επιστρέφει τους συντελεστές ενός πολυωνύμου δεύτερου βαθμού για τα "**x**" και "**y**".
- Η εντολή "**print(f)**" εμφανίζει τους συντελεστές του πολυωνύμου.

Τρόπος Εκτέλεσης Αλγόριθμου Συνάρτηση polyfit – ΠΑΡΑΔΕΙΓΜΑ

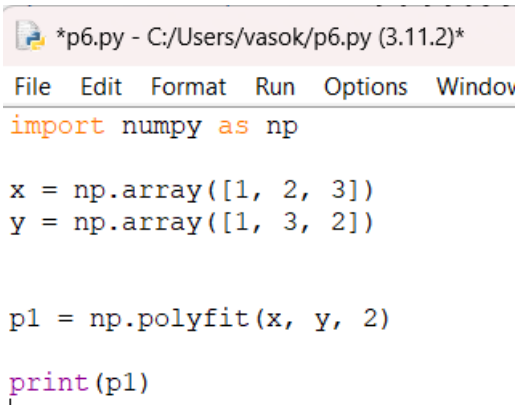
Για να εκτελεστεί ο αλγόριθμος "**Συνάρτηση polyfit – ΠΑΡΑΔΕΙΓΜΑ**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εκτυπώνονται οι συντελεστές του πολυωνύμου όπως φαίνεται παρακάτω.



Εκτέλεση αλγορίθμου Συνάρτηση polyfit – ΠΑΡΑΔΕΙΓΜΑ

Αλγόριθμος Παράδειγμα (polyfit – polyval – polyder – polyval(polyder,2))

Ο αλγόριθμος **Παράδειγμα polyfit** βρίσκει το πολυώνυμο παρεμβολής που διέρχεται από τα σημεία Α(1,1), Β(2,3) και Γ(3,2).



Αλγόριθμος Παράδειγμα polyfit

- Η εντολή "**import numpy as np**", η "**x=np.array([1,2,3])**" και η "**y=np.array([1,3,2])**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Συνάρτηση interp1 – Παράδειγμα για x=1.5 με γραμμική παρεμβολή**.
- Η εντολή "**p1=np.polyfit(x,y,2)**" επιστρέφει τους συντελεστές ενός πολυωνύμου δεύτερου βαθμού για τα "**x**" και "**y**".
- Η εντολή "**print(p1)**" εμφανίζει τους συντελεστές του πολυωνύμου.

Ο αλγόριθμος **Παράδειγμα polyval** βρίσκει την τιμή του πολυωνύμου "**p**" για την τιμή 1.7 που διέρχεται από τα σημεία Α(1,1), Β(2,3) και Γ(3,2).

```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([1, 3, 2])

p1 = np.polyfit(x, y, 2)
val=np.polyval(p1,1.7)

print(val)
```

Αλγόριθμος Παράδειγμα polyval

- Η εντολή "**import numpy as np**", η "**x=np.array([1,2,3])**", "**y=np.array([1,3,2])**" και η "**p1=np.polyfit(x,y,2)**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**val=np.polyval(p1,1.7)**" υπολογίζει την τιμή του πολυωνύμου για "**x=1.7**".
- Η εντολή "**print(val)**" εμφανίζει την τιμή του πολυωνύμου για "**x=1.7**".

Ο αλγόριθμος **Παράδειγμα polyder** υπολογίζει την παράγωγο συνάρτησης του πολυωνύμου "**p**" που διέρχεται από τα σημεία Α(1,1), Β(2,3) και Γ(3,2).

```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([1, 3, 2])

p1 = np.polyfit(x, y, 2)
val=np.polyder(p1)

print(val)
```

Αλγόριθμος Παράδειγμα polyder

- Η εντολή "**import numpy as np**", η "**x=np.array([1,2,3])**", "**y=np.array([1,3,2])**" και η "**p1=np.polyfit(x,y,2)**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**val=np.polyder(p1)**" υπολογίζει την παράγωγο συνάρτησης του πολυωνύμου "**p1**".
- Η εντολή "**print(val)**" εμφανίζει την τιμή της παραγώγου συνάρτησης του πολυωνύμου "**p1**".

Ο αλγόριθμος **Παράδειγμα polyval(polyder,2)** βρίσκει την τιμή του πολυωνύμου "**polyder**" που είναι η παράγωγος της "**p**" για την τιμή 2 που διέρχεται από τα σημεία Α(1,1), Β(2,3) και Γ(3,2).

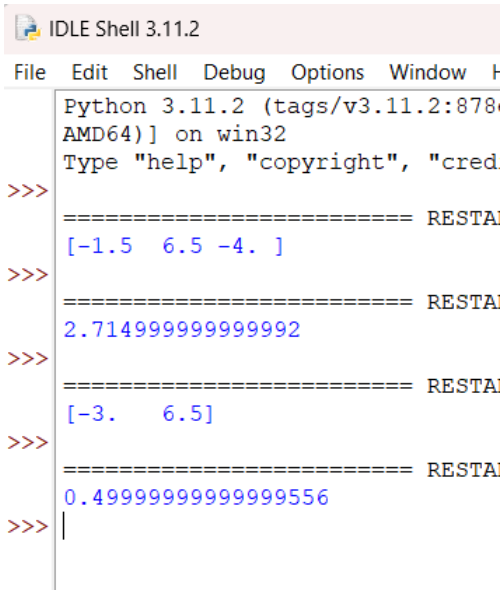
```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([1, 3, 2])

p1 = np.polyfit(x, y, 2)
val=np.polyder(p1)
g=np.polyval(val,2)

print(g)
```

Αλγόριθμος Παράδειγμα polyval(polyder,2)

- Η εντολή "**import numpy as np**", η "**x=np.array([1,2,3])**", "**y=np.array([1,3,2])**", η "**p1=np.polyfit(x,y,2)**" και η "**val=np.polyder(p1)**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit** και **Αλγόριθμο Παράδειγμα polyder**.
- Η εντολή "**g=np.polyval(val,2)**" υπολογίζει την τιμή του πολυωνύμου για "**x=2**".
- Η εντολή "**print(g)**" εμφανίζει την τιμή του πολυωνύμου για "**x=2**".

Τρόπος Εκτέλεσης Αλγόριθμος Παράδειγμα (polyfit – polyval – polyder - polyval(polyder,2))

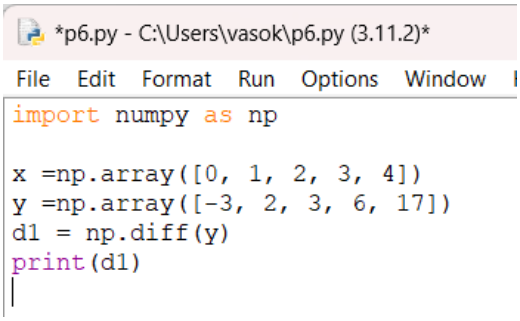
Για να εκτελεστούν οι αλγόριθμοι " **Παράδειγμα (polyfit – polyval – polyder - polyval(polyder,2))**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζονται τα αποτελέσματα



Εκτέλεση αλγορίθμων Παράδειγμα (polyfit – polyval – polyder - polyval(polyder,2))

Αλγόριθμος Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))

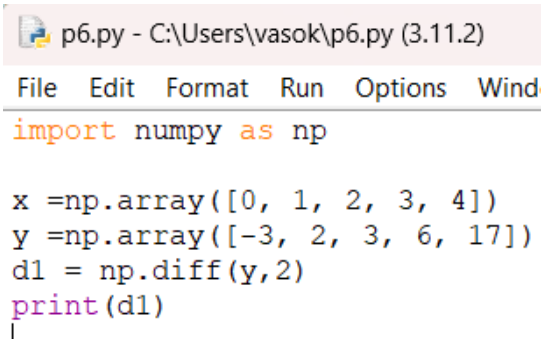
Ο αλγόριθμος **Παράδειγμα diff(y)** υπολογίζει τις διαφορές των στοιχείων του "**y**".



Αλγόριθμος Παράδειγμα diff(y)

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**d1 = np.diff(y)**" υπολογίζει τις διαφορές μεταξύ διαδοχικών στοιχείων του πίνακα "**y**" και επιστρέφει ένα νέο πίνακα που περιέχει τις διαφορές αυτές.
- Η εντολή "**print(d1)**" εμφανίζει τον πίνακα των διαφορών "**d1**".

Ο αλγόριθμος **Παράδειγμα diff(y,2)** υπολογίζει την διαφορά δεύτερης τάξης του πίνακα "**y**".



Αλγόριθμος Παράδειγμα diff(y,2)

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**d1 = np.diff(y,2)**" υπολογίζει την διαφορά δεύτερης τάξης μεταξύ των στοιχείων του πίνακα "**y**" και επιστρέφει ένα νέο πίνακα.
- Η εντολή "**print(d1)**" εμφανίζει τον πίνακα των διαφορών "**d1**".

Ο αλγόριθμος **Παράδειγμα diff(y,3)** υπολογίζει την διαφορά τρίτης τάξης του πίνακα "**y**".

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Windo

```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])
d1 = np.diff(y,3)
print(d1)
```

Αλγόριθμος Παράδειγμα diff(y,3)

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**d1 = np.diff(y,3)**" υπολογίζει την διαφορά τρίτης τάξης μεταξύ των στοιχείων του πίνακα "**y**" και επιστρέφει ένα νέο πίνακα.
- Η εντολή "**print(d1)**" εμφανίζει τον πίνακα των διαφορών "**d1**".

Ο αλγόριθμος **Παράδειγμα diff(y,4)** υπολογίζει την διαφορά τέταρτης τάξης του πίνακα "**y**".

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window

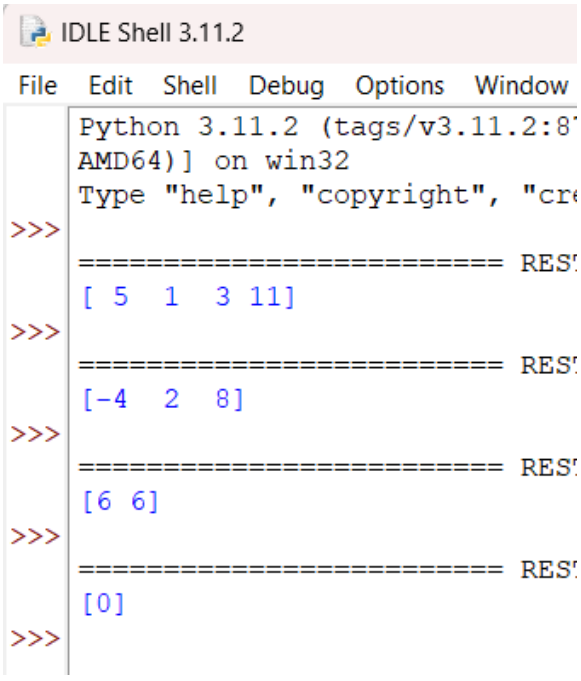
```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])
d1 = np.diff(y,4)
print(d1)
```

Αλγόριθμος Παράδειγμα diff(y,4)

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**d1 = np.diff(y,4)**" υπολογίζει την διαφορά τέταρτης τάξης μεταξύ των στοιχείων του πίνακα "**y**" και επιστρέφει ένα νέο πίνακα.
- Η εντολή "**print(d1)**" εμφανίζει τον πίνακα των διαφορών "**d1**".

Τρόπος Εκτέλεσης Αλγόριθμος Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))

Για να εκτελεστούν οι αλγόριθμοι " **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζονται τα αποτελέσματα



Εκτέλεση Αλγορίθμων Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 0$

Ο αλγόριθμος **Πρώτη Παράγωγος - Παράδειγμα για $\chi_0 = 0$** υπολογίζει την τιμής της παραγώγου για το πολυώνυμο "**Δ-ΝG**" για "$\chi_0 = 0$" που έχει μέχρι 4 διαφορές.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

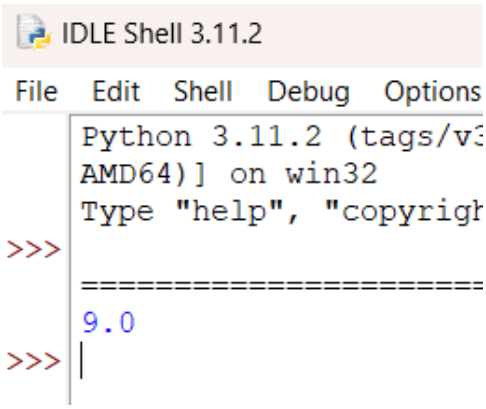
```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])

y0 = y[:5]
d1 = np.diff(y0)
d2 = np.diff(y0,2)
d3 = np.diff(y0,3)
d4 = np.diff(y0,4)

res = 1/1 * (d1[0] - 1/2 * (d2[0]) + 1/3 * (d3[0]) - 1/4 * (d4[0]))
print(res)
```

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 0$

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**y0 = y[:5]**" αντιγράφει τα πέντε πρώτα στοιχεία του πίνακα "**y**" στον πίνακα "**y0**".
- Οι εντολές "**d1 = np.diff(y0)**", "**d2 = np.diff(y0,2)**", "**d3 = np.diff(y0,3)**" και η "**d4 = np.diff(y0,4)**" αναλύθηκαν παραπάνω στους αλγορίθμους **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**.
- Η εντολή "**res = 1/1 * (d1[0] - 1/2 * (d2[0]) + 1/3 * (d3[0]) - 1/4 * (d4[0]))**" υπολογίζει την τιμή της παραγώγου για $\chi_0 = 0$ και την εκχωρεί στην μεταβλητή "**res**".
- Η εντολή "**print(res)**" εμφανίζει την τιμή της παραγώγου για $\chi_0 = 0$.

Τρόπος Εκτέλεσης Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 0$

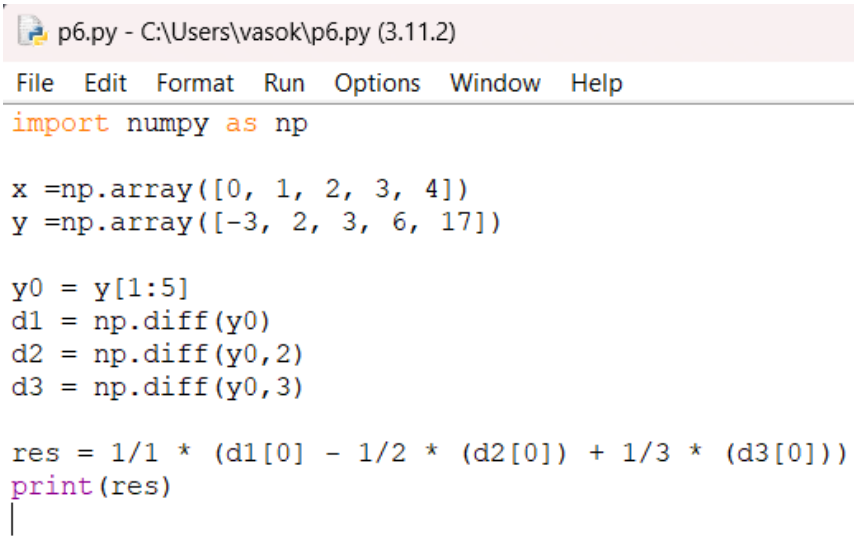
Για να εκτελεστεί ο αλγόριθμος " **Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 0$**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.



Εκτέλεση αλγόριθμου Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 0$

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 1$

Ο αλγόριθμος **Πρώτη Παράγωγος - Παράδειγμα για $\chi_0 = 1$** υπολογίζει την τιμής της παραγώγου για το πολυώνυμο "**Δ-ΝΘ**" για "$\chi_0 = 1$" που έχει μέχρι 3 διαφορές.

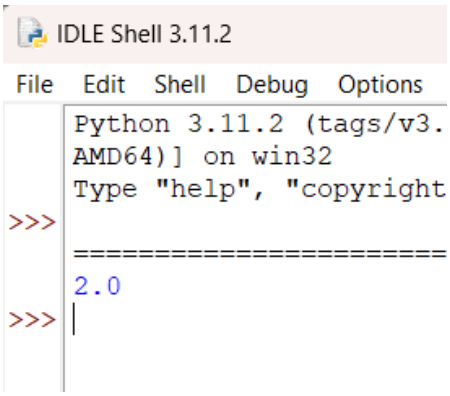


Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 1$

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**y0 = y[1:5]**" ξεκινάει να αντιγράφει από το στοιχείο της θέσης 1 μέχρι την θέση 4 του πίνακα "**y**" στον πίνακα "**y0**".
- Οι εντολές "**d1 = np.diff(y0)**", "**d2 = np.diff(y0,2)**" και η "**d3 = np.diff(y0,3)**" αναλύθηκαν παραπάνω στους αλγορίθμους **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**.
- Η εντολή "**res = 1/1 * (d1[0] - 1/2 * (d2[0]) + 1/3 * (d3[0]))**" υπολογίζει την τιμή της παραγώγου για $\chi_0 = 1$ και την εκχωρεί στην μεταβλητή "**res**".
- Η εντολή "**print(res)**" εμφανίζει την τιμή της παραγώγου για $\chi_0 = 1$.

Τρόπος Εκτέλεσης Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 1$

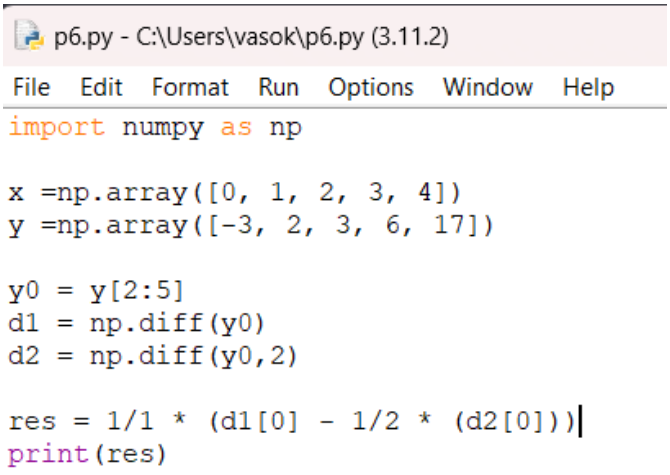
Για να εκτελεστεί ο αλγόριθμος " **Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 1$**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.



Εκτέλεση αλγόριθμου Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 1$

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 2$

Ο αλγόριθμος **Πρώτη Παράγωγος - Παράδειγμα για $\chi_0 = 2$** υπολογίζει την τιμής της παραγώγου για το πολυώνυμο "**Δ-ΝG**" για "$\chi_0 = 2$" που έχει μέχρι 2 διαφορές.

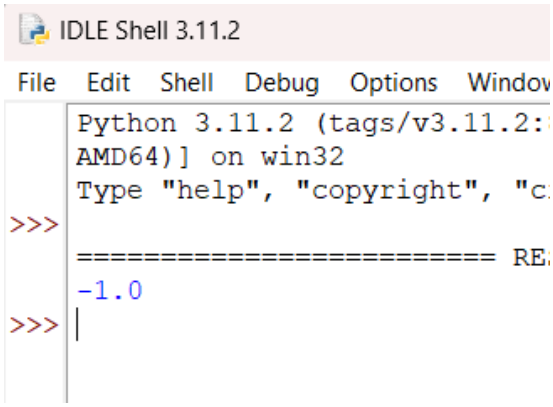


Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 2$

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**y0 = y[2:5]**" ξεκινάει να αντιγράφει από το στοιχείο της θέσης 2 μέχρι την θέση 4 του πίνακα "**y**" στον πίνακα "**y0**".
- Οι εντολές "**d1 = np.diff(y0)**" και η "**d2 = np.diff(y0,2)**" αναλύθηκαν παραπάνω στους αλγορίθμους **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**.
- Η εντολή "**res = 1/1 * (d1[0] - 1/2 * (d2[0]))**" υπολογίζει την τιμή της παραγώγου για $\chi_0 = 2$ και την εκχωρεί στην μεταβλητή "**res**".
- Η εντολή "**print(res)**" εμφανίζει την τιμή της παραγώγου για $\chi_0 = 2$.

Τρόπος Εκτέλεσης Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 2$

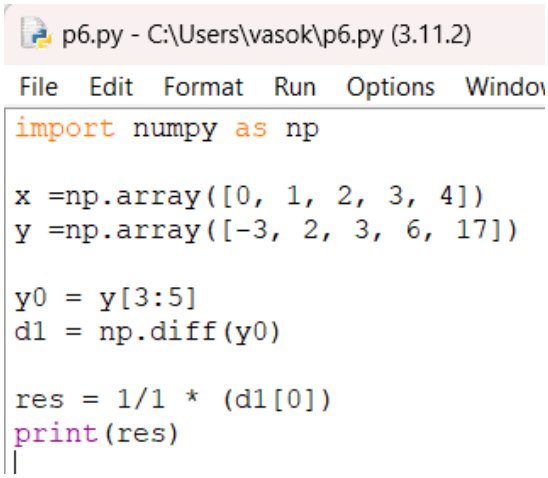
Για να εκτελεστεί ο αλγόριθμος " **Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 2$**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.



Εκτέλεση αλγόριθμου Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 2$

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 3$

Ο αλγόριθμος **Πρώτη Παράγωγος - Παράδειγμα για $\chi_0 = 3$** υπολογίζει την τιμής της παραγώγου για το πολυώνυμο "**Δ-ΝΘ**" για "$\chi_0 = 3$" που έχει μέχρι 1 διαφορές.

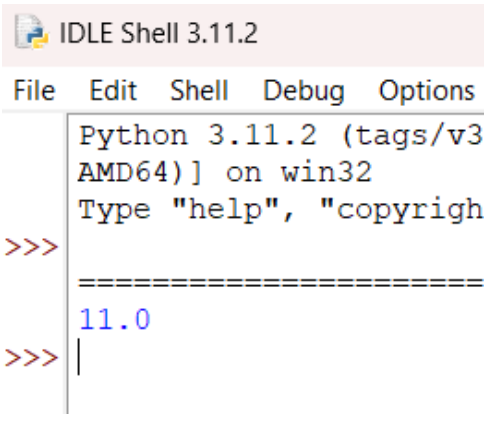


Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 3$

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**y0 = y[3:5]**" ξεκινάει να αντιγράφει από το στοιχείο της θέσης 3 μέχρι την θέση 4 του πίνακα "**y**" στον πίνακα "**y0**".
- Η εντολή "**d1 = np.diff(y0)**" αναλύθηκε παραπάνω στους αλγορίθμους **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**.
- Η εντολή " **res = 1/1 * (d1[0])**" υπολογίζει την τιμή της παραγώγου για $\chi_0 = 3$ και την εκχωρεί στην μεταβλητή "**res**".
- Η εντολή "**print(res)**" εμφανίζει την τιμή της παραγώγου για $\chi_0 = 3$.

Τρόπος Εκτέλεσης Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 3$

Για να εκτελεστεί ο αλγόριθμος " **Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 3$**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.



Εκτέλεση αλγόριθμου Πρώτη Παράγωγος – Παράδειγμα για $\chi_0 = 3$

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 1$

Ο αλγόριθμος **Πρώτη Παράγωγος - Παράδειγμα για $\chi_n = 1$** υπολογίζει την τιμής της παραγώγου για το πολυώνυμο "**∇-NG**" για "$\chi_n = 1$" που έχει μέχρι 1 διαφορές.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window

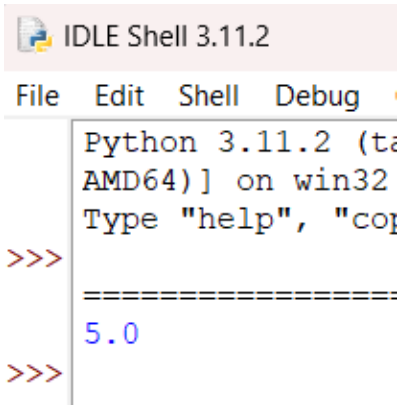
```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])

y0 = y[0:2]
d1 = np.diff(y0)

res = 1/1 * (d1[-1])
print(res)
```

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 1$

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**y0 = y[0:2]**" ξεκινάει να αντιγράφει από το στοιχείο της θέσης 0 μέχρι την θέση 1 του πίνακα "**y**" στον πίνακα "**y0**".
- Η εντολή "**d1 = np.diff(y0)**" αναλύθηκε παραπάνω στους αλγορίθμους **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**.
- Η εντολή "**res = 1/1 * (d1[-1])**" υπολογίζει την τιμή της παραγώγου και την εκχωρεί στην μεταβλητή "**res**".
- Η εντολή "**print(res)**" εμφανίζει την τιμή της παραγώγου.

Τρόπος Εκτέλεσης Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 1$

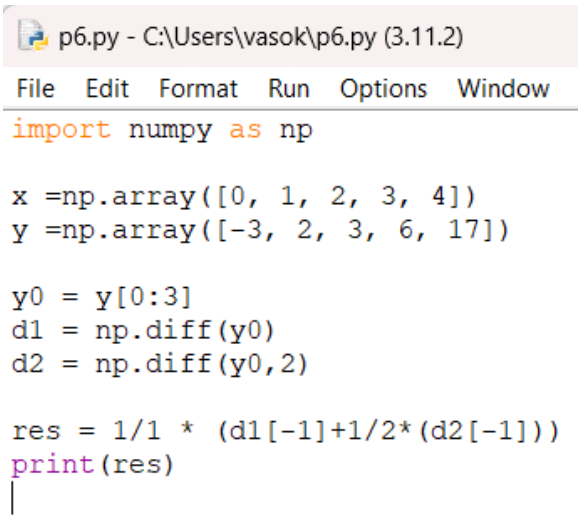
Για να εκτελεστεί ο αλγόριθμος " **Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 1$**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.



Εκτέλεση αλγόριθμου Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 1$

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 2$

Ο αλγόριθμος **Πρώτη Παράγωγος - Παράδειγμα για $\chi_n = 2$** υπολογίζει την τιμής της παραγώγου για το πολυώνυμο "**∇-NG**" για "$\chi_n = 2$" που έχει μέχρι 2 διαφορές.

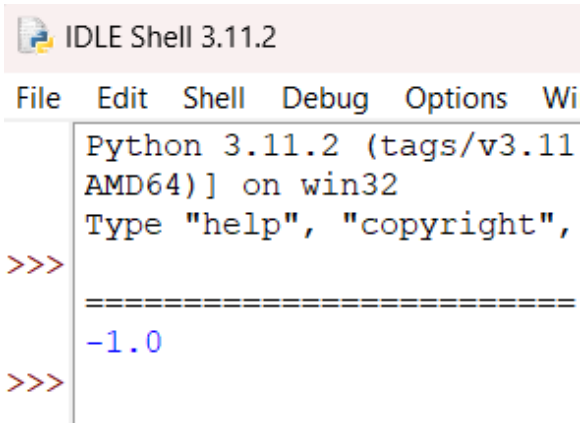


Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 2$

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**y0 = y[0:3]**" ξεκινάει να αντιγράφει από το στοιχείο της θέσης 0 μέχρι την θέση 2 του πίνακα "**y**" στον πίνακα "**y0**".
- Οι εντολές "**d1 = np.diff(y0)**" και η "**d2 = np.diff(y0,2)**" αναλύθηκαν παραπάνω στους αλγορίθμους **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**.
- Η εντολή " **res = 1/1 * (d1[-1]+1/2 * (d2[-1]))**" υπολογίζει την τιμή της παραγώγου και την εκχωρεί στην μεταβλητή "**res**".
- Η εντολή "**print(res)**" εμφανίζει την τιμή της παραγώγου.

Τρόπος Εκτέλεσης Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 2$

Για να εκτελεστεί ο αλγόριθμος " **Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 2$**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.



Εκτέλεση αλγόριθμου Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 2$

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 3$

Ο αλγόριθμος **Πρώτη Παράγωγος - Παράδειγμα για $\chi_n = 3$** υπολογίζει την τιμής της παραγώγου για το πολυώνυμο "**∇-NG**" για "$\chi_n = 3$" που έχει μέχρι 3 διαφορές.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

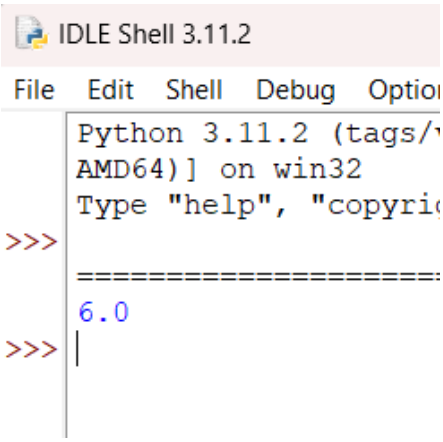
```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])

y0 = y[0:4]
d1 = np.diff(y0)
d2 = np.diff(y0,2)
d3 = np.diff(y0,3)

res = 1/1 * (d1[-1]+1/2*(d2[-1])+1/3*(d3[-1]))
print(res)
```

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 3$

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και "**y=np.array([])**", αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**y0 = y[0:4]**" ξεκινάει να αντιγράφει από το στοιχείο της θέσης 0 μέχρι την θέση 3 του πίνακα "**y**" στον πίνακα "**y0**".
- Οι εντολές "**d1 = np.diff(y0)**", "**d2 = np.diff(y0,2)**" και η "**d3 = np.diff(y0,3)**" αναλύθηκαν παραπάνω στους αλγορίθμους **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**.
- Η εντολή " **res = 1/1 * (d1[-1]+1/2*(d2[-1])+1/3*(d3[-1]))**" υπολογίζει την τιμή της παραγώγου και την εκχωρεί στην μεταβλητή "**res**".
- Η εντολή "**print(res)**" εμφανίζει την τιμή της παραγώγου.

Τρόπος Εκτέλεσης Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 3$

Για να εκτελεστεί ο αλγόριθμος " **Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 3$**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.



Εκτέλεση αλγόριθμου Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 3$

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 4$

Ο αλγόριθμος **Πρώτη Παράγωγος - Παράδειγμα για $\chi_n = 4$** υπολογίζει την τιμής της παραγώγου για το πολυώνυμο "**∇-NG**" για "$\chi_n = 4$" που έχει μέχρι 4 διαφορές.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

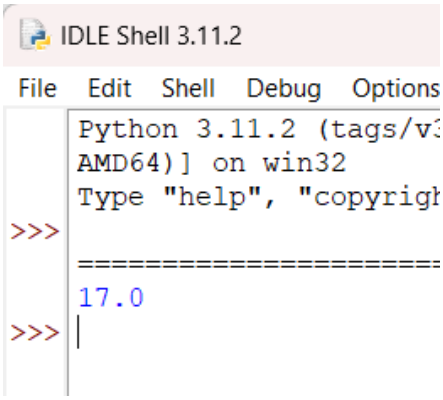
```
import numpy as np

x =np.array([0, 1, 2, 3, 4])
y =np.array([-3, 2, 3, 6, 17])

y0 = y[0:5]
d1 = np.diff(y0)
d2 = np.diff(y0,2)
d3 = np.diff(y0,3)
d4 = np.diff(y0,4)

res = 1/1 * ((d1[-1]+1/2*(d2[-1])+1/3*(d3[-1])))+1/4*(d4[-1])
print(res)
```

Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 4$

- Η εντολή "**import numpy as np**", η "**x=np.array([])**" και η "**y=np.array([])**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**y0 = y[0:5]**" ξεκινάει να αντιγράφει από το στοιχείο της θέσης 0 μέχρι την θέση 4 του πίνακα "**y**" στον πίνακα "**y0**".
- Οι εντολές "**d1 = np.diff(y0)**", "**d2 = np.diff(y0,2)**", "**d3 = np.diff(y0,3)**" και η "**d4 = np.diff(y0,4)**" αναλύθηκαν παραπάνω στους αλγορίθμους **Παράδειγμα (diff(y) – diff(y,2) – diff(y,3) - diff(y,4))**.
- Η εντολή " **res = 1/1 * ((d1[-1]+1/2*(d2[-1])+1/3*(d3[-1])))+1/4*(d4[-1])**" υπολογίζει την τιμή της παραγώγου και την εκχωρεί στην μεταβλητή "**res**".
- Η εντολή "**print(res)**" εμφανίζει την τιμή της παραγώγου.

Τρόπος Εκτέλεσης Αλγόριθμος Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 4$

Για να εκτελεστεί ο αλγόριθμος " **Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 4$**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.



Εκτέλεση αλγόριθμου Πρώτη Παράγωγος – Παράδειγμα για $\chi_n = 4$

Αλγόριθμος Κανόνας Τραπεζίου - Υλοποίηση

Ο αλγόριθμος **Κανόνας Τραπεζίου – Υλοποίηση** υπολογίζει την τιμή του ορισμένου ολοκληρώματος $I = \int_a^b f(x)dx$ με τον κανόνα Τραπεζίου.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Hel

```
import numpy as np
def trapeziou(f, a, b, n):
    h = (b - a) / n
    S = f(a)
    for i in range(1, n):
        x = a + h*i
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/2
    return I
```

Αλγόριθμος Κανόνας Τραπεζίου – Υλοποίηση

- Η εντολή "**import numpy as np**" αναλύθηκε στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η συνάρτηση "**def trapeziou(f, a, b, n):**" υλοποιεί την μέθοδο τραπεζίου για τον υπολογισμό της προσέγγισης ενός ολοκληρώματος. Δέχεται ως ορίσματα:
  - Το "**f**" : που είναι μία συνάρτηση "f".
  - Το "**a**" : είναι το κάτω όριο του ολοκληρώματος.
  - Το "**b**" : είναι το πάνω όριο του ολοκληρώματος.
  - Το "**n**" : είναι το πλήθος των υποδιαστημάτων που θα εφαρμοστεί ο τύπος.
- Η εντολή "**h = (b - a) / n**" υπολογίζει το βήμα των ισαπεχόντων σημείων.
- Η εντολή "**S = f(a)**" αρχικοποιεί την μεταβλητή "**S**" με την τιμή της συνάρτησης στο κάτω όριο του διαστήματος.
- Σε αυτό το κομμάτι κώδικα "**for i in range(1, n):**"
  "**x = a + h*i**"
  "**S =S+2f(x)**"
  "**S =S+ f(b)**"
  "**I = h*S/2**"
  "**return I**"

επιλέγεται ένα σημείο "**x**" μέσα στο εύρος της ολοκλήρωσης και υπολογίζεται η τιμή της συνάρτησης "**f**" στο σημείο αυτό. Στη συνέχεια, πολλαπλασιάζουμε την τιμή αυτή με το "**2**" και την προσθέτουμε στο σύνολο των τραπεζίων που έχουν υπολογιστεί μέχρι στιγμής. Επαναλαμβάνετε αυτή η διαδικασία για τα υπόλοιπα σημεία "**i**" στο εύρος της ολοκλήρωσης, από το δεύτερο έως το προτελευταίο. Τέλος, προσθέτουμε την τιμή της συνάρτησης "**f**" στο τελευταίο σημείο "**b**", πολλαπλασιάζουμε με το "**1**" και το προσθέτουμε στο σύνολο των τραπεζίων. Το αποτέλεσμα αυτού του αθροίσματος πολλαπλασιάζεται με το "**h/2**", και έτσι υπολογίζουμε την τελική τιμή του ολοκληρώματος. Αυτή η τιμή επιστρέφεται στην κλήση της συνάρτησης.

Αλγόριθμος Κανόνας Τραπεζίου – Παράδειγμα 1

Ο αλγόριθμος **Κανόνας Τραπεζίου – Παράδειγμα 1** υπολογίζει την τιμή του ορισμένου ολοκληρώματος $I = \int_0^4 (x \cdot \sin(2x) + 4)dx$ για n = 1, n = 2, n = 4, n = 100, n = 1000.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

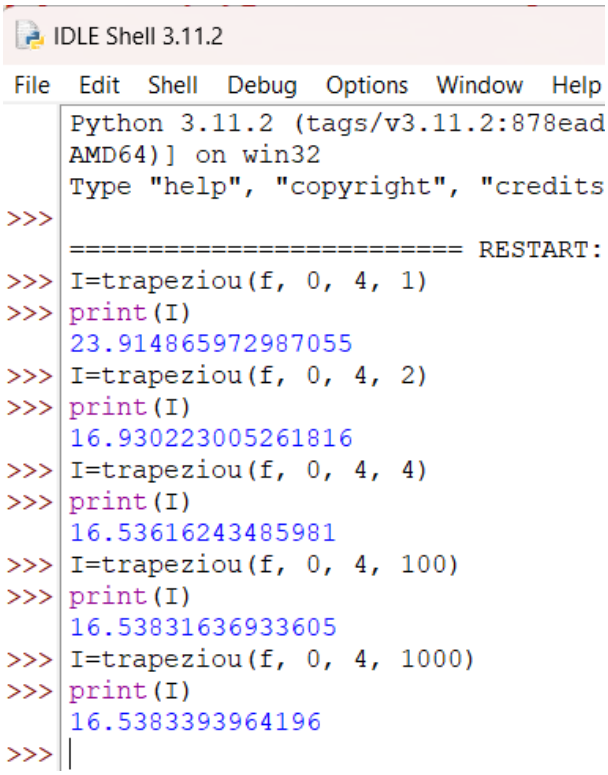
```
import numpy as np
def trapeziou(f, a, b, n):
    h = (b - a) / n
    S = f(a)
    for i in range(1, n):
        x = a + h*i
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/2
    return I

def f(x):
    return x*np.sin(2*x)+4
```

Αλγόριθμος Κανόνας Τραπεζίου – Παράδειγμα 1

- Οι εντολές από το "**import numpy as np**" έως και "**return I**" αναλύθηκαν στον παραπάνω **Αλγόριθμο** Κανόνας Τραπεζίου – Υλοποίηση.
- Η συνάρτηση "**def f(x):**"
  "**return x*np.sin(2*x)+4**"
  ορίζει την συνάρτηση "**f(x)**".

Τρόπος Εκτέλεσης Αλγόριθμος Κανόνας Τραπεζίου – Παράδειγμα 1

Για να εκτελεστεί ο αλγόριθμος "**Κανόνας Τραπεζίου – Παράδειγμα 1**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση αλγόριθμου Κανόνας Τραπεζίου – Παράδειγμα 1

- Η εντολή "**I=trapeziou(f, 0, 4, 1)**" επιστρέφει την τιμή του ορισμένου ολοκληρώματος για τα ορίσματα "**f**", "**0**", "**4**" και "**1**" που δίνουμε. Ομοίως και για τις υπόλοιπες εντολές.

Αλγόριθμος Κανόνας Τραπεζίου – Παράδειγμα 2

Ο αλγόριθμος **Κανόνας Τραπεζίου – Παράδειγμα 2** υπολογίζει την τιμή του ορισμένου ολοκληρώματος $\boldsymbol{I = \int_0^4 (x \cdot \sin(2x) + 4) dx}$ για n = 500, για n = 499 και την ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window

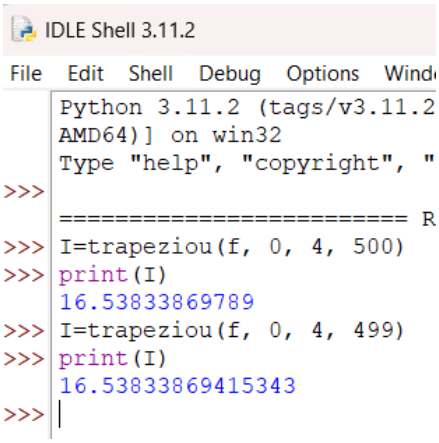
```
import numpy as np
def trapeziou(f, a, b, n):
    h = (b - a) / n
    S = f(a)
    for i in range(1, n):
        x = a + h*i
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/2
    return I

def f(x):
    return x*np.sin(2*x)+4
```

Αλγόριθμος Κανόνας Τραπεζίου – Παράδειγμα 2

- Οι εντολές από το "**import numpy as np**" έως και "**return I**" αναλύθηκαν στον παραπάνω **Αλγόριθμο Κανόνας Τραπεζίου – Υλοποίηση**.
- Η συνάρτηση "**def f(x):**"
  "**return x*np.sin(2*x)+4**"
  ορίζει την συνάρτηση "**f(x)**".

Τρόπος Εκτέλεσης Αλγόριθμος Κανόνας Τραπεζίου – Παράδειγμα 2

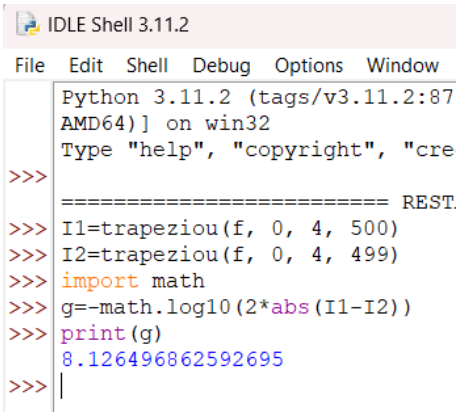
Για να εκτελεστεί ο αλγόριθμος "**Κανόνας Τραπεζίου – Παράδειγμα 2**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση αλγόριθμου Κανόνας Τραπεζίου – Παράδειγμα 2

- Η εντολή "**I=trapeziou(f, 0, 4, 500)**" επιστρέφει την τιμή του ορισμένου ολοκληρώματος για τα ορίσματα "**f**", "**0**", "**4**" και "**500**" που δίνουμε. Ομοίως και για την "**I=trapeziou(f, 0, 4, 499)**".

Για να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία εκτελούνται οι παρακάτω εντολές στο IDLE Shell.



Ακρίβεια αποτελέσματος αλγόριθμου Κανόνας Τραπεζίου – Παράδειγμα 2

- Η εντολή "**I=trapeziou(f, 0, 4, 500)**" επιστρέφει την τιμή του ορισμένου ολοκληρώματος για τα ορίσματα "**f**", "**0**", "**4**" και "**500**" που δίνουμε. Ομοίως και για την "**I=trapeziou(f, 0, 4, 499)**".
- Η εντολή "**import numpy as np**" αναλύθηκε στον παραπάνω **Αλγόριθμο Κανόνας Τραπεζίου – Υλοποίηση**.
- Η εντολή "**g=-math.log10(2*abs(I1-I2))**" υπολογίζει την ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.
- Η εντολή "**print(g)**" εκτυπώνει την ακρίβεια των δεκαδικών ψηφίων.

Αλγόριθμος Κανόνας Simpson - Υλοποίηση

Ο αλγόριθμος **Κανόνας Simpson – Υλοποίηση** υπολογίζει την τιμή του ορισμένου ολοκληρώματος $I = \int_a^b f(x)dx$ με τον κανόνα Simpson.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Wind

```
import numpy as np
import math
def simpson(f, a, b, n):
    h = (b - a) / (2 * n)
    S = f(a)
    for i in range(1, n+1):
        x = a + h*(2*i-1)
        S =S+4*f(x)
    for i in range(1, n):
        x = a + h*(2*i)
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/3
    return I
```

Αλγόριθμος Κανόνας Simpson – Υλοποίηση

- Η εντολή "**import numpy as np**" αναλύθηκε στον παραπάνω **Αλγόριθμο Παράδειγμα polyfit**.
- Η εντολή "**import math**" εισάγει την βιβλιοθήκη μαθηματικών της Python, η οποία περιλαμβάνει συναρτήσεις για την επίλυση μαθηματικών προβλημάτων, όπως τον υπολογισμό της ρίζας ενός αριθμού, την εκθετική συνάρτηση κτλ.
- Η συνάρτηση "**def simpson(f, a, b, n):**" υλοποιεί την μέθοδο Simpson για τον υπολογισμό της προσέγγισης ενός ολοκληρώματος. Δέχεται ως ορίσματα:
  - Το "**f**" : που είναι μία συνάρτηση "f".
  - Το "**a**" : είναι το κάτω όριο του ολοκληρώματος.
  - Το "**b**" : είναι το πάνω όριο του ολοκληρώματος.
  - Το "**n**" : είναι το πλήθος των υποδιαστημάτων που θα εφαρμοστεί ο τύπος.
- Η εντολή " **h = (b - a) / (2 * n)** " υπολογίζει το βήμα των ισαπεχόντων σημείων.

- Η εντολή "**S = f(a)**" αρχικοποιεί την μεταβλητή "**S**" με την τιμή της συνάρτησης στο κάτω όριο του διαστήματος.
- Σε αυτό το κομμάτι κώδικα "**for i in range(1, n+1):**"
  "**x = a + h*(2*i-1)**"
  "**S =S+4*f(x)**"
  υπολογίζει τα αθροίσματα των τιμών της συνάρτησης στις περιττές θέσεις του διαστήματος [a,b] και τα πολλαπλασιάζει με το 4. Το αποτέλεσμα αυτού του υπολογισμού αποθηκεύεται στη μεταβλητή "**S**".
- Σε αυτό το κομμάτι κώδικα "**for i in range(1, n):**"
  "**x = a + h*(2*i)**"
  "**S =S+2*f(x)**"
  υπολογίζει τα αθροίσματα των τιμών της συνάρτησης στις άρτιες θέσεις του διαστήματος [a,b] και τα πολλαπλασιάζει με το 2. Το αποτέλεσμα αυτού του υπολογισμού αποθηκεύεται στη μεταβλητή "**S**".
- Η εντολή "**S =S+ f(b)**" υπολογίζει το άθροισμα της συνάρτησης "**f**" στο άκρο "**b**" του διαστήματος "**[a,b]**" και προσθέτει την τιμή αυτή στον αριθμό "**S**".
- Η εντολή "**I = h*S/3**" υπολογίζει το τελικό αποτέλεσμα του ολοκληρώματος.
- Η εντολή "**return I**" εκτυπώνει το αποτέλεσμα.

Αλγόριθμος Κανόνας Simpson – Παράδειγμα 1

Ο αλγόριθμος **Κανόνας Simpson – Παράδειγμα 1** υπολογίζει την τιμή του ορισμένου ολοκληρώματος $I = \int_0^4 (x \cdot \sin(2x) + 4)dx$ με τον κανόνα Simpson για n=1, n=2, n=4, n=100, n=1000.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Wind

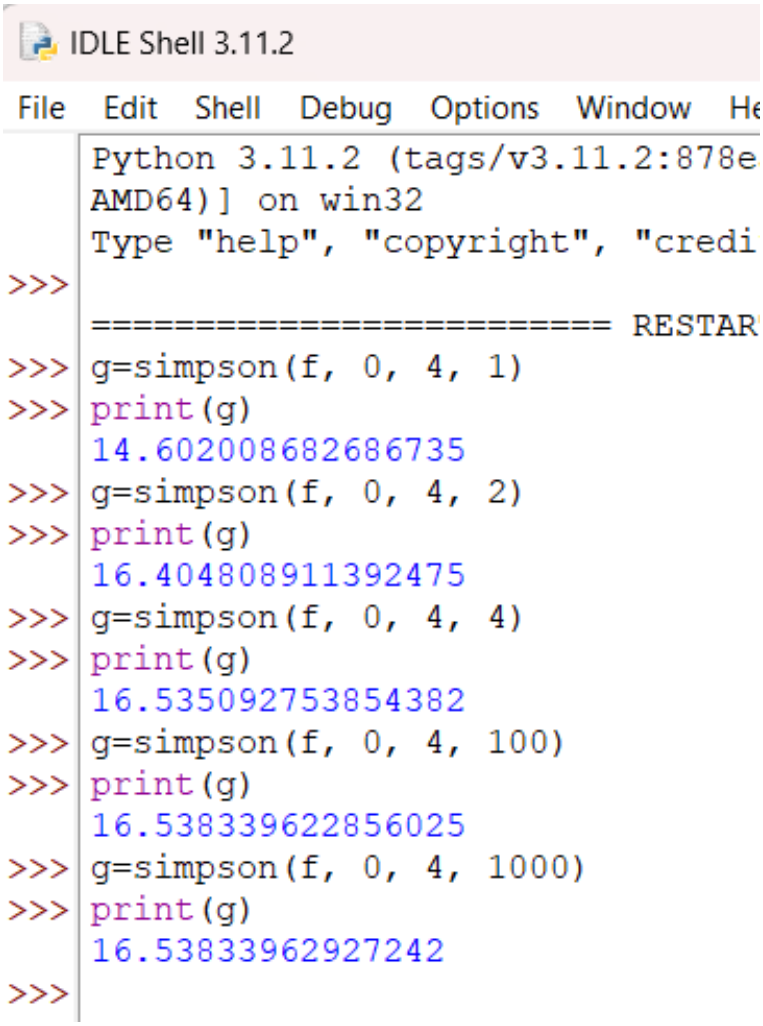
```
import numpy as np
import math
def simpson(f, a, b, n):
    h = (b - a) / (2 * n)
    S = f(a)
    for i in range(1, n+1):
        x = a + h*(2*i-1)
        S =S+4*f(x)
    for i in range(1, n):
        x = a + h*(2*i)
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/3
    return I
```

Αλγόριθμος Κανόνας Simpson – Παράδειγμα 1

- Ο κώδικας **Αλγόριθμος Κανόνας Simpson – Παράδειγμα 1** αναλύθηκε στον παραπάνω **Αλγόριθμο Κανόνας Simpson - Υλοποίηση**.

Τρόπος Εκτέλεσης Αλγόριθμος Κανόνας Simpson – Παράδειγμα 1

Για να εκτελεστεί ο αλγόριθμος "**Κανόνας Simpson – Παράδειγμα 1**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση αλγόριθμου Κανόνας Simpson – Παράδειγμα 1

- Η εντολή "**g=simpson(f, 0, 4, 1)**" επιστρέφει την τιμή του ορισμένου ολοκληρώματος για τα ορίσματα "**f**", "**0**", "**4**" και "**1**" που δίνουμε. Ομοίως και για τις υπόλοιπες εντολές.

Αλγόριθμος Κανόνας Simpson – Παράδειγμα 2

Ο αλγόριθμος **Κανόνας Simpson – Παράδειγμα 2** υπολογίζει την τιμή του ορισμένου ολοκληρώματος $I = \int_0^4 (x \cdot \sin(2x) + 4)dx$ με τον κανόνα Simpson για n=500 και την ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Wind

```
import numpy as np
import math
def simpson(f, a, b, n):
    h = (b - a) / (2 * n)
    S = f(a)
    for i in range(1, n+1):
        x = a + h*(2*i-1)
        S =S+4*f(x)
    for i in range(1, n):
        x = a + h*(2*i)
        S =S+2*f(x)
    S =S+ f(b)
    I = h*S/3
    return I
```

Αλγόριθμος Κανόνας Simpson – Παράδειγμα 2

- Ο κώδικας **Αλγόριθμος Κανόνας Simpson – Παράδειγμα 2** αναλύθηκε στον παραπάνω **Αλγόριθμο Κανόνας Simpson - Υλοποίηση**.

Τρόπος Εκτέλεσης Αλγόριθμος Κανόνας Simpson – Παράδειγμα 2

Για να εκτελεστεί ο αλγόριθμος "**Κανόνας Simpson – Παράδειγμα 2**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.

IDLE Shell 3.11.2

File Edit Shell Debug Options Window

```
Python 3.11.2 (tags/v3.11.2:87
AMD64)] on win32
Type "help", "copyright", "cre
>>>
========================= REST
>>> I1=simpson(f, 0, 4, 500)
>>> I2=simpson(f, 0, 4, 499)
>>> print(I1)
16.538339629262797
>>> print(I2)
16.53833962926269
>>>
```

Εκτέλεση αλγόριθμου Κανόνας Simpson – Παράδειγμα 2

- Η εντολή "**g=simpson(f, 0, 4, 500)**" επιστρέφει την τιμή του ορισμένου ολοκληρώματος για τα ορίσματα "**f**", "**0**", "**4**" και "**500**" που δίνουμε. Ομοίως και για την "**g=simpson(f, 0, 4, 499)**".

Για να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία εκτελούνται οι παρακάτω εντολές στο IDLE Shell.

```
>>> import math
>>> g=-math.log10(2*abs(I1-I2))
>>> print(g)
12.671288541487455
>>>
```

Ακρίβεια αποτελέσματος αλγόριθμου Κανόνας Simpson – Παράδειγμα 2

- Η εντολή "**import math**" αναλύθηκε στον παραπάνω **Αλγόριθμο Κανόνας Simpson – Υλοποίηση**.
- Η εντολή "**g=-math.log10(2*abs(I1-I2))**" υπολογίζει την ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.
- Η εντολή "**print(g)**" εκτυπώνει την ακρίβεια των δεκαδικών ψηφίων.

Αλγόριθμος Κανόνας Ορθογωνίου – Παράδειγμα 1

Ο αλγόριθμος **Κανόνας Ορθογωνίου – Παράδειγμα 1** υπολογίζει την τιμή του ορισμένου ολοκληρώματος $I = \int_0^4 (x \cdot \sin(2x) + 4)dx$ με τον κανόνα ορθογωνίου για n=4 και n=200.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

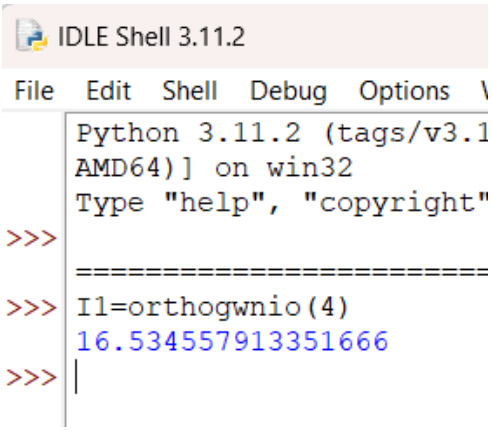
```
from math import sin
def orthogwnio(n):
    a = 0
    b = 4
    h = (b - a) / n
    x = [a + h / 2 + i * h for i in range(n)]
    I499 = h * sum(f(xi) for xi in x)
    print(I499)
def f(x):
    return x * sin(2 * x) + 4
```

Αλγόριθμος Κανόνας Ορθογωνίου – Παράδειγμα 1

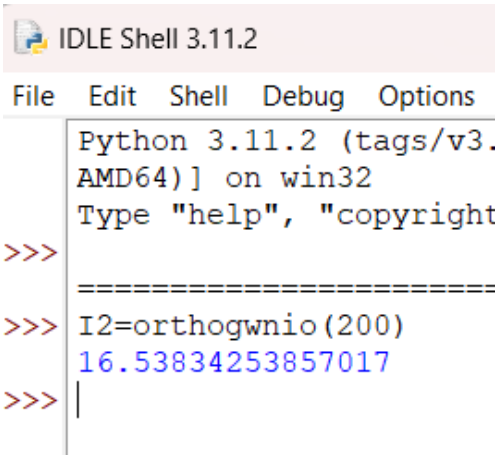
- Η εντολή "**from math import sin**" εισάγει την συνάρτηση "**sin**" από την βιβλιοθήκη "**math**", η οποία επιστρέφει τον αριθμό "**sin(x)**".
- Η εντολή "**def orthogwnio(n):**" υπολογίζει την τιμή του ορισμένου ολοκληρώματος με τον κανόνα του ορθογωνίου. Δέχεται ως όρισμα τον ακέραιο "**n**", δηλαδή το πλήθος των υποδιαστημάτων.
- Η εντολή "**a = 0**" και η "**b = 4**" εκχωρούν στις μεταβλητές "**a**" και "**b**" το κάτω και άνω όριο του ολοκληρώματος αντίστοιχα.
- Η εντολή "**h = (b - a) / n**" εκχωρεί στην μεταβλητή "**h**" το βήμα των ισαπεχόντων σημείων.
- Η εντολή "**x = [a + h / 2 + i * h for i in range(n)]**" εκχωρεί στην λίστα "**x**" τα σημεία στα οποία πρέπει να υπολογιστούν οι τιμές της συνάρτησης για τον υπολογισμό του ολοκληρώματος.
- Η εντολή "**I499 = h * sum(f(xi) for xi in x)**" υπολογίζει την τιμή του ορισμένου ολοκληρώματος.
- Η εντολή "**print(I499)**" εκτυπώνει την τιμή του ολοκληρώματος που υπολογίστηκε.

Τρόπος Εκτέλεσης Αλγόριθμος Κανόνας Ορθογωνίου – Παράδειγμα 1

Για να εκτελεστεί ο αλγόριθμος "**Κανόνας Ορθογωνίου – Παράδειγμα 1**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.



Εκτέλεση αλγόριθμου Κανόνας Ορθογωνίου – Παράδειγμα 1 (n=4)



Εκτέλεση αλγόριθμου Κανόνας Ορθογωνίου – Παράδειγμα 1 (n=200)

- Η εντολή "**I1=orthogwnio(4)**" επιστρέφει την τιμή του ορισμένου ολοκληρώματος για "**n=4**". Ομοίως και για την ""**I2=orthogwnio(200)**".

Αλγόριθμος Κανόνας Ορθογωνίου – Παράδειγμα 2

Ο αλγόριθμος **Κανόνας Ορθογωνίου – Παράδειγμα 2** υπολογίζει την τιμή του ορισμένου ολοκληρώματος $\boldsymbol{I = \int_0^4 (x \cdot \sin(2x) + 4) dx}$ με τον κανόνα ορθογωνίου για n=500 και την ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

```
from math import sin
def orthogwnio(n):
    a = 0
    b = 4
    h = (b - a) / n
    x = [a + h / 2 + i * h for i in range(n)]
    I499 = h * sum(f(xi) for xi in x)
    print(I499)
def f(x):
    return x * sin(2 * x) + 4
```

Αλγόριθμος Κανόνας Ορθογωνίου – Παράδειγμα 2

- Ο κώδικας **Αλγόριθμος Κανόνας Ορθογωνίου – Παράδειγμα 2** αναλύθηκε στον παραπάνω **Αλγόριθμο Κανόνας Ορθογωνίου – Παράδειγμα 1**.

Τρόπος Εκτέλεσης Αλγόριθμος Κανόνας Ορθογωνίου – Παράδειγμα 2

Για να εκτελεστεί ο αλγόριθμος "**Κανόνας Ορθογωνίου – Παράδειγμα 2**" πρέπει να εκτελεστούν οι παρακάτω εντολές στο IDLE Shell της Python.

IDLE Shell 3.11.2

File Edit Shell Debug Optic

```
Python 3.11.2 (tags/
AMD64)] on win32
Type "help", "copyri
>>>
==========
>>> I1=orthogwnio(500)
16.53834009494922
```

Εκτέλεση αλγόριθμου Κανόνας Ορθογωνίου – Παράδειγμα 2 (n=500)

- Η εντολή "**I1=orthogwnio(500)**" επιστρέφει την τιμή του ορισμένου ολοκληρώματος για "**n=500**".

Για να βρεθεί η ακρίβεια του αποτελέσματος σε δεκαδικά ψηφία εκτελείται ο παρακάτω αλγόριθμος.

p6.py - C:\Users\vasok\p6.py (3.11.2)

File Edit Format Run Options Window Help

```
from math import sin
import math

def f(x):
    return x * sin(2 * x) + 4

a = 0
b = 4
n = 500
n1=499
h = (b - a) / n
x = [a + h / 2 + i * h for i in range(n)]
I500 = h * sum(f(xi) for xi in x)
x = [a + h / 2 + i * h for i in range(n1)]
I499 = h * sum(f(xi) for xi in x)
I1=print(I500)
I2=print(I499)
g=-math.log10(2*abs(I500-I499))
print(g)
```

Αλγόριθμος ακρίβεια αποτελέσματος Κανόνας Ορθογωνίου – Παράδειγμα 2

Για να εκτελεστεί ο αλγόριθμος "**Ακρίβεια Αποτελέσματος Κανόνας Ορθογωνίου – Παράδειγμα 2**" πρέπει από την γραμμή εργαλείων του Editor να πατήσουμε το Run και στο IDLE Shell μας εμφανίζεται τα αποτέλεσμα.

```
======================
16.53834009494922
16.474676092241776
0.8950760640644286
>>>
```

Εκτέλεση αλγόριθμου ακρίβεια αποτελέσματος Κανόνας Ορθογωνίου – Παράδειγμα 2

Σύγκριση Matlab με Python

Εισαγωγή Matlab – Python

Οι γλώσσες προγραμματισμού Matlab και Python είναι δύο από τις πιο δημοφιλείς γλώσσες προγραμματισμού στον κόσμο σήμερα. Και οι δύο χρησιμοποιούνται στους επιστημονικούς υπολογισμούς, την ανάλυση δεδομένων και τη μηχανική μάθηση. Ενώ και οι δύο γλώσσες είναι ικανές να επιτελέσουν πολλές από τις ίδιες εργασίες, υπάρχουν ορισμένες διαφορές μεταξύ τους που μπορεί να κάνουν την μία καταλληλότερη για μια συγκεκριμένη εργασία από την άλλη.

Matlab

Το Matlab είναι γλώσσα αριθμητικών υπολογισμών που αναπτύχθηκε από την MathWorks. Είναι γνωστή για τις ισχυρές πράξεις πινάκων, τις ενσωματωμένες συναρτήσεις και την διαισθητική σύνταξη. Επίσης, είναι κατάλληλο για αριθμητική ανάλυση, επεξεργασία σήματος και επεξεργασία εικόνας. Αξίζει να αναφερθεί πως χρησιμοποιείται συχνά σε ακαδημαϊκά και ερευνητικά περιβάλλοντα.

Πλεονεκτήματα Matlab

Το συντακτικό του matlab είναι εύκολο στην εκμάθηση αλλά και στην χρήση, γεγονός που το καθιστά δημοφιλή επιλογή για αρχάριους. Διαθέτει μια μεγάλη βιβλιοθήκη ενσωματωμένων συναρτήσεων και εργαλειοθηκών που διευκολύνουν την εκτέλεση πολύπλοκων μαθηματικών πράξεων. Ωστόσο, έχει σχεδιαστεί ειδικά για αριθμητικούς υπολογισμούς, πράγμα που σημαίνει ότι είναι βελτιστοποιημένο για ταχύτητα και αποτελεσματικότητα κατά την εργασία με μεγάλα σύνολα δεδομένων. Διαθέτει ισχυρό γραφικό περιβάλλον που διευκολύνει την οπτικοποίηση δεδομένων αλλά και τη δημιουργία

γραφικών παραστάσεων και διαγραμμάτων. Σημαντικό να αναφερθεί πως διαθέτει μια μεγάλη και ενεργή κοινότητα χρηστών, πράγμα που σημαίνει ότι υπάρχουν πολλοί διαθέσιμοι πόροι για εκμάθηση και αντιμετώπιση προβλημάτων.

Μειονεκτήματα Matlab

Το Matlab είναι μια ιδιόκτητη γλώσσα, που σημαίνει ότι απαιτείται άδεια για να μπορέσουμε να την χρησιμοποιήσουμε. Το κόστος της άδειας χρήσης του μπορεί να είναι απαγορευτικό, ειδικά για μεμονωμένους χρήστες ή μικρές επιχειρήσεις. Έχει παρατηρηθεί πως η γλώσσα Matlab δεν είναι τόσο ευέλικτη όσο άλλες γλώσσες προγραμματισμού όπως είναι η Python, για αυτό είναι λιγότερο κατάλληλη για εργασίες που περιλαμβάνουν επεξεργασία κειμένου, ανάπτυξη ιστοσελίδων ή άλλες υπολογιστικές εργασίες γενικού σκοπού.

Python

Η Python είναι μια γλώσσα προγραμματισμού ανοικτού κώδικα, γενικού σκοπού, η οποία χρησιμοποιείται ευρέως στους επιστημονικούς υπολογισμούς, στην ανάλυση δεδομένων και στη μηχανική μάθηση. Αξίζει να σημειωθεί ότι είναι γνωστή για την ευελιξία της και χρησιμοποιείται συχνά σε ένα ευρύ φάσμα βιομηχανιών και εφαρμογών.

Πλεονεκτήματα Python

Η Python είναι μια γλώσσα ανοικτού κώδικα, δηλαδή η χρήση και η διανομή της είναι ελεύθερη. Έχει μια μεγάλη και ενεργή κοινότητα χρηστών, πράγμα που σημαίνει ότι υπάρχουν πολλοί διαθέσιμοι πόροι για εκμάθηση και αντιμετώπιση προβλημάτων. Ακόμη, η Python είναι μια γλώσσα γενικού σκοπού, πράγμα που σημαίνει ότι μπορεί να χρησιμοποιηθεί για ένα ευρύ φάσμα εργασιών πέραν των επιστημονικών υπολογισμών και της ανάλυσης δεδομένων. Αξίζει να τονισθεί πως διαθέτει μια μεγάλη βιβλιοθήκη από ενότητες και πακέτα τρίτων κατασκευαστών που διευκολύνουν την εκτέλεση

σύνθετων εργασιών όπως η επεξεργασία φυσικής γλώσσας και η μηχανική μάθηση. Σπουδαίο είναι το γεγονός πως η Python είναι ιδιαίτερα προσαρμόσιμη και μπορεί να ενσωματωθεί με άλλες γλώσσες και εργαλεία.

Μειονεκτήματα Python

Η σύνταξη της Python μπορεί να είναι πιο δύσκολη στην εκμάθηση από αυτή του Matlab, ειδικά για αρχάριους. Επιπλέον, δεν είναι τόσο γρήγορη ή αποδοτική όσο η Matlab για ορισμένες εργασίες αριθμητικού υπολογισμού, αν και η χρήση βιβλιοθηκών όπως η NumPy μπορεί να το μετριάσει αυτό σε κάποιο βαθμό. Επίσης σημαντικό μειονέκτημα είναι πως οι γραφικές δυνατότητες της Python δεν είναι τόσο ισχυρές όσο της Matlab, ωστόσο βιβλιοθήκες όπως η Matplotlib μπορούν να παρέχουν παρόμοια λειτουργικότητα. Διαπιστώνεται πως η Python δεν είναι τόσο κατάλληλη για ορισμένες εξειδικευμένες εφαρμογές, όπως είναι για παράδειγμα τα συστήματα ελέγχου ή η επεξεργασία εικόνας, όσο η Matlab.

Συμπεράσματα

Συνοψίζοντας, το Matlab και η Python είναι και οι δύο ισχυρές γλώσσες προγραμματισμού με τα δικά τους δυνατά και αδύνατα σημεία. Από τη μία η Matlab μπορεί να είναι πιο κατάλληλη για εργασίες αριθμητικής ανάλυσης και επεξεργασίας σήματος, ενώ από την άλλη η Python μπορεί να είναι πιο ευέλικτη και να είναι πιο κατάλληλη για ένα ευρύτερο φάσμα εργασιών. Το εύκολο συντακτικό, οι ενσωματωμένες συναρτήσεις και οι ισχυρές γραφικές δυνατότητες του Matlab το καθιστούν εξαιρετική επιλογή για αριθμητική ανάλυση και επεξεργασία σήματος, ενώ η ευελιξία, η πολυχρηστικότητα και η εκτεταμένη βιβλιοθήκη ενοτήτων της Python το καθιστούν καλή επιλογή για ένα ευρύτερο φάσμα εργασιών. Τελικά, η επιλογή μεταξύ Matlab και Python θα εξαρτηθεί από τις συγκεκριμένες ανάγκες και απαιτήσεις του έργου, καθώς και από τις προσωπικές προτιμήσεις και την εμπειρία του χρήστη. Κατά τη διάρκεια της μετατροπής των αλγορίθμων από Matlab σε Python διαπιστώθηκε πως ενώ καταφέρνει η Python να ολοκληρώσει τις μαθηματικές εξισώσεις και τις πολύπλοκες μαθηματικές πράξεις που χρειάστηκε, οι κώδικες γίνονται πιο περίπλοκοι σε σύγκριση πάντα με το Matlab.

Βιβλιογραφία

[1] Δρ. Δημήτριος Βαρσάμης, "Διαφάνειες Εργαστηριακού Οδηγού" «Αριθμητική Ανάλυση και Επιστημονικός Προγραμματισμός», Οκτώβριος 2020

[2] Δρ. Δημήτριος Βαρσάμης, "Ανάπτυξη Λογισμικού - Python", Οκτώβριος 2019.

[3] Πλεξουσάκης Μιχάλης, "ΜΕΜ104 Γλώσσα Προγραμματισμού Ι (Μαθηματικά και Εφαρμοσμένα Μαθηματικά)", 3 Δεκεμβρίου 2019. [Σύνδεσμος]: http://users.tem.uoc.gr/~plex/mem104-Fall2019/lectures/lecture08.pdf

[4] Ταμπάκη Ειρήνη – Μαρία, "Mini εγχειρίδιο Python", Φεβρουάριος 2014. [Σύνδεσμος] : https://arch.icte.uowm.gr/docs/Mini_Python_Notebook/index.html

[5] Μόσχος Αχιλλεύς, "Διαδραστική -πολυμεσική αναπαράσταση αλγορίθμων ασφάλειας δικτύων", 2015 Θεσσαλονίκη. [Σύνδεσμος]: https://ikee.lib.auth.gr/record/280036/files/Moschos_2075.pdf

[6] Δημήτρης Λεβεντέας, TasPython, "Εκμάθηση Python Βήμα Βήμα «Οδηγός Python Μέσω Παραδειγμάτων»", [Σύνδεσμος]: https://fdocument.org/document/python-tutorial-in-greek.html?page=1

[7] Ζούλος Αλκιβιάδης Νικόλαος, "Πτυχιακή Εργασία «Ανάπτυξη Εργαστηριακών Εφαρμογών σε Python για Επεξεργασία Μεγάλων Δεδομένων»", Ιούλιος 2020. [Σύνδεσμος]: http://okeanis.lib2.uniwa.gr/xmlui/bitstream/handle/123456789/5381/%CE%91%CE%BB%CE%BA%CE%B9%CE%B2%CE%B9%CE%AC%CE%B4%CE%B7%CF%82%20%CE%96%CE%BF%CF%85%CE%BB%CF%8C%CF%82%20%CE%A0%CF%84%CF%85%CF%87%CE%B9%CE%B1%CE%BA%CE%AE%20%CE%95%CF%81%CE%B3%CE%B1%CF%83%CE%AF%CE%B1%20%CE%99%CE%BF%CF%8D%CE%BB%CE%B9%CE%BF%CF%82%202020%28CD%29.pdf%281%29.pdf?sequence=1&isAllowed=y

[8] Κωστοπούλου Στεργιανή, "Διπλωματική Εργασία «Ανάλυση Δεδομένων με την χρήση της Python» ", Ιούνιος 2018. [Σύνδεσμος]: https://ir.lib.uth.gr/xmlui/bitstream/handle/11615/49068/17857.pdf?sequence=1&isAllowed=y

[9] Εικόνα Εξώφυλλου. [Σύνδεσμος]: https://developers.redhat.com/articles/2023/01/17/how-fedora-and-tox-make-python-testing-easier