Τελευταία ενημέρωση 22/10/2025

Ανάλυση δεδομένων με τη γλώσσα προγραμματισμού Python ΕΚΔΔΑ

2.1 Εισαγωγή στην Python (2/2)

Εβδομάδα 2/9

Προετοιμασία διαφανειών: Γκόγκος Χρήστος

Σύγχρονα περιβάλλοντα ανάπτυξης εφαρμογών σε Python

Integrated Development Environments

IDEs

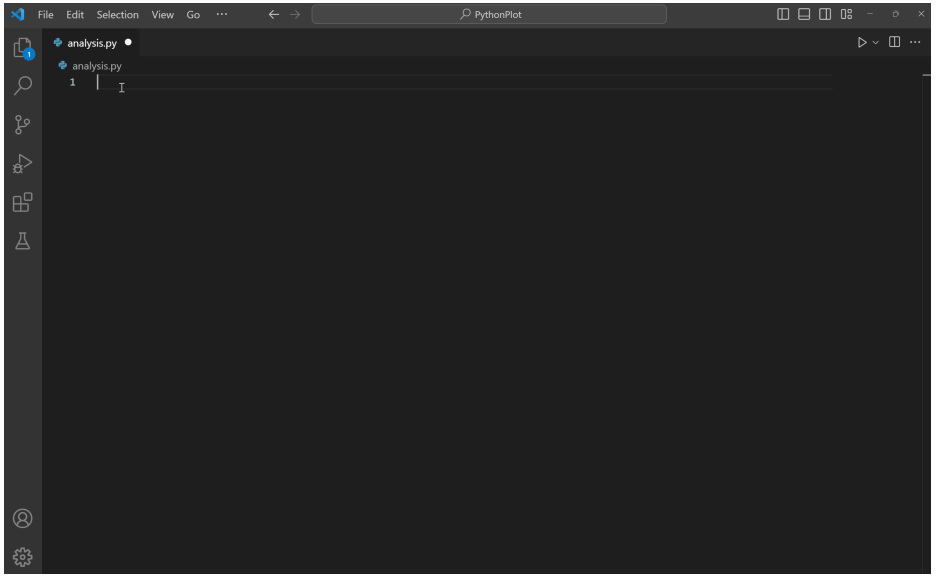
- IDE (Integrated Development Environment) είναι ένα περιβάλλον ανάπτυξης κώδικα που βασικό σκοπό έχει να προσφέρει στον προγραμματιστή όλα τα εργαλεία που χρειάζεται και τις ευκολίες που επιθυμεί έτσι ώστε το σύνολο της εργασίας του να μπορεί να επιτευχθεί μέσα από αυτό
- Ένα IDE τυπικά αυξάνει την παραγωγικότητα του προγραμματιστή καθώς μέσα από ένα κοινό περιβάλλον παρέχονται πολλές ευκολίες όπως η οργάνωση κώδικα σε projects, η εύκολη συγγραφή και εκτέλεση κώδικα, ο εντοπισμός και διόρθωση σφαλμάτων, η σύνδεση με λογισμικά όπως το git για διατήρηση εκδόσεων του κώδικα, η φόρτωση και η ενημέρωση εξωτερικών βιβλιοθηκών, η δημιουργία ιδεατών περιβαλλόντων κ.α.
- Διαδεδομένα IDEs για ανάπτυξη κώδικα σε Python είναι τα:
  - Visual Studio Code + Python addons
  - Jetbrains PyCharm
  - Spyder IDE
  - PyDev
- Συχνά, υπάρχουν προγραμματιστές που προτιμούν κάποιο άλλο IDE από τα παραπάνω ή προτιμούν κάποιο άλλο περιβάλλον ανάπτυξης κώδικα (π.χ. neovim, vim, emacs)

Visual Studio Code

- Το VS Code είναι ένα δωρεάν, ελαφρύ (lightweight) και ισχυρά επεκτάσιμο περιβάλλον για ανάπτυξη εφαρμογών σε Python, αλλά και σε άλλες γλώσσες προγραμματισμού
- Τα βασικά χαρακτηριστικά του VS Code που διευκολύνουν την ανάπτυξη κώδικα σε Python είναι:
  - Python extension: χρωματισμός κώδικα (syntax highlighting), αυτόματη συμπλήρωση κώδικα, άμεσος εντοπισμός σφαλμάτων (linting)
  - Debugger: βηματική εκτέλεση κώδικα, σημεία διακοπής εκτέλεσης (breakpoints), παρακολούθηση μεταβλητών (watches) κ.α.
  - Ενσωματωμένο τερματικό: εκτέλεση εντολών στο τερματικό χωρίς να φύγουμε από το VS Code
  - Διευκολύνσεις πλοήγησης στον κώδικα και refactoring (μαζικές αλλαγές στον κώδικα)
  - Υποστήριξη για ιδεατά περιβάλλοντα (virtual environments) που δημιουργούν ανεξάρτητες εγκαταστάσεις εκδόσεων της Python και βιβλιοθηκών
  - Υποστήριξη απευθείας εκτέλεσης Jupyter notebooks μέσα από το VS Code
  - Εγκατάσταση επιπλέον επεκτάσεων για την Python (π.χ. black, isort)
  - Version control για διατήρηση πλήρους ιστορικού αλλαγών στον κώδικα και συνεργασία σε ομάδες προγραμματιστών
  - Live Share: συνεργασία σε πραγματικό χρόνο με άλλους προγραμματιστές

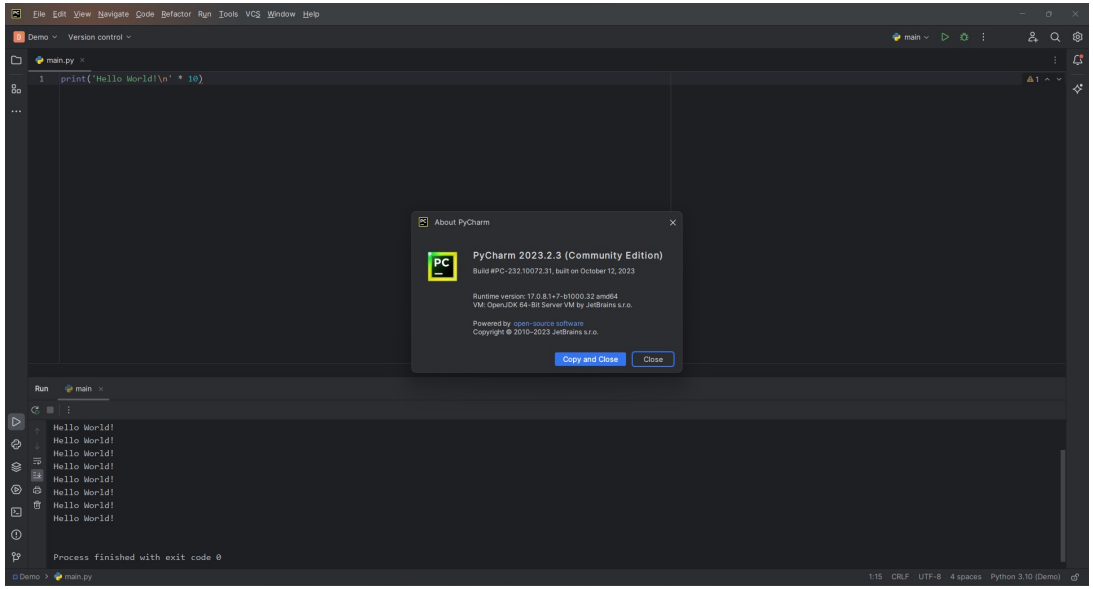
Εγκατάσταση του VS Code

- Εγκατάσταση του Visual Studio Code από το https://code.visualstudio.com/download
- Εγκατάσταση του VS Code Python extension, ακολουθήστε τις οδηγίες στο https://code.visualstudio.com/docs/python/python-tutorial



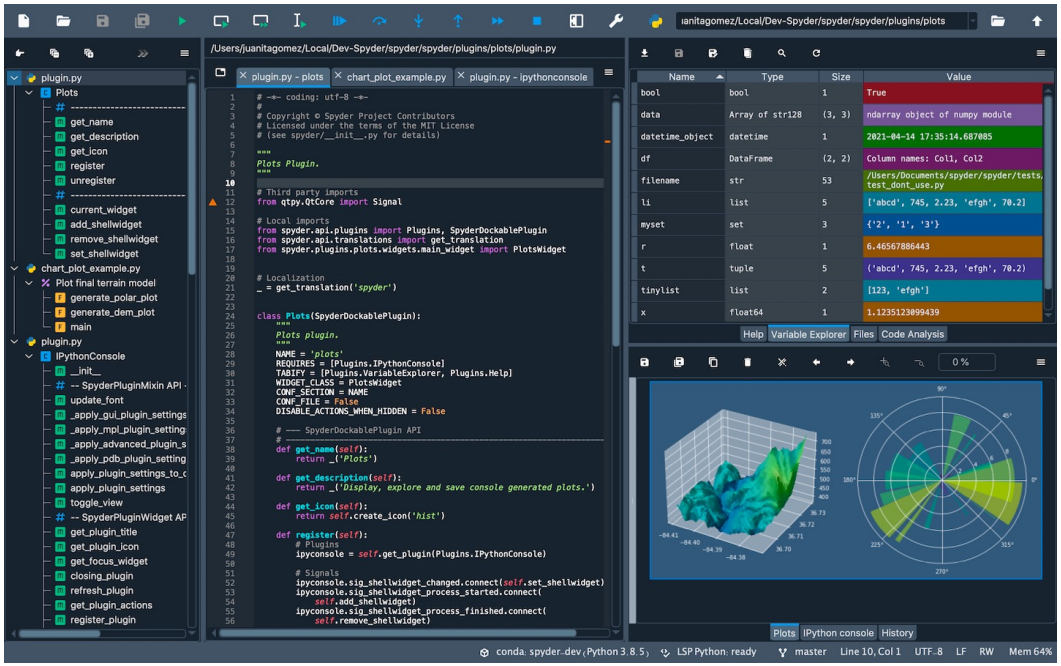
PyCharm

- Το PyCharm είναι ένα IDE που έχει αναπτυχθεί ειδικά για την Python από την Jetbrains
- Έχει πληθώρα χαρακτηριστικών και γενικά θεωρείται ως ένα επαγγελματικό περιβάλλον για την ανάπτυξη κώδικα Python ειδικά για μεγάλες εφαρμογές
- Διατίθεται σε δύο εκδόσεις:
  - **Community Edition**: είναι δωρεάν, open source και παρέχει πολλά χαρακτηριστικά που διευκολύνουν την ανάπτυξη κώδικα σε Python
  - **Professional Edition**: είναι εμπορικό λογισμικό, περιέχει επιπλέον δυνατότητες σε σχέση με το Community, π.χ. για Βάσεις Δεδομένων και Web Frameworks

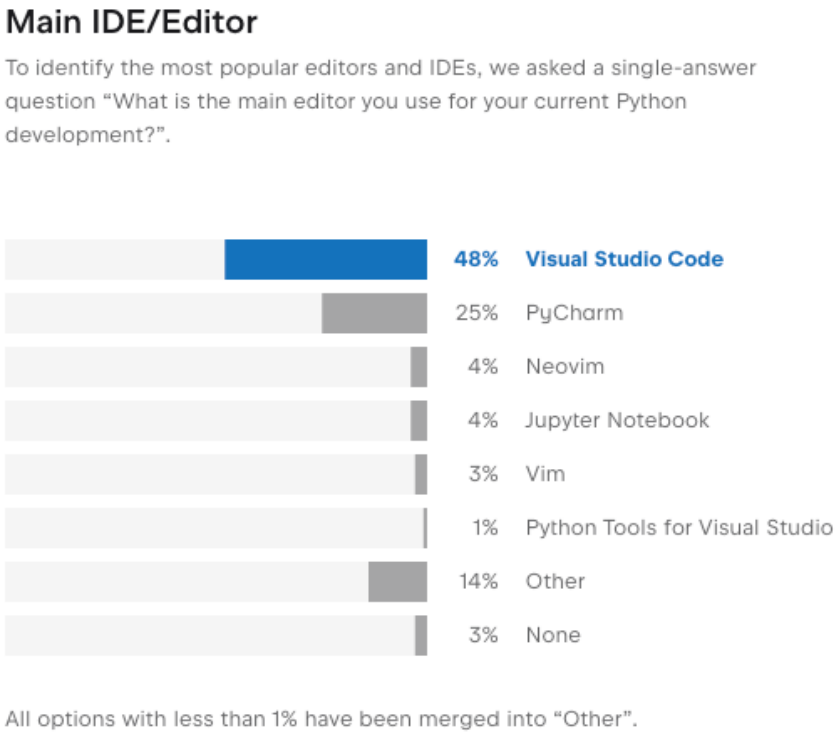


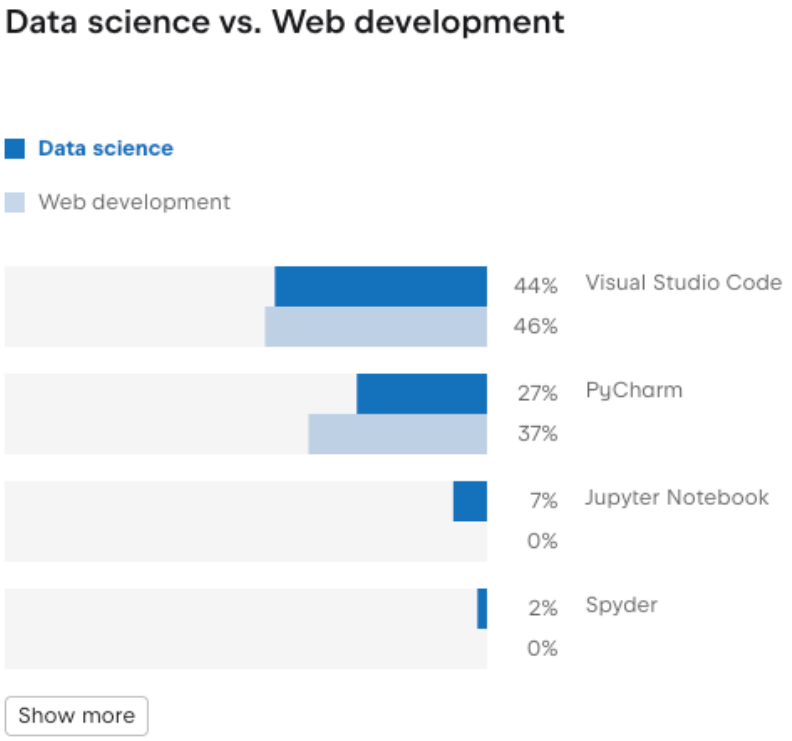
Spyder

- Το Spyder IDE εγκαθίσταται μαζί με τη διανομή Anaconda της Python και αποτελεί ένα ισχυρό IDE, εστιασμένο στην ανάπτυξη επιστημονικών εφαρμογών
- Μπορεί ωστόσο να εγκατασταθεί και αυτόνομα από το https://www.spyder-ide.org/
- Σε κάποιο βαθμό μοιάζει ως περιβάλλον με το MATLAB (π.χ. διαθέτει variable explorer)



Ποια IDEs είναι τα πλέον διαδεδομένα;





https://lp.jetbrains.com/python-developers-survey-2024/#development-tools

Modules και Packages

Οργάνωση κώδικα σε τμήματα και πακέτα

Modules

- Ένα module (τμήμα) είναι μια συλλογή από αντικείμενα Python που μπορεί να γίνει import
- Πρόκειται για ένα αρχείο .py που χρησιμοποιείται για λογική οργάνωση του κώδικα σε επαναχρησιμοποιήσιμα στοιχεία
- Ένα module μπορεί να περιέχει συναρτήσεις, κλάσεις, αντικείμενα κ.α.

Τύποι modules

- Built-in modules (έχουν γραφεί σε C και διατίθενται από τον διερμηνευτή της Python, χωρίς να φορτώνονται από αρχεία)
- Παραδείγματα built-in modules
  - sys => για πρόσβαση σε παραμέτρους και συναρτήσεις του συστήματος
  - math => για μαθηματικές συναρτήσεις
  - time => για συναρτήσεις χρόνου
  - os => για αλληλεπίδραση με το Λειτουργικό Σύστημα
  - random => για δημιουργία τυχαίων αριθμών
- Modules τρίτων κατασκευαστών που εγκαθίστανται με pip (π.χ., numpy, pandas)
- Modules του χρήστη/προγραμματιστή (user-defined)

Χρήση modules

- Έστω για παράδειγμα ότι θέλουμε να χρησιμοποιήσουμε από το module math τη σταθερά pi και τη συνάρτηση sin() για τον υπολογισμό του ημιτόνου των $30^o = \pi/6$
- Για να συμβεί αυτό θα πρέπει να δοθεί η εντολή:
  import math
- Οπότε για τον υπολογισμό του ημιτόνου των 30 μοιρών μπορεί να κληθεί η συνάρτηση ως εξής:
  math.sin(math.pi / 6)
- Η χρήση του ονόματος του module, ως namespace εξυπηρετεί ώστε αν έχουν και άλλα modules συνάρτηση (ή άλλο αντικείμενο) με το ίδιο όνομα (π.χ. sin ή pi) να μην προκαλείται σύγκρουση ονομάτων

```
Python 3.13.7 | packaged by conda-forge | (main, Sep  3 2025,
14:24:46) [Clang 19.1.7 ] on darwin
Type "help", "copyright", "credits" or "license" for more info
rmation.
>>> math.pi
Traceback (most recent call last):
  File "<python-input-0>", line 1, in <module>
    math.pi
    ^^^^
NameError: name 'math' is not defined. Did you forget to impor
t 'math'?
>>> import math
>>> x = math.pi / 6
>>> math.sin(x)
0.49999999999999994
>>> sin(x)
Traceback (most recent call last):
  File "<python-input-4>", line 1, in <module>
    sin(x)
    ^^^
NameError: name 'sin' is not defined. Did you mean: 'bin'?
>>> from math import sin
>>> sin(x)
0.49999999999999994
>>> pi
Traceback (most recent call last):
  File "<python-input-7>", line 1, in <module>
    pi
NameError: name 'pi' is not defined
>>> from math import *
>>> pi
3.141592653589793
```

Παράδειγμα ενός module ορισμένου από τον χρήστη (1/2)

- Έστω ένα module με όνομα my_mod.py που ενδεχόμενα περιέχει συναρτήσεις, κλάσεις και αναθέσεις αντικειμένων και σταθερών σε ονόματα μεταβλητών
- Τα παραπάνω αποτελούν τα λεγόμενα attributes (χαρακτηριστικά) του module
- Εκτός από αυτά τα attributes, υπάρχουν και άλλα attributes, κοινά για όλα τα modules, όπως τα:
  - __file__ => το όνομα αρχείου του module
  - __name__ => λαμβάνει την τιμή "__main__" αν εκτελείται απευθείας το ίδιο το module, αλλιώς λαμβάνει την τιμή __file__
  - __doc__ => το docstring του module
- Το module μπορεί να γίνει import από άλλα προγράμματα και τα attributes του να προσπελαστούν χρησιμοποιώντας σημειογραφία τελείας (dot notation)

Παράδειγμα ενός module ορισμένου από τον χρήστη (2/2)

Το module my_mod.py

```
"""
Ένα απλό module χρήστη
"""

PI = 3.14159

def square(x):
    return x * x


def hello(name):
    return f"Hello, {name}!"
```

main.py (στον ίδιο φάκελο με το my_mod.py)

```
import my_mod

print(my_mod.PI)  # 3.14159
print(my_mod.square(5))  # 25
print(my_mod.hello("Μαρία"))  # Hello, Μαρία!

print(my_mod.__file__)  # εμφανίζει το path του my_mod.py
print(my_mod.__name__)  # εμφανίζει "my_mod"
print(__name__)  # εμφανίζει "__main__"
print(my_mod.__doc__)  # εμφανίζει "Ένα απλό module χρήστη"
```

```
$ python main.py
3.14159
25
Hello, Μαρία!
/Users/.../my_mod.py
__main__
my_mod

Ένα απλό module χρήστη
```

__name__ == "__main__" (1/2)

- Έστω ένα αρχείο με όνομα a.py, που περιέχει συναρτήσεις αλλά και κάποιο κώδικα εκτός συναρτήσεων
- Αν το a.py γίνει import από ένα αρχείο b.py και το b.py εκτελεστεί τότε θα εκτελεστεί ο κώδικας εκτός συναρτήσεων του a.py (αυτό συνήθως δεν είναι επιθυμητό)
- Στο παράδειγμα κατά την εκτέλεση του b.py εμφανίζονται αποτελέσματα που δεν θα θέλαμε να εμφανιστούν

a.py

```
def my_function():
    print("Αυτή είναι μια συνάρτηση του module a!")


for _ in range(2):
    print("""Κάτι που πρέπει να γίνεται
όταν εκτελείται το a.py"""
    )
```

b.py

```
import a


a.my_function()
```

```
$ python b.py
Κάτι που πρέπει να γίνεται
όταν εκτελείται το a.py
Κάτι που πρέπει να γίνεται
όταν εκτελείται το a.py
Αυτή είναι μια συνάρτηση του module a!
```

__name__ == "__main__" (2/2)

- Για να αποφευχθεί η αυτόματη εκτέλεση κώδικα που βρίσκεται εκτός συναρτήσεων της a.py, όταν γίνεται import a σε άλλο module, ορίζεται ο κώδικας αυτός να εκτελείται μόνο αν
  __name__ == "__main__""
- Στο παράδειγμα κατά την εκτέλεση του b.py η συμπεριφορά του πλέον είναι η επιθυμητή

a.py

```
def my_function():
    print("Αυτή είναι μια συνάρτηση του module a!")

if __name__ == "__main__":
    # Κώδικας που εκτελείται μόνο όταν το a.py
    # τρέχει ως κύριο πρόγραμμα
    for _ in range(2):
        print("""Κάτι που πρέπει να γίνεται
    όταν εκτελείται το a.py"""
        )
```

b.py

```
import a


a.my_function()
```

```
$ python b.py
Αυτή είναι μια συνάρτηση του module a!
```

Packages

- Ένα package (πακέτο) είναι μια συλλογή από modules και άλλα packages σε έναν κατάλογο με ιεραρχική δομή
- Για παράδειγμα το package numpy έχει subpackages όπως τα core, linalg, random κ.α.
  - Το subpackage numpy.linalg περιέχει τα modules linalg, lapack_lite, info
- Συνεπώς, τα packages είναι μια ειδική περίπτωση modules που αναπαρίστανται ως φάκελοι στο σύστημα αρχείων
  - Κάθε φάκελος πρέπει να περιέχει ένα αρχείο __init__.py (που μπορεί ωστόσο να είναι άδειο) που αφορά την αρχικοποίηση του package/subpackage

```
A:
 [[1 2]
 [3 4]]
Αντίστροφος του A:
 [[-2.  1. ]
 [ 1.5 -0.5]]
A x A^-1:
 [[1. 0.]
 [0. 1.]]
```

```python
import numpy as np
from numpy import linalg

# αποφυγή επιστημονικής σημειογραφίας
np.set_printoptions(suppress=True)

A = np.array([[1, 2], [3, 4]])
print("A:\n", A)

# χρήση από το subpackage numpy.linalg
inv_A = linalg.inv(A)
print("Αντίστροφος του A:\n", inv_A)

I = A @ inv_A  # το γινόμενο A * A^-1 με τον τελεστή @
print("A x A^-1:\n", I)
```

Παράδειγμα δημιουργίας ενός package του χρήστη και κλήσης συναρτήσεων του

- Έστω ένα package του χρήστη με όνομα my_package που περιέχει δύο αρχεία .py:
  - math_utils.py
  - text_utils.py
- Η δομή του φακέλου είναι:

```
my_package/
  __init__.py
  math_utils.py
  text_utils.py
main.py
```

- Το αρχείο __init__.py στο παράδειγμα αυτό είναι κενό
- Το my_package πρόκειται να χρησιμοποιηθεί από το main.py

- Τα περιεχόμενα των αρχείων είναι:

math_utils.py

```python
def add(a, b):
    return a + b
```

text_utils.py

```python
def whisper(message):
    return message.lower() + "..."
```

main.py

```python
from my_package import text_utils, math_utils


print(math_utils.add(2, 3))  # 5
print(text_utils.whisper("Hello"))  # "hello..."
```

```
$ python main.py
5
hello...
```

Παράδειγμα δημιουργίας ενός package του χρήστη και κλήσης συναρτήσεων του (εναλλακτικός τρόπος)

- Η δομή του φακέλου παραμένει:

```
my_package/
  __init__.py
  math_utils.py
  text_utils.py
main.py
```

- Το αρχείο __init__.py στο παράδειγμα αυτό περιέχει:

__init__.py

```python
from .math_utils import add
from .text_utils import whisper
```

- Τα περιεχόμενα των υπόλοιπων αρχείων είναι:

math_utils.py

```python
def add(a, b):
    return a + b
```

text_utils.py

```python
def whisper(message):
    return message.lower() + "..."
```

main.py

```python
import my_package

print(my_package.math_utils.add(2, 3))  # 5
print(my_package.text_utils.whisper("Hello")) # "hello..."
```

```
$ python main.py
5
hello...
```

Πλεονεκτήματα των modules και των packages

- Επιτρέπουν την καλύτερη οργάνωση κώδικα και τη δημιουργία μεγάλων εφαρμογών
- Ενισχύουν την επαναχρησιμοποίηση κώδικα
- Διευκολύνουν τη συντήρηση κώδικα
- Επιτρέπουν το διαχωρισμό χώρων ονομάτων (namespaces) έτσι ώστε να αποφεύγονται συγκρούσεις ονομάτων

Άσκηση #1 (διαμερισμός κώδικα): Εκφώνηση

- Δίνεται ο κώδικας του data_loader.py
- Γράψτε το statistics.py στο οποίο:
  - Κάντε import το data_loader
  - Ορίστε τη συνάρτηση mean(data) που να υπολογίζει και να επιστρέφει το μέσο όρο της λίστας data
  - Καλέστε τη συνάρτηση mean() για τη λίστα που επιστρέφει η συνάρτηση load_data() του data_loader και εμφανίστε το αποτέλεσμα
  - Φροντίστε έτσι ώστε το μόνο που θα εμφανίζεται κατά την εκτέλεση του statistics.py να είναι ο μέσος όρος, ενώ η συμπεριφορά του data_loader.py κατά την εκτέλεσή του να παραμένει η ίδια και μετά τις όποιες αλλαγές σας

data_loader.py

```
def load_data():
    return [10, 20, 15, 21, 34, 23, 24, 30]


a_list = load_data()
print(a_list)
```

```
$ python data_loader.py
[10, 20, 15, 21, 34, 23, 24, 30]
```

Άσκηση #1: Λύση

data_loader.py

```python
def load_data():
    return [10, 20, 15, 21, 34, 23, 24, 30]

if __name__ == "__main__":
    a_list = load_data()
    print(a_list)
```

```
$ python data_loader.py
[10, 20, 15, 21, 34, 23, 24, 30]
```

statistics.py

```python
import data_loader


def mean(data):
    return sum(data) / len(data)


if __name__ == "__main__":
    data = data_loader.load_data()
    print(f"Mean: {mean(data)}")
```

```
$ python statistics.py
Mean: 22.125
```

Αντικειμενοστραφής προγραμματισμός

Object Oriented Programming (OOP)

Αντικειμενοστραφής Προγραμματισμός

- Ο αντικειμενοστραφής προγραμματισμός (OOP = Object Oriented Programming) είναι ένας τρόπος δόμησης λογισμικού που στοχεύει στη διαχείριση της πολυπλοκότητας που συνεπάγεται η συγγραφή μεγάλων και σύνθετων εφαρμογών
- Ενώ στον διαδικασιακό προγραμματισμό (procedural programming) οι συναρτήσεις και οι δομές δεδομένων στις οποίες επενεργούν οι συναρτήσεις είναι ξεχωριστές οντότητες, **στον OOP τα δεδομένα και οι συναρτήσεις που επενεργούν σε αυτά βρίσκονται μαζί**, οπότε η πρόσβαση σε κάθε ομάδα δεδομένων επιτρέπεται μόνο στις συναρτήσεις που έχει σχεδιαστεί να έχουν πρόσβαση
- Η μετάβαση από το διαδικασιακό προγραμματισμό στον OOP συνήθως βελτιώνει την επαναχρησιμοποίηση (reusability) του κώδικα καθιστώντας ευκολότερη τη συντήρηση και επέκταση του κώδικα
- Ο αντικειμενοστραφής προγραμματισμός είναι το κυρίαρχο προγραμματιστικό υπόδειγμα (programming paradigm) σήμερα
- Πολλές σημαντικές γλώσσες προγραμματισμού υποστηρίζουν (ή και επιβάλουν) τον αντικειμενοστραφή προγραμματισμό όπως η Python, C++, Java κ.α.

Κλάσεις και αντικείμενα

- Κεντρική έννοια στον OOP είναι η έννοια της κλάσης (class) που αναπαριστά μια κατηγορία αντικειμένων (objects)
- Ένα αντικείμενο διαθέτει κατάσταση (state) και συμπεριφορά (behavior), δηλαδή λειτουργίες που επιδρούν στην κατάστασή του
  - Η κατάσταση ενός αντικειμένου αφορά τα δεδομένα που αποθηκεύονται σε μεταβλητές που ανήκουν στο αντικείμενο και που ονομάζονται χαρακτηριστικά (attributes)
  - Η συμπεριφορά ενός αντικειμένου αφορά τις ενέργειες που μπορεί να επιτελέσει το αντικείμενο και ορίζονται ως συναρτήσεις στην κλάση του αντικειμένου που ονομάζονται μέθοδοι (methods)
  - Τόσο η πρόσβαση στις ιδιότητες ενός αντικειμένου όσο και στις μεθόδους του γίνεται με dot notation
- Η κλάση λειτουργεί ως "εργοστάσιο" παραγωγής αντικειμένων και κάθε αντικείμενο που δημιουργείται από μια κλάση αναφέρεται ως στιγμιότυπο (instance) της κλάσης
- Συχνά οι κλάσεις αναφέρονται και ως σχέδια (blueprints) δημιουργίας αντικειμένων

Παράδειγμα σταδιακής δημιουργίας μια κλάσης στο REPL

- Στο παράδειγμα αυτό αρχικά δημιουργείται μια κλάση με όνομα Rectangle χωρίς καθόλου περιεχόμενο
- Μετά, δημιουργείται ένα στιγμιότυπο της Rectangle, το αντικείμενο r1
- Μετά, προστίθενται οι ιδιότητες width και height στο αντικείμενο r1
- Μετά, προστίθεται στην κλάση μια μέθοδος area() που υπολογίζει και επιστρέφει το εμβαδόν ορθογωνίου για τα στιγμιότυπά της
- Τέλος, εκτυπώνονται οι τιμές των ιδιοτήτων του αντικειμένου r1 και καλείται σε αυτό η μέθοδος area(), το ίδιο συμβαίνει και για ένα ακόμη αντικείμενο, το r2
- Αυτός δεν είναι ο τυπικός τρόπος δημιουργίας κλάσεων και αντικειμένων αλλά δείχνει τη δυναμική φύση προγραμματισμού που υποστηρίζει η Python
- Δείτε στην επόμενη διαφάνεια έναν τυπικό τρόπο δημιουργίας της ίδιας κλάσης

```
>>> class Rectangle:
...     pass
...
>>> r1 = Rectangle()
>>> r1.width, r1.height = 4, 5
>>> def area(self):
...     return self.width * self.height
...
>>> Rectangle.area = area
>>>
>>> r1.width
4
>>> r1.height
5
>>> r1.area()
20
>>> r2 = Rectangle()
>>> r2.width, r2.height = 2, 7
>>> r2.area()
14
```

Η κλάση Rectangle ως ενιαίος κώδικας

self είναι το αντικείμενο για το οποίο καλείται η μέθοδος

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

r1 = Rectangle(4, 5)
print(f"Rectangle 1: {r1.width}x{r1.height}, area = {r1.area()}")
r2 = Rectangle(2, 7)
print(f"Rectangle 2: {r2.width}x{r2.height}, area = {r2.area()}")
```

αντιστοιχεί στο self της __init__

αντιστοιχεί στο self της area()

```
Rectangle 1: 4x5, area = 20
Rectangle 2: 2x7, area = 14
```

αντιστοιχεί στο self της area()

Η μέθοδος __init__

- Οι μέθοδοι που αναλαμβάνουν τη δημιουργία αντικειμένων ονομάζονται στον OOP κατασκευαστές (constructors)
- Η μέθοδος __init__ είναι μια μέθοδος αρχικοποίησης που εκτελείται αυτόματα όταν δημιουργείται ένα νέο στιγμιότυπο μιας κλάσης με μια εντολή της μορφής:

```
r1 = Rectangle(4, 5)
```

- Δέχεται τουλάχιστον μια παράμετρο, την self που αναφέρεται στο νέο αντικείμενο που δημιουργείται
- Δεν επιστρέφει το αντικείμενο που δημιουργείται, απλά το αρχικοποιεί (επιστρέφει None)

Η μέθοδος area() στο παράδειγμα της κλάσης Rectangle

- Η μέθοδος area() της κλάσης Rectangle όπως και κάθε άλλη μέθοδος οποιασδήποτε άλλης κλάσης έχει ως πρώτη παράμετρο το self που αντιστοιχεί στο αντικείμενο για το οποίο γίνεται η κλήση της μεθόδου

```
class Rectangle:
    ...
    def area(self):
        return self.width * self.height
```

Παράδειγμα δημιουργίας μιας κλάσης και στιγμιοτύπων της

- Έστω ένας ψηφιακός μετρητής που ξεκινά από το μηδέν και επιστρέφει στο μηδέν όταν φτάσει και ξεπεράσει μια μέγιστη τιμή
- Στο παράδειγμα αυτό θα δημιουργηθεί η κλάση DigitalCounter με ιδιότητες την τιμή του μετρητή (count) και τη μέγιστη τιμή μετρητή (max_value) και τις ακόλουθες μεθόδους:
  - μέθοδο set_max() που θέτει μέγιστη τιμή στο μετρητή
  - μέθοδο increment() που αυξάνει τον μετρητή κατά ένα
  - μέθοδο clear() που μηδενίζει τον μετρητή
- Στη συνέχεια θα δημιουργηθούν 2 μετρητές και θα κληθούν μέθοδοί τους

Η κλάση DigitalCounter

dc.py

```python
class DigitalCounter:
    def __init__(self, max_value=10):
        self.count = 0
        self.max_value = max_value

    def set_max(self, value):
        self.max_value = value

    def increment(self):
        self.count += 1
        if self.count > self.max_value:
            self.count = 0

    def clear(self):
        self.count = 0
```

main.py

```python
from dc import DigitalCounter

counter1 = DigitalCounter()
counter1.set_max(5)
for _ in range(7):
    counter1.increment()
    print(counter1.count)  # 1, 2, 3, 4, 5, 0, 1
counter1.clear()
print(counter1.count)  # 0
counter2 = DigitalCounter(3)
counter2.increment()
counter2.increment()
print(counter2.count) # 2
```

```
$ python main.py
1
2
3
4
5
0
1
0
1
2
```

Dunder μέθοδοι αντικειμένων

- Οι μέθοδοι που το όνομά τους ξεκινά και τελειώνει με 2 κάτω παύλες ονομάζονται dunder (**d**ouble **under**score) μέθοδοι ή αλλιώς magic μέθοδοι
- Οι μέθοδοι αυτοί επιτρέπουν να προσδιοριστεί κατάλληλα η συμπεριφορά built-in λειτουργιών των αντικειμένων, όπως η δημιουργία αντικειμένων, η εμφάνισή αντικειμένων όταν εκτυπώνονται κ.α.
- Η __init__ είναι μια dunder μέθοδος, αλλά υπάρχουν και άλλες όπως οι:
  - __str__ => επιστρέφει μια συμβολοσειρά που θα εκτυπώνεται όταν εκτυπώνεται το αντικείμενο
  - __repr__ => παρόμοια με την __str__(), χρησιμοποιείται για λόγους debugging, επιστρέφει μια μορφή κειμένου που μπορεί να χρησιμοποιηθεί για δημιουργία του αντικειμένου
  - __del__ => καλείται αυτόματα όταν καταστρέφεται το αντικείμενο, δηλαδή όταν δεν υπάρχουν πλέον μεταβλητές που να αποτελούν αναφορές σε αυτό

Παράδειγμα με τη μέθοδο __str__ και τη μέθοδο __repr__

- Η __str__ στοχεύει στους χρήστες τους προγράμματος και παρουσιάζει μια μορφή εκτύπωσης αντικειμένου που είναι εύκολα αναγνώσιμη και έχει ωραία μορφοποίηση
- Η __repr__ στοχεύει στους προγραμματιστές του προγράμματος και παρουσιάζει μια μορφή εκτύπωσης αντικειμένου που είναι χρήσιμη στην αποσφαλμάτωση του κώδικα και ιδανικά μπορεί να χρησιμοποιηθεί για να δημιουργηθεί από αυτή το αντικείμενο

```
class Book:
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price

    def __str__(self):
        return f"'{self.title}' by {self.author}"

    def __repr__(self):
        return (
            f"Book(title={self.title!r},
author={self.author!r}, price={self.price!r})"
        )


b = Book("1984", "George Orwell", 9.99)

print(b)
print(str(b))
print(repr(b))
```

```
'1984' by George Orwell
'1984' by George Orwell
Book(title='1984', author='George Orwell', price=9.99)
```

Παράδειγμα με τη μέθοδο __del__

- Η μέθοδος __del__ είναι ένας καταστροφέας (destructor), δηλαδή καλείται αυτόματα για ένα αντικείμενο όταν το αντικείμενο πρόκειται να καταστραφεί, όταν δεν υπάρχουν πλέον αναφορές σε αυτό
- Στο παράδειγμα δημιουργούνται υποθετικές συνδέσεις με κάποια υπηρεσία με κάθε σύνδεση να έχει έναν μοναδικό κωδικό, παρατηρήστε πότε καλείται κάθε φορά η __del__

```python
import uuid


class Connection:
    def __init__(self):
        self.code = str(uuid.uuid4()).split("-")[0]
        print(f"Η σύνδεση {self.code} άνοιξε.")

    def __del__(self):
        print(f"Η σύνδεση {self.code} έκλεισε.")


def demo():
    c1 = Connection()
    c2 = Connection()
    c3 = Connection()
    print("Εκτελείται εργασία...")

    del c2
    print("Μία σύνδεση διαγράφηκε.")

demo()
print("Η λειτουργία demo ολοκληρώθηκε.")
```

```
Η σύνδεση 8e139c19 άνοιξε.
Η σύνδεση b702af7c άνοιξε.
Η σύνδεση 6a0a764c άνοιξε.
Εκτελείται εργασία...
Η σύνδεση b702af7c έκλεισε.
Μία σύνδεση διαγράφηκε.
Η σύνδεση 8e139c19 έκλεισε.
Η σύνδεση 6a0a764c έκλεισε.
Η λειτουργία demo ολοκληρώθηκε.
```

Άσκηση #2 (κλάσεις & αντικείμενα): Εκφώνηση

- Να υλοποιήσετε μια κλάση Book, η οποία θα περιγράφει ένα βιβλίο με πεδία title, isbn και ratings (λίστα με βαθμολογίες, αρχικά κενή). Η κλάση θα πρέπει να περιλαμβάνει μέθοδο add_rating(score) που προσθέτει μια βαθμολογία από 1 έως 5, μέθοδο average_rating() που υπολογίζει και επιστρέφει το μέσο όρο των βαθμολογιών (ή μήνυμα αν δεν υπάρχουν βαθμολογίες), καθώς και μέθοδο __str__ που επιστρέφει σε μορφή κειμένου τον τίτλο, το ISBN και τη μέση βαθμολογία του βιβλίου.
- Στο κύριο πρόγραμμα να δημιουργηθούν τουλάχιστον δύο αντικείμενα της κλάσης Book, να προστεθούν βαθμολογίες και να εμφανιστούν τα βιβλία στην οθόνη.

Άσκηση #2: Λύση

```
class Book:
    def __init__(self, title, isbn):
        self.title = title
        self.isbn = isbn
        self.ratings = []

    def add_rating(self, score):
        if 1 <= score <= 5:
            self.ratings.append(score)
        else:
            print("Η βαθμολογία πρέπει να είναι από 1 έως 5.")

    def average_rating(self):
        if not self.ratings:
            return "Δεν υπάρχουν βαθμολογίες"
        return sum(self.ratings) / len(self.ratings)

    def __str__(self):
        avg = self.average_rating()
        return f"Τίτλος: {self.title}, ISBN: {self.isbn}, Μέση βαθμολογία: {avg}"

book1 = Book("Ο Μικρός Πρίγκιπας", "978-960-453-083-7"); book2 = Book("Η Φόνισσα", "978-960-01-0547-7")
book1.add_rating(5);book1.add_rating(4);book2.add_rating(4);book2.add_rating(4)
print(book1);print(book2)
```

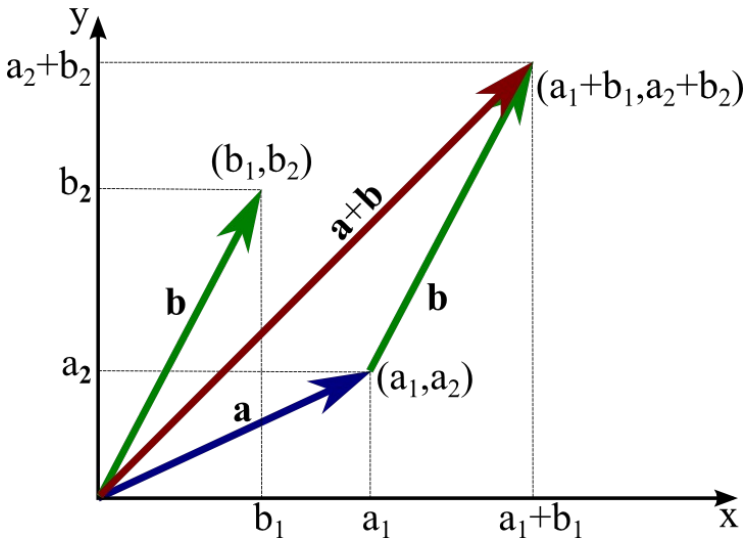
```
Τίτλος: Ο Μικρός Πρίγκιπας, ISBN: 978-960-453-083-7, Μέση βαθμολογία: 4.5
Τίτλος: Η Φόνισσα, ISBN: 978-960-01-0547-7, Μέση βαθμολογία: 4.0
```

Επιπλέον dunder μέθοδοι

- Η μέθοδος __add__ καθορίζει τη συμπεριφορά του τελεστή + για αντικείμενα μιας κλάσης, επιτρέπει δηλαδή να καθορίσουμε τι σημαίνει πρόσθεση για τα δικά μας αντικείμενα (αντίστοιχα υπάρχουν οι __sub__, __mul__, __truediv__ για αφαίρεση, πολλαπλασιασμό και διαίρεση καθώς και πολλές άλλες)
- Η μέθοδος __call__ επιτρέπει σε ένα αντικείμενο να κληθεί όπως θα καλούσαμε μια συνάρτηση (δηλαδή όταν χρησιμοποιούμε παρενθέσεις ( ) σε ένα αντικείμενο η Python εντοπίζει την __call__ μέθοδο της κλάσης του αντικειμένου)
- Η μέθοδος __getitem__ επιτρέπει στα αντικείμενα μιας κλάσης του χρήστη να χρησιμοποιούν τις αγκύλες, όπως χρησιμοποιούνται για παράδειγμα στις λίστες και στα λεξικά

Παράδειγμα με τη μέθοδο __add__

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        if isinstance(other, Vector):
            return Vector(self.x + other.x, self.y + other.y)
        return NotImplemented

    def __repr__(self):
        return f"Vector({self.x}, {self.y})"


v1 = Vector(3, 4)
v2 = Vector(1, 2)

v3 = v1 + v2
print(v3)  # Vector(4, 6)
```



https://mathinsight.org/image/vector_2d_add

Παράδειγμα με τη μέθοδο __call__

- Στο παράδειγμα αυτό η κλάση Multiplier αποθηκεύει έναν συντελεστή πολλαπλασιασμού (factor) σε κάθε αντικείμενό της κατά τη δημιουργία του αντικειμένου
- Τα αντικείμενα της Multiplier μπορούν να χρησιμοποιηθούν ως συναρτήσεις που πολλαπλασιάζουν το όρισμα που δέχονται κατά την κλήση τους με τον αποθηκευμένο συντελεστή
- Αυτό σημαίνει ότι η:
  `double = Multiplier(2)`
  δημιουργεί το αντικείμενο double, που μπορεί να κληθεί ως:
  `double(5)`
  και να πολλαπλασιάσει το όρισμα 5 με το 2 και να επιστρέψει την τιμή 10

```
class Multiplier:
    def __init__(self, factor):
        self.factor = factor

    def __call__(self, value):
        return value * self.factor


double = Multiplier(2)
triple = Multiplier(3)

print(double(5))  # 10
print(double(100))  # 200
print(triple(5))  # 15
print(triple(100))  # 300
```

```
10
200
15
300
```

Παράδειγμα με τη μέθοδο __getitem__

- Η κλάση GreekDictionary αποθηκεύει ένα λεξικό με αγγλικές λέξεις και τις ελληνικές μεταφράσεις τους
- Η __getitem__ επιτρέπει την πρόσβαση στο αποθηκευμένο λεξικό χρησιμοποιώντας αγκύλες όπως dictionary['hello']
- Έτσι, αντί για το: `dictionary.words['hello']` μπορούμε να γράψουμε: `dictionary['hello']` για να λάβουμε το ίδιο αποτέλεσμα.

```
class GreekDictionary:
    def __init__(self):
        # fmt: off
        self.words = {
            "hello": "γεια σου",
            "thank you": "ευχαριστώ",
            "goodbye": "αντίο"
        }
        # fmt: on

    def __getitem__(self, english_word):
        return self.words.get(english_word, "Δεν βρέθηκε")

dictionary = GreekDictionary()
print(dictionary["hello"])
print(dictionary["thank you"])
print(dictionary["water"])
```

```
γεια σου
ευχαριστώ
Δεν βρέθηκε
```

Σύγκριση αντικειμένων

- Η σύγκριση δύο αντικειμένων για ισότητα (με τον τελεστή ==) ελέγχει αν πρόκειται για το ίδιο αντικείμενο καθώς η προκαθορισμένη υλοποίηση είναι η υλοποίηση του τελεστή is
- Αλλά, οι κλάσεις μπορούν να ορίσουν τη μέθοδο __eq__ για να καθορίσουν τι σημαίνει η ισότητα (αντίστοιχα οι μέθοδοι για !=, <, <=, >, >= είναι οι __ne__, __lt__, __le__, __gt__, __ge)

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __eq__(self, other):
        if not isinstance(other, Point):
            return NotImplemented
        return self.x == other.x and self.y == other.y

    def __repr__(self):
        return f"Point({self.x}, {self.y})"

p1 = Point(2, 3)
p2 = Point(2, 3)
print(p1)        # Point(2, 3)
print(p1 == p2)  # True
```

Αν δεν υπήρχε η υλοποίηση της μεθόδου __eq__() στην κλάση Point τότε η σύγκριση p1 == p2 θα επέστρεφε False

Εισαγωγή αντικειμένων σε σύνολα και χρήση αντικειμένων ως κλειδιά λεξικών

- Τα σύνολα επιτρέπουν μόνο την εισαγωγή αντικειμένων που είναι hashable, δηλαδή αντικειμένων με υλοποίηση για τη μέθοδο __hash__
- Ομοίως για αντικείμενα ως κλειδιά λεξικών
- Γενικότερα ισχύει ότι: αν a == b, τότε θα πρέπει να ισχύει και hash(a) == hash(b) για να μπορούν να εισαχθούν και να λειτουργούν σωστά τα a και b σε σύνολα

Εισαγωγή αντικειμένων σε σύνολο

points_in_set_notok.py

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __eq__(self, other):
        if not isinstance(other, Point):
            return NotImplemented
        return self.x == other.x and self.y == other.y

    def __repr__(self):
        return f"Point({self.x}, {self.y})"

points = set()
points.add(Point(2,3))
points.add(Point(2,3))
print(points)
```

```
$ python points_in_set_notok.py
Traceback (most recent call last):
  File "points_in_set_notok.py", line 14, in <module>
    points.add(Point(2,3))
    ~~~~~~~~~~^^^^^^^^^^^^
TypeError: unhashable type: 'Point'
```

points_in_set_ok.py

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __eq__(self, other):
        if not isinstance(other, Point):
            return NotImplemented
        return self.x == other.x and self.y == other.y

    def __hash__(self):
        return hash((self.x, self.y))

    def __repr__(self):
        return f"Point({self.x}, {self.y})"

points = set()
points.add(Point(2,3))
points.add(Point(2,3))
print(points)
```

```
$ python points_in_set_ok.py
{Point(2, 3)}
```

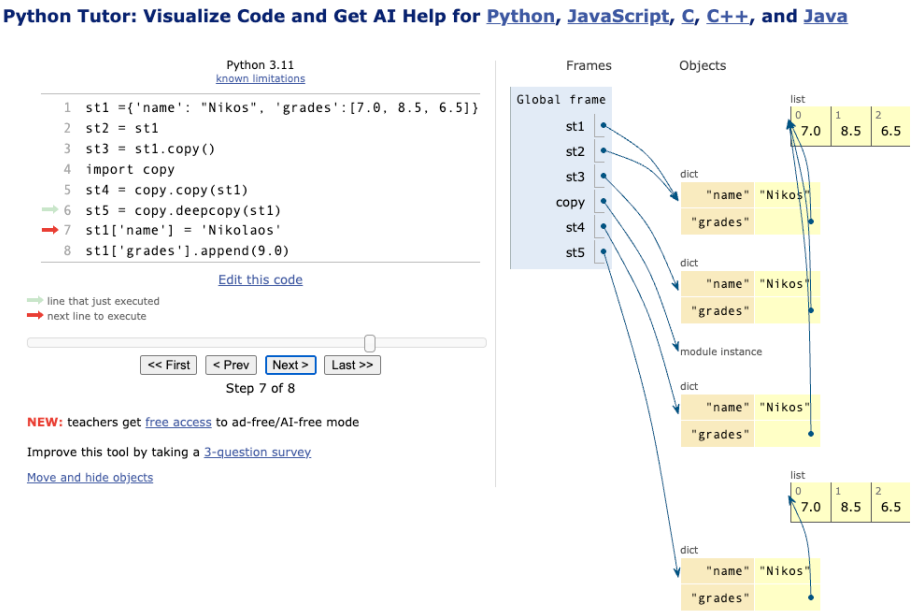
Αντιγραφή αντικειμένων

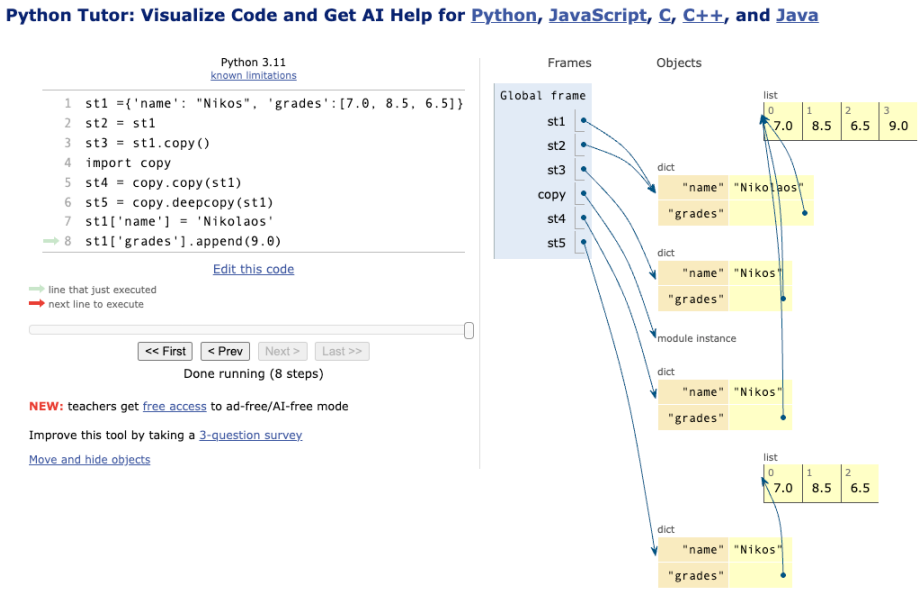
- Η αντιγραφή αντικειμένων είναι ένα θέμα που απαιτεί προσοχή
- Η απλή ανάθεση μιας μεταβλητής που είναι αναφορά σε ένα αντικείμενο σε μια άλλη μεταβλητή, δεν κάνει αντιγραφή, αλλά δύο αναφορές προς το ίδιο αντικείμενο
- Για να γίνει πραγματική αντιγραφή πρέπει να χρησιμοποιηθεί το module copy
- Η συνάρτηση copy.copy() δημιουργεί ένα νέο αντικείμενο και λειτουργεί σωστά για απλά αντικείμενα
- Η συνάρτηση copy.deepcopy() δημιουργεί ένα νέο αντικείμενο και αναδρομικά αντιγράφει όλα τα εμφωλευμένα αντικείμενα που υπάρχουν από το παλιό στο νέο αντικείμενο

- Παράδειγμα αντιγραφής αναφοράς με ανάθεση vs. αντιγραφής με copy.copy() και copy.deepcopy() για λεξικά με εμφωλευμένα αντικείμενα:

```
>>> st1 ={'name': "Nikos", 'grades':[7.0, 8.5, 6.5]}
>>> st2 = st1
>>> st3 = st1.copy()
>>> import copy
>>> st4 = copy.copy(st1)
>>> st5 = copy.deepcopy(st1)
>>> st1['name'] = 'Nikolaos'
>>> st1['grades'].append(9.0)
>>> st1
{'name': 'Nikolaos', 'grades': [7.0, 8.5, 6.5, 9.0]}
>>> st2
{'name': 'Nikolaos', 'grades': [7.0, 8.5, 6.5, 9.0]}
>>> st3
{'name': 'Nikos', 'grades': [7.0, 8.5, 6.5, 9.0]}
>>> st4
{'name': 'Nikos', 'grades': [7.0, 8.5, 6.5, 9.0]}
>>> st5
{'name': 'Nikos', 'grades': [7.0, 8.5, 6.5]}
```

Σχηματική αναπαράσταση αναφορών, και αντιγραφών





[Οπτικοποίηση εκτέλεσης κώδικα]

Ρηχή vs. βαθιά αντιγραφή αντικειμένων

```python
import copy


class ShoppingCart:
    def __init__(self, items=None):
        self.items = items or []  # λίστα με στοιχεία [όνομα, ποσότητα]

    def __repr__(self):
        return f"ShoppingCart({self.items})"


cart = ShoppingCart([["ΓΑΛΑ 1L", 1], ["ΜΕΛΙ 1KG", 1]])
shallow_cart = copy.copy(cart)
deep_cart = copy.deepcopy(cart)

# Τροποποίηση αρχικού αντικειμένου
cart.items[0][1] = 2

print("Αρχικό καλάθι  : ", cart)
print("Ρηχή αντιγραφή : ", shallow_cart)
print("Βαθιά αντιγραφή: ", deep_cart)
```

```
Αρχικό καλάθι  : ShoppingCart([['ΓΑΛΑ 1L', 2], ['ΜΕΛΙ 1KG', 1]])
Ρηχή αντιγραφή : ShoppingCart([['ΓΑΛΑ 1L', 2], ['ΜΕΛΙ 1KG', 1]])
Βαθιά αντιγραφή: ShoppingCart([['ΓΑΛΑ 1L', 1], ['ΜΕΛΙ 1KG', 1]])
```

Κληρονομικότητα

- Κληρονομικότητα (inheritance) είναι ένα βασικό χαρακτηριστικό του OOP που επιτρέπει σε μια κλάση να οριστεί ως υποκλάση μιας άλλης κλάσης (υπερκλάση), κληρονομώντας τα πεδία δεδομένων και τις μεθόδους της
- Η κληρονομικότητα διευκολύνει την επαναχρησιμοποίηση κώδικα (code reuse) καθώς οι υποκλάσεις δεν γράφονται από το μηδέν, αλλά χρησιμοποιούν ήδη υπάρχοντα κώδικα που τον εξειδικεύουν
- Για να κληρονομήσει μια κλάση Α από μια κλάση Β, θα πρέπει κατά τον ορισμό της κλάσης Α, η κλάση Β να τοποθετηθεί σε παρενθέσεις δίπλα από το όνομα της κλάσης Α

Παράδειγμα με κληρονομικότητα

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def work(self):
        return f"Ο {self.name} εργάζεται."

    def __repr__(self):
        return f"Employee(name={self.name}, salary={self.salary})"


class Manager(Employee):
    def __init__(self, name, salary, team_size):
        super().__init__(name, salary)  # κλήση κατασκευαστή υπερκλάσης
        self.team_size = team_size

    def work(self):
        return f"Ο {self.name} διευθύνει {self.team_size} άτομα."


e1 = Employee("Πέτρος", 2000);e2 = Manager("Μαρία", 4000, team_size=5)
print(e1);print(e1.work())
print(e2);print(e2.work())
```

```
Employee(name=Πέτρος, salary=2000)
Ο Πέτρος εργάζεται.
Employee(name=Μαρία, salary=4000)
Ο Μαρία διευθύνει 5 άτομα.
```

Άσκηση #3 (κληρονομικότητα): Εκφώνηση

- Να γράψετε ένα πρόγραμμα σε Python που να δείχνει τη χρήση της κληρονομικότητας (inheritance) μέσα από την έννοια των ηλεκτρονικών συσκευών. Δημιουργήστε μια βασική κλάση Device με ιδιότητες όπως brand και power (ισχύς σε Watt), καθώς και μια μέθοδο turn_on() που να εμφανίζει μήνυμα ότι η συσκευή ενεργοποιήθηκε.
- Στη συνέχεια, δημιουργήστε δύο υποκλάσεις: Laptop, που να προσθέτει την ιδιότητα battery_life και να υπερκαλύπτει τη μέθοδο turn_on() ώστε να εμφανίζει μήνυμα ότι ο φορητός υπολογιστής ενεργοποιήθηκε με μπαταρία, και TV, που να προσθέτει την ιδιότητα screen_size και να υπερκαλύπτει τη μέθοδο turn_on() ώστε να εμφανίζει μήνυμα ότι η τηλεόραση ενεργοποιήθηκε και προβάλλει εικόνα.
- Δημιουργήστε αντικείμενα και των δύο υποκλάσεων και καλέστε τη μέθοδο turn_on() για να φανεί η διαφορετική συμπεριφορά κάθε συσκευής.

Άσκηση #3: Λύση

```python
class Device:
    def __init__(self, brand, power):
        self.brand = brand
        self.power = power
    def turn_on(self):
        return f"Η συσκευή {self.brand} ισχύος {self.power}W ενεργοποιήθηκε."
    def __repr__(self):
        return f"Device(brand={self.brand}, power={self.power})"

class Laptop(Device):
    def __init__(self, brand, power, battery_life):
        super().__init__(brand, power)
        self.battery_life = battery_life
    def turn_on(self):
        return f"Ο φορητός υπολογιστής {self.brand} ενεργοποιήθηκε."
    def __repr__(self):
        return f"Laptop(brand={self.brand}, power={self.power},battery_life={self.battery_life})"

class TV(Device):
    def __init__(self, brand, power, screen_size):
        super().__init__(brand, power)
        self.screen_size = screen_size
    def turn_on(self):
        return f'Η τηλεόραση {self.brand} {self.screen_size} ενεργοποιήθηκε'
    def __repr__(self):
        return f"TV(brand={self.brand}, power={self.power}, screen_size={self.screen_size})"
```

```python
d1 = Device("Philips", 100)
l1 = Laptop("Dell", 65, 10)
t1 = TV("Samsung", 150, 55)

print(d1)
print(d1.turn_on())
print(l1)
print(l1.turn_on())
print(t1)
print(t1.turn_on())
```

```
Device(brand=Philips, power=100)
Η συσκευή Philips ισχύος 100W ενεργοποιήθηκε.
Laptop(brand=Dell, power=65,battery_life=10)
Ο φορητός υπολογιστής Dell ενεργοποιήθηκε.
TV(brand=Samsung, power=150, screen_size=55)
Η τηλεόραση Samsung 55 ενεργοποιήθηκε
```

Μεταβλητές κλάσης και μέθοδοι κλάσης

- Για τη διευκόλυνση της σχεδίασης λύσεων σε προβλήματα μπορεί να είναι χρήσιμο να οριστούν μεταβλητές που να είναι κοινές για όλα τα αντικείμενα μιας κλάσης και μέθοδοι που να επιδρούν πάνω σε αυτές τις μεταβλητές
- Οι μεταβλητές κλάσης ορίζονται εντός της κλάσης και εκτός των μεθόδων της
- Οι μέθοδοι κλάσης ορίζονται με την επισημείωση (annotation) @classmethod και λαμβάνουν ως πρώτο όρισμα το cls που αντιστοιχεί στην ίδια την κλάση
- Η πρόσβαση στις μεταβλητές κλάσης και στις μεθόδους κλάσης γίνεται μέσω του ονόματος της κλάσης (π.χ. Student.total_students) ή μέσω ενός αντικειμένου της κλάσης (π.χ. s1.total_students)

```python
class Student:
    total_students = 0

    def __init__(self, name):
        self.name = name
        Student.total_students += 1

    @classmethod
    def how_many(cls):
        return cls.total_students


s1 = Student("Νίκος")
s2 = Student("Μαρία")

print("s1:", s1.name)
print("s2:", s2.name)
print("Πλήθος σπουδαστών:", Student.how_many())
print("Πλήθος σπουδαστών:", Student.total_students)
print("Πλήθος σπουδαστών:", s1.total_students)
```

```
s1: Νίκος
s2: Μαρία
Πλήθος σπουδαστών: 2
Πλήθος σπουδαστών: 2
Πλήθος σπουδαστών: 2
```

Χρήση μεθόδου κλάσης ως εναλλακτικού κατασκευαστή

- Μια συνηθισμένη χρήση των μεθόδων κλάσης είναι για τη δημιουργία των λεγόμενων factory μεθόδων που λειτουργούν ως εναλλακτικός τρόπος δημιουργίας αντικειμένων
- Στο παράδειγμα η κλάση Circle έχει βασικό κατασκευαστή που δέχεται ως όρισμα την ακτίνα του κύκλου και δευτερεύοντα κατασκευαστή μέσω μεθόδου κλάσης που δέχεται ως όρισμα το εμβαδόν του κύκλου

```python
import math


class Circle:
    def __init__(self, radius):
        self.radius = radius

    @classmethod
    def from_area(cls, area):
        radius = math.sqrt(area / math.pi)
        return cls(radius)

    def area(self):
        return math.pi * self.radius**2


c1 = Circle(5)
c2 = Circle.from_area(78.54)

print(f"c1: {c1.radius:.2f}, area: {c1.area():.2f}")
print(f"c2: {c2.radius:.2f}, area: {c2.area():.2f}")
```

```
c1: 5.00, area: 78.54
c2: 5.00, area: 78.54
```

Στατικές μεταβλητές και στατικές μέθοδοι

- Οι μεταβλητές κλάσης που αναφέρθηκαν ήδη ονομάζονται αλλιώς και στατικές μεταβλητές ή στατικά δεδομένα της κλάσης
- Προσοχή όμως οι μέθοδοι κλάσης και οι στατικές μέθοδοι είναι διαφορετικές έννοιες
  - Οι στατικές μέθοδοι ορίζονται εντός μιας κλάσης με την επισημείωση @staticmethod αλλά δεν λαμβάνουν ως πρώτο όρισμα το self ή το cls
  - Ωστόσο, οι στατικές μέθοδοι δεν συνδέονται ούτε με την κλάση ούτε με τα στιγμιότυπα της κλάσης
  - Πρόκειται για συναρτήσεις που απλά γράφονται μέσα σε μια κλάση για καλύτερη οργάνωση κώδικα, αλλά δεν έχουν πρόσβαση στις μεταβλητές της κλάσης ούτε στις μεταβλητές των στιγμιοτύπων της κλάσης

```
class FileHelper:
    @staticmethod
    def get_extension(filename):
        """Επιστρέφει την επέκταση αρχείου"""
        return filename.split(".")[-1] if "." \
            in filename else None


print(FileHelper.get_extension("document.pdf"))
print(FileHelper.get_extension("archive.tar.gz"))
print(FileHelper.get_extension("no_extension"))
```

```
pdf
gz
None
```

Σύνθεση

- Η σύνθεση (composition) είναι ένας τρόπος συσχέτισης κλάσεων που αναφέρεται ως συσχέτιση τύπου "has-a" (έχει ένα)
- Χρησιμοποιείται όταν ένα αντικείμενο συντίθεται από άλλα αντικείμενα τα οποία διατηρεί ως χαρακτηριστικά
- Για παράδειγμα, η κλάση Car μπορεί να ορίζει το χαρακτηριστικό engine που να είναι αντικείμενο της κλάσης Engine, οπότε σε αυτή την περίπτωση λέμε ότι το αντικείμενο Car έχει ένα αντικείμενο Engine

```python
class Engine:
    def __init__(self, horsepower, engine_type):
        self.horsepower = horsepower
        self.engine_type = engine_type

    def start(self):
        print(f"{self.engine_type} engine with
{self.horsepower} HP started.")


class Car:
    def __init__(self, make, model, engine):
        self.make = make
        self.model = model
        self.engine = engine # Σύνθεση

    def start_car(self):
        print(f"Starting {self.make} {self.model}...")
        self.engine.start()


v6_engine = Engine(300, "V6")
my_car = Car("Toyota", "Supra", v6_engine)
my_car.start_car()
```

```
Starting Toyota Supra...
V6 engine with 300 HP started.
```

Άσκηση #4 (σύνθεση): Εκφώνηση

- Να γράψετε ένα πρόγραμμα σε Python που να αναπαριστά τη σχέση σύνθεσης (composition) μέσα από την αλληλεπίδραση μεταξύ ενός υπολογιστή και του επεξεργαστή του.
- Δημιουργήστε μια κλάση CPU με ιδιότητες όπως model και speed, καθώς και μια μέθοδο process() που να εμφανίζει μήνυμα επεξεργασίας δεδομένων.
- Δημιουργήστε μια κλάση Computer που να περιέχει ένα αντικείμενο τύπου CPU (σύνθεση) και να διαθέτει ιδιότητες όπως brand και ram, καθώς και μια μέθοδο start() που να εμφανίζει μήνυμα εκκίνησης του υπολογιστή και να καλεί τη μέθοδο process() της CPU.
- Δημιουργήστε αντικείμενα των δύο κλάσεων και δείξτε την αλληλεπίδραση τους μέσω της μεθόδου start().

Άσκηση #4: Λύση

```python
class CPU:
    def __init__(self, model, speed):
        self.model = model
        self.speed = speed

    def process(self):
        print(f"Η CPU {self.model} στα {self.speed}GHz επεξεργάζεται δεδομένα...")


class Computer:
    def __init__(self, brand, ram, cpu):
        self.brand = brand
        self.ram = ram
        self.cpu = cpu

    def start(self):
        print(f"Εκκίνηση υπολογιστή {self.brand} με {self.ram}GB RAM...")
        self.cpu.process()


cpu = CPU("Intel i7", 3.4)
my_computer = Computer("Dell", 16, cpu)
my_computer.start()
```

```
Εκκίνηση υπολογιστή Dell με 16GB RAM...
Η CPU Intel i7 στα 3.4GHz επεξεργάζεται δεδομένα...
```

Ενθυλάκωση

- Μια βασική αρχή του αντικειμενοστραφούς προγραμματισμού είναι η ενθυλάκωση (encapsulation) σύμφωνα με την οποία μπορούν να οριστούν επίπεδα προστασίας (private/protected/public) έτσι ώστε να περιορίζεται η πρόσβαση διαφόρων συναρτήσεων στα χαρακτηριστικά των αντικειμένων
- Στην Python δεν υπάρχουν τα private/protected/public αλλά υπάρχουν οι εξής συμβάσεις που ακολουθούνται από τους προγραμματιστές, αλλά δεν επιβάλλονται από τη γλώσσα:
  - όταν μια μεταβλητή μιας κλάσης ξεκινά με δύο κάτω παύλες η μεταβλητή θεωρείται ότι είναι private (επιτρέπεται η πρόσβαση μόνο από μεθόδους της ίδιας κλάσης)
  - όταν μια μεταβλητή μιας κλάσης ξεκινά με μια κάτω παύλα τότε θεωρείται ως protected (επιτρέπεται η πρόσβαση μόνο από μεθόδους της ίδιας κλάσης και των υποκλάσεών της)
  - όταν μια μεταβλητή μιας κλάσης δεν ξεκινά με κάτω παύλα τότε θεωρείται ως public (επιτρέπεται η πρόσβαση από οποιαδήποτε συνάρτηση ή μέθοδο)

Παράδειγμα "επιπέδων προστασίας"

```python
class BankAccount:
    def __init__(self, owner, balance):
        self.owner = owner  # public
        self._balance = balance  # protected
        self.__pin = 1234  # private

    def deposit(self, amount):
        self._balance += amount
        print(f"Κατάθεση ποσού {amount})

    def __verify_pin(self, pin):  # private
        return pin == self.__pin

    def withdraw(self, amount, pin):
        if self.__verify_pin(pin):
            if amount <= self._balance:
                self._balance -= amount
                print(f"Ανάληψη ποσού {amount}")
            else:
                print("Μη επαρκές υπόλοιπο.")
        else:
            print("Λανθασμένος PIN!")
```

```python
account = BankAccount("Christos", 1000)

print(account.owner)  # επιτρέπεται
# print(account._balance)  # επιτρέπεται ΔΕΝ συνιστάται
# print(account.__pin)     # AttributeError

account.deposit(200)
account.withdraw(100, 1234)

# Μπορούμε να προσπελάσουμε το private πεδίο με name mangling:
print(account._BankAccount__pin)
```

Αντικατάσταση getters και setters με ιδιότητες (properties)

- Οι μέθοδοι που είναι γνωστοί ως getters και setters είναι μέθοδοι που δίνουν πρόσβαση με ελεγχόμενο τρόπο στα μέλη δεδομένων της κλάσης
- Στο παράδειγμα ο setter δεν επιτρέπει την ανάθεση θερμοκρασίας μικρότερης από τους -273.15 βαθμούς κελσίου
- Οι getters και οι setters, συνίσταται να αντικαθίστανται από τα λεγόμενα properties για καθαρότερο και ιδιωματικό Python κώδικα

```
class Temperature:
    def __init__(self, celsius):
        self._celsius = celsius

    def get_celsius(self):
        return self._celsius

    def set_celsius(self, value):
        if value < -273.15:
            raise ValueError(
                "θερμοκρασία κάτω από το
απόλυτο μηδέν δεν μπορεί να υπάρξει."
            )
        self._celsius = value


t = Temperature(25)
print(t.get_celsius())  # 25
t.set_celsius(30)
print(t.get_celsius())  # 30
```

Getters/setters vs. properties

```python
class Temperature:
    def __init__(self, celsius):
        self._celsius = celsius

    def get_celsius(self):
        return self._celsius

    def set_celsius(self, value):
        if value < -273.15:
            raise ValueError(
                "θερμοκρασία κάτω από το
απόλυτο μηδέν δεν μπορεί να υπάρξει."
            )
        self._celsius = value


t = Temperature(25)
print(t.get_celsius())  # 25
t.set_celsius(30)
print(t.get_celsius())  # 30
```

annotation →

```python
class Temperature:
    def __init__(self, celsius):
        self._celsius = celsius

    @property
    def celsius(self):
        return self._celsius

    @celsius.setter
    def celsius(self, value):
        if value < -273.15:
            raise ValueError("θερμοκρασία κάτω
από το απόλυτο μηδέν δεν μπορεί να υπάρξει.")
        self._celsius = value


t = Temperature(25)
print(t.celsius)  # 25
t.celsius = 30
print(t.celsius)  # 30
```

pythonic

Αρχεία

Files

Γενικά περί αρχείων

- Κατά την εκτέλεση ενός προγράμματος τα δεδομένα βρίσκονται στην κύρια μνήμη του υπολογιστή, τη μνήμη RAM
  - Η μνήμη RAM δεν διατηρεί τα δεδομένα διαφορετικών εκτελέσεων του προγράμματος και είναι volatile, δηλαδή χάνει τα δεδομένα της όταν η τροδοφοσία ρεύματος σταματήσει
- Από την άλλη μεριά μόνιμη αποθήκευση μπορεί να επιτευχθεί αν τα δεδομένα αποθηκευτούν με τη μορφή αρχείου σε κάποιο αποθηκευτικό μέσο όπως είναι οι σκληροί δίσκοι (HDD = Hard Disk Drives), οι δίσκοι SSD (Solid State Drives), οι οπτικοί δίσκοι (CD=Compact Disks) κ.α.
  - Τα αρχεία οργανώνονται σε ιεραρχίες φακέλων για ευκολότερη πρόσβαση
  - Για να προσπελαστεί ένα αρχείο θα πρέπει να ανοιχθεί πρώτα προσδιορίζοντας αν πρόκειται να γίνει ανάγνωση (read) ή τροποποίηση (write), να ακολουθήσει πρόσβαση η επεξεργασία και τέλος το αρχείο να κλείσει

Αρχεία κειμένου

- Τα αρχεία κειμένου (text files) είναι αναγνώσιμα αν τα ανοίξουμε με ένα επεξεργαστή απλού κειμένου όπως το Notepad, το Notepad++, το VS Code κ.α.
- Συνήθως τα αρχεία κειμένου έχουν επέκταση .txt, αλλά πολλοί τύποι αρχείων με προγραμματιστικό ενδιαφέρον επίσης είναι αρχεία κειμένου, όπως τα αρχεία: CSV, XML, JSON, YAML, κ.α.
- Επίσης, τα αρχεία πηγαίου κώδικα (source code), στην Python αλλά και σε όλες τις γλώσσες προγραμματισμού είναι αρχεία κειμένου

Άνοιγμα αρχείου

- Η συνάρτηση με την οποία ανοίγουμε ένα αρχείο είναι η open() που δέχεται ως πρώτο όρισμα το όνομα του αρχείου και ως δεύτερο όρισμα την κατάσταση (mode) στην οποία θα ανοίξει το αρχείο, ενώ επιστρέφει ένα αντικείμενο αρχείου

f = open(<όνομα_αρχείου>, <κατάσταση>)

- Οι πιθανές καταστάσεις στις οποίες μπορεί να ανοίξει ένα αρχείο είναι:
  - r => ανοίγει το αρχείο για ανάγνωση, επιστρέφει σφάλμα αν το αρχείο δεν υπάρχει
  - w => ανοίγει το αρχείο για εγγραφή, το δημιουργεί αν δεν υπάρχει, το διαγράφει και το ξαναδημιουργεί αν δεν υπάρχει
  - a => ανοίγει το αρχείο για εγγραφή στο τέλος από τα πιθανά περιεχόμενά του (κάνει δηλαδή append), το δημιουργεί αν δεν υπάρχει

Παράδειγμα ανάγνωσης αρχείου κειμένου

- Έστω ότι υπάρχει ένα αρχείο κειμένου με όνομα lorem_ipsum.txt στον ίδιο φάκελο με το αρχείο κώδικα που το διαβάζει και μετρά όλα τα φωνήεντα του κειμένου

lorem_ipsum.txt

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed euismod, urna eu tincidunt consectetur, nisi nisl aliquam nunc, eget aliquam massa nisl quis neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Etiam at risus et justo dignissim congue. Donec congue lacinia dui, a porttitor lectus condimentum laoreet. Nunc eu ullamcorper orci. Quisque eget odio ac lectus vestibulum faucibus eget in metus.

count_vowels.py

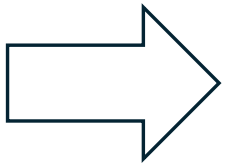
```
filename = "lorem_ipsum.txt"
vowels = "aeiouAEIOU"
count = 0

f = open(filename, "r")
for line in f.read():
    for char in line:
        if char in vowels:
            count += 1
f.close()

print(f"Πλήθος φωνηέντων: {count}")
```

```
$ python count_vowels.py
Πλήθος φωνηέντων: 199
```

Context manager

- Ένας τρόπος με τον οποίο μπορεί να ανοίγει και στη συνέχεια να κλείνει αυτόματα ένα αρχείο είναι με τη χρήση του λεγόμενου context manager με την εντολή with
- Ο κώδικας του προηγούμενου παραδείγματος γίνεται ισοδύναμα:

```
filename = "lorem_ipsum.txt"
vowels = "aeiouAEIOU"
count = 0

f = open(filename, "r")
for line in f.read():
    for char in line:
        if char in vowels:
            count += 1
f.close()

print(f"Πλήθος φωνηέντων: {count}")
```



```
filename = "lorem_ipsum.txt"
vowels = "aeiouAEIOU"
count = 0

with open(filename, "r") as f:
    for line in f.read():
        for char in line:
            if char in vowels:
                count += 1

print(f" Πλήθος φωνηέντων: {count}")
```

Παράδειγμα εγγραφής σε αρχείο κειμένου

- Στο παράδειγμα αυτό ο χρήστης εισάγει έναν θετικό ακέραιο αριθμό και εγγράφεται η ακολουθία Collatz που ξεκινά από αυτόν τον αριθμό σε ένα αρχείο κειμένου (out.txt) με 1 τιμή ανά γραμμή
- Η ακολουθία Collatz δημιουργείται με τον εξής απλό τρόπο: ξεκινά από έναν θετικό ακέραιο και αν είναι άρτιος τότε ο αριθμός διαιρείται ακέραια με το 2, αλλιώς πολλαπλασιάζεται με το 3 και προστίθεται το 1, και αυτό συμβαίνει μέχρι να προκύψει η τιμή 1 (π.χ. για αρχική τιμή 12 η ακολουθία διαμορφώνεται ως 12, 6, 3, 10, 5, 16, 8, 4, 2, 1)

```
s = input("Εισάγετε έναν θετικό ακέραιο: ")

if not s.strip().isdigit() or int(s) <= 0:
    print("Μη έγκυρη είσοδος.")
else:
    n = int(s)
    with open("out.txt", "w") as f:
        while True:
            f.write(f"{n}\n")
            if n == 1:
                break
            if n % 2 == 0:
                n //= 2
            else:
                n = 3 * n + 1
    print("Η ακολουθία Collatz γράφτηκε στο out.txt")
```

```
Εισάγετε έναν θετικό ακέραιο: 7
Η ακολουθία Collatz γράφτηκε στο out.txt
```

```
7
22
11
34
17
52
26
13
40
20
10
5
16
8
4
2
1
```

Αρχεία CSV

- Τα αρχεία CSV (Comma Separated Values) είναι αρχεία κειμένου που αποθηκεύουν τα δεδομένα σε μορφή πίνακα
- Συχνά, άλλα όχι απαραίτητα, η πρώτη γραμμή ενός CSV αρχείου είναι μια γραμμή με επικεφαλίδες τα ονόματα των στηλών
- Στη συνέχεια, κάθε γραμμή περιέχει μια εγγραφή που οι τιμές της χωρίζονται με κόμματα (,) ή με κάποιο άλλο χαρακτήρα (π.χ., διάστημα, tab, ελληνικό ερωτηματικό)
- Τα αρχεία CSV είναι δημοφιλή διότι:
  - Είναι απλά και υποστηρίζονται καθολικά (από λογιστικά φύλλα όπως το Excel, λογισμικά στατιστικής επεξεργασίας, βάσεις δεδομένων, γλώσσες προγραμματισμού)
  - Μπορούν να διαβαστούν και να γίνουν κατανοητά από τον χρήστη
  - Μπορούν εύκολα να μεταφερθούν μεταξύ διαφορετικών συστημάτων
- Η ανάγνωση αρχείων CSV και η εγγραφή σε αρχεία CSV μπορεί να γίνει εύκολα με τη βιβλιοθήκη PANDAS
- Ωστόσο, μπορεί να χρησιμοποιηθεί το module csv που ανήκει στην τυπική βιβλιοθήκη της Python και είναι ιδιαίτερα ελαφρύ (lightweight)

Ανάγνωση από αρχεία CSV με το module csv

- Το module csv διαθέτει τη μέθοδο DictReader() που επιστρέφει κάθε γραμμή του αρχείου CSV ως λεξικό όπου κλειδιά είναι τα ονόματα των στηλών που λαμβάνει από την πρώτη γραμμή του αρχείου
- Επίσης, το module csv διαθέτει τη μέθοδο reader() που επιστρέφει λίστες αντί για λεξικά
- Δείτε στη συνέχεια τους τρόπους που μπορούν να χρησιμοποιηθούν οι μέθοδοι DictReader() και reader() για την ανάγνωση των 3 πρώτων γραμμών δεδομένων από το αρχείο με τιμές της μετοχής της Google στο διάστημα 4/1/2010 έως 30/12/2022 που μπορείτε να μεταφορτώσετε από το https://www.kaggle.com/datasets/alirezajavid1999/google-stock-2010-2023 (απαιτεί δημιουργία λογαριασμού στο kaggle.com)

Google_Stock_Train (2010-2022).csv > data

```
1    Date,Open,High,Low,Close,Adj Close,Volume
2    2010-01-04,15.689439,15.753504,15.621622,15.684434,15.684434,78169752
3    2010-01-05,15.695195,15.711712,15.554054,15.615365,15.615365,120067812
4    2010-01-06,15.662162,15.662162,15.174174,15.221722,15.221722,158988852
5    2010-01-07,15.250250,15.265265,14.831081,14.867367,14.867367,256315428
6    2010-01-08,14.814815,15.096346,14.742492,15.065566,15.065566,188783028
7    2010-01-11,15.126627,15.126627,14.865866,15.042793,15.042793,288227484
8    2010-01-12,14.956206,14.968969,14.714715,14.776777,14.776777,193937868
9    2010-01-13,14.426677,14.724224,14.361862,14.691942,14.691942,259604136
10   2010-01-14,14.612112,14.869870,14.584835,14.761011,14.761011,169434396
11   2010-01-15,14.848348,14.853854,14.465465,14.514515,14.514515,217162620
12   2010-01-19,14.544545,14.775275,14.421672,14.705205,14.705205,172495332
13   2010-01-20,14.664164,14.664164,14.396647,14.524775,14.524775,129897972
14   2010-01-21,14.600601,14.685185,14.320571,14.589089,14.589089,252055692
15   2010-01-22,14.126627,14.279279,13.384885,13.764014,13.764014,271743984
16   2010-01-25,13.678428,13.760761,13.401151,13.513514,13.513514,176619204
17   2010-01-26,13.462713,13.753754,13.420671,13.574074,13.574074,174045780
18   2010-01-27,13.545295,13.704955,13.396146,13.566066,13.566066,158417424
19   2010-01-28,13.625876,13.688689,13.278278,13.370621,13.370621,129034836
20   2010-01-29,13.475726,13.538288,13.153403,13.261762,13.261762,165454380
```

```
3269 2022-12-23,87.110001,89.550003,87.070000,89.230003,89.230003,23003000
3270 2022-12-27,88.800003,88.940002,87.010002,87.389999,87.389999,20097300
3271 2022-12-28,86.980003,88.040001,85.940002,86.019997,86.019997,19523200
3272 2022-12-29,86.620003,88.849998,86.610001,88.449997,88.449997,23333500
3273 2022-12-30,86.980003,88.300003,86.570000,88.230003,88.230003,23986300
```

Ανάγνωση αρχείου CSV με το module csv με δύο τρόπους

Με csv.DictReader()

```
import csv

fn = "Google_Stock_Train (2010-2022).csv"
with open(fn, newline="", encoding="utf-8") as f:
    reader = csv.DictReader(f)
    for i, row in enumerate(reader):
        print(row)
        if i >= 2:
            break
```

```
{'Date': '2010-01-04', 'Open': '15.689439', 'High': '15.753504', 'Low': '15.621622', 'Close': '15.684434', 'Adj Close': '15.684434', 'Volume': '78169752'}
{'Date': '2010-01-05', 'Open': '15.695195', 'High': '15.711712', 'Low': '15.554054', 'Close': '15.615365', 'Adj Close': '15.615365', 'Volume': '120067812'}
{'Date': '2010-01-06', 'Open': '15.662162', 'High': '15.662162', 'Low': '15.174174', 'Close': '15.221722', 'Adj Close': '15.221722', 'Volume': '158988852'}
```

Με csv.reader()

```
import csv

fn = "Google_Stock_Train (2010-2022).csv"
with open(fn, newline='', encoding="utf-8") as f:
    reader = csv.reader(f)
    for i, row in enumerate(reader):
        print(row)
        if i >= 3:
            break
```

```
['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
['2010-01-04', '15.689439', '15.753504', '15.621622', '15.684434', '15.684434', '78169752']
['2010-01-05', '15.695195', '15.711712', '15.554054', '15.615365', '15.615365', '120067812']
['2010-01-06', '15.662162', '15.662162', '15.174174', '15.221722', '15.221722', '158988852']
```

Παράδειγμα εξαγωγής επιθυμητής πληροφορίας από CSV

- Εύρεση ημερομηνίας με τη μέγιστη ημερήσια διακύμανση τιμής για την μετοχή της Google από το 2010 μέχρι το 2022 (δεδομένα από το kaggle.com)

```python
import csv

fn = "Google_Stock_Train (2010-2022).csv"
with open(fn, newline="", encoding="utf-8") as f:
    reader = csv.DictReader(f)
    max_diff, max_row = -1, None
    for row in reader:
        low = float(row["Low"])
        high = float(row["High"])
        diff = high - low
        if diff > max_diff:
            max_diff = diff
            max_row = row
print(f"Μέγιστη διακύμανση: {max_diff:.2f}")
print(f"Ημερομηνία:  {max_row['Date']}")
```

```python
import pandas as pd

fn = "Google_Stock_Train (2010-2022).csv"
df = pd.read_csv(fn)
df["Range"] = df["High"] - df["Low"]
max_row = df.loc[df["Range"].idxmax()]

print(f"Μέγιστη διακύμανση: {max_row['Range']:.2f}")
print(f"Ημερομηνία: {max_row['Date']}")
```

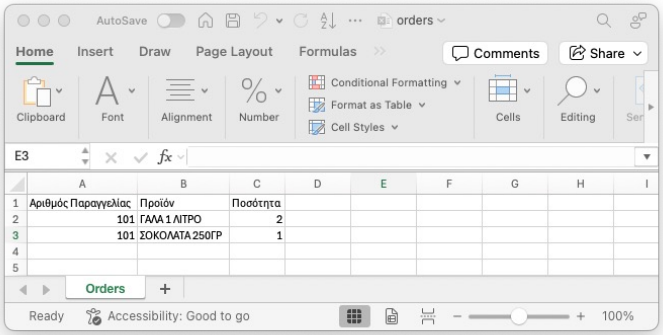
```
Μέγιστη διακύμανση: 9.24
Ημερομηνία:  2021-10-27
```

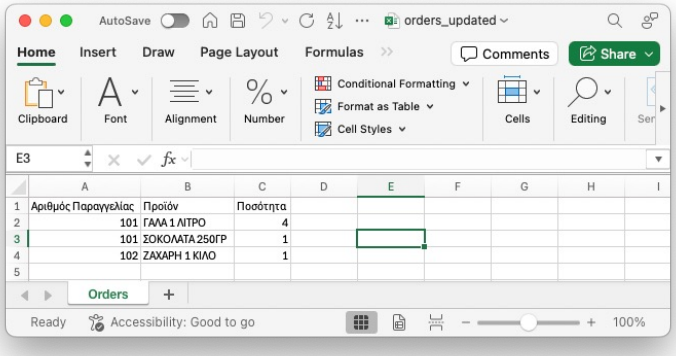
Αρχεία Excel

- Το Excel είναι ένα λογισμικό που έχει μεγάλη αποδοχή, ακόμη και σήμερα, σε επιχειρήσεις και οργανισμούς, λόγω ευκολίας χρήσης και πληθώρας χαρακτηριστικών
  - Ωστόσο, με τη χρήση της Python μπορούν εργασίες με αρχεία Excel να αυτοματοποιηθούν σε ακόμα μεγαλύτερο βαθμό και παράλληλα να χρησιμοποιηθούν οι επιπλέον δυνατότητες της Python και των βιβλιοθηκών της
- Η ανάγνωση αρχείων Excel και η εγγραφή αποτελεσμάτων σε αρχεία Excel μπορεί να γίνει εύκολα με τη βιβλιοθήκη PANDAS
- Ωστόσο, η openpyxl είναι μια ελαφρύτερη εξωτερική βιβλιοθήκη για ανάγνωση εγγραφή και τροποποίηση αρχείων Excel τύπου .xlsx (Excel 2010 και μεταγενέστερο)
  - Πρέπει να εγκατασταθεί με το pip για να μπορεί να χρησιμοποιηθεί (https://pypi.org/project/openpyxl/)
  - Δεν απαιτεί να είναι εγκατεστημένο το Excel για να χρησιμοποιηθεί
- Τα βήματα χειρισμού ενός αρχείου Excel με το openpyxl είναι τα ακόλουθα:
  1. Φόρτωση ενός αρχείου .xlsx (στην ορολογία του Excel, ονομάζεται Workbook)
  2. Πρόσβαση σε ένα φύλλο (sheet) του workbook
  3. Ανάγνωση, εγγραφή, τροποίηση κελιών (cells)
  4. Αποθήκευση αλλαγών

Παράδειγμα με openpyxl

```
from openpyxl import load_workbook

# 1. Φόρτωση αρχείου Excel (Workbook)
wb = load_workbook("orders.xlsx")
# 2. Πρόσβαση σε φύλλο (Sheet)
sheet = wb["Orders"]
# 3a. Ανάγνωση κελιών
print("Πρώτη σειρά του φύλλου:")
for cell in sheet[1]:  # 1η γραμμή (header)
    print(cell.value, end="\t")
print("\n")
# 3b. Ανάγνωση συγκεκριμένων κελιών
order_id = sheet["A2"].value
product = sheet["B2"].value
quantity = sheet["C2"].value
print(f"Order ID: {order_id}, Product: {product}, Quantity: {quantity}")
# 3c. Εγγραφή σε κελιά
sheet["A4"] = 102  # Αριθμός παραγγελίας
sheet["B4"] = "ΖΑΧΑΡΗ 1 ΚΙΛΟ"  # Προϊόν
sheet["C4"] = 1  # Ποσότητα
# 3d. Ενημέρωση κελιού
sheet["C2"] = quantity + 2  # αύξηση ποσότητας παραγγελίας
# 4. Αποθήκευση αλλαγών
wb.save("orders_updated.xlsx")
print("Οι αλλαγές αποθηκεύτηκαν στο 'orders_updated.xlsx'.")
```





Άσκηση #5 (αρχεία κειμένου): Εκφώνηση

- Γράψτε ένα πρόγραμμα που να διαβάζει δύο αρχεία κειμένου και να συγκρίνει τα περιεχόμενά τους. Συγκεκριμένα το πρόγραμμα θα πρέπει να εμφανίζει πόσες και ποιες λέξεις είναι κοινές στα αρχεία. Να αφαιρεθούν σημεία στίξης και να μην υπάρχει διάκριση πεζών κεφαλαίων.
- Για τα δύο αρχεία κειμένου εντοπίστε από τη wikipedia δύο άρθρα, ένα για τη Python και ένα για τη Java, αντιγράψτε από μια παράγραφο και δημιουργήστε το καθένα.

Άσκηση #5: Λύση

```
file1 = "python.txt"
file2 = "java.txt"

with open(file1, "r", encoding="utf-8") as f:
    text1 = f.read()

with open(file2, "r", encoding="utf-8") as f:
    text2 = f.read()

text1 = text1.lower()
text2 = text2.lower()

for ch in [".", ",", ";", ":", "!", "?", "(", ")", '"', "'"]:
    text1 = text1.replace(ch, "")
    text2 = text2.replace(ch, "")

words1 = set(text1.split())
words2 = set(text2.split())
common_words = words1.intersection(words2)
print("Αριθμός κοινών λέξεων:", len(common_words))
print("Κοινές λέξεις:", ", ".join(sorted(common_words)))
```

Παράδειγμα εξόδου

```
Αριθμός κοινών λέξεων: 13
Κοινές λέξεις: a, and, as, has, in, is, it, language, of, on, programming, the, to
```

Βάσεις Δεδομένων

Databases

Σχεσιακές Βάσεις Δεδομένων

- Οι Βάσεις Δεδομένων επιτρέπουν την οργανωμένη αποθήκευση δεδομένων και τη γρήγορη ανάκτηση και ενημέρωσή τους
- Οι Σχεσιακές Βάσεις Δεδομένων είναι η πλέον διαδεδομένη κατηγορία Βάσεων Δεδομένων
- Τα δεδομένα αποθηκεύονται σε πίνακες με διακριτό όνομα για τον καθένα, που αποτελούνται από πεδία και περιέχουν εγγραφές
- Κάθε πίνακας έχει ένα πεδίο (ή έναν συνδυασμό πεδίων) που αποτελεί το λεγόμενο πρωτεύον κλειδί (primary key) που αναγνωρίζει μοναδικά τις εγγραφές του
- Επίσης οι πίνακες συσχετίζονται μεταξύ τους με τα λεγόμενα ξένα κλειδιά (foreign key) – ένα ξένο κλειδί είναι ένα πεδίο ενός πίνακα που συσχετίζεται με το πρωτεύον κλειδί ενός άλλου πίνακα

Η γλώσσα SQL

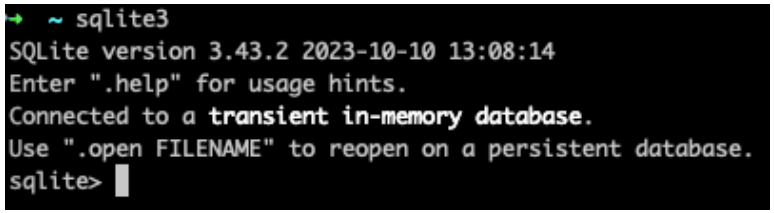
- Η SQL (Structured Query Language) είναι μια πρότυπη γλώσσα ερωτημάτων για σχεσιακές βάσεις δεδομένων
- Είναι πολύ δημοφιλής και χρησιμοποιείται από όλα τα σχεσιακά συστήματα διαχείρισης βάσεων δεδομένων (π.χ., Oracle, Microsoft SQL Server, IBM DB2, Postgres, MySQL, MariaDB, DuckDB, SQLite κ.α.)
  - Στη συνέχεια θα παρουσιαστούν παραδείγματα με την SQLite
- Μέσω εντολών SQL δίνεται η δυνατότητα για:
  - Δημιουργία/διαγραφή Βάσεων Δεδομένων
  - Δημιουργία/διαγραφή πινάκων
  - Εισαγωγή δεδομένων σε πίνακες
  - Ενημέρωση δεδομένων πινάκων
  - Ανάκτηση δεδομένων (η ανάκτηση δεδομένων μπορεί να αφορά πολλούς πίνακες που συνδέονται μεταξύ τους μέσω ξένων κλειδιών και αντιστοιχεί σε πολύπλοκα ερωτήματα)
  - Δημιουργία χρηστών και ανάθεση δικαιωμάτων (π.χ., ανάθεση δικαιώματος διαγραφής πινάκων)

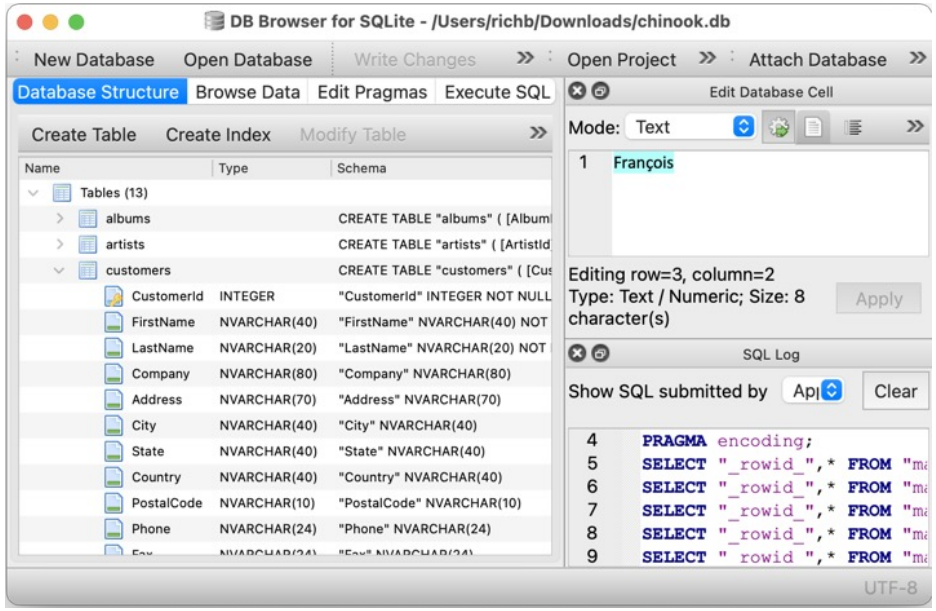
Βασικές εντολές SQL

Εντολές	Σύνταξη (γενική)	Παράδειγμα
Δημιουργία Βάσης	CREATE DATABASE όνομα;	CREATE DATABASE SchoolDB;
Διαγραφή Βάσης	DROP DATABASE όνομα;	DROP DATABASE SchoolDB;
Δημιουργία Πίνακα	CREATE TABLE όνομα_πίνακα (στήλη1 τύπος, στήλη2 τύπος, ...);	CREATE TABLE Students (ID INT, Name VARCHAR(50), Age INT);
Διαγραφή Πίνακα	DROP TABLE όνομα_πίνακα;	DROP TABLE Students;
Τροποποίηση Πίνακα	ALTER TABLE όνομα_πίνακα εντολή;	ALTER TABLE Students ADD Email VARCHAR(100);
Εισαγωγή Εγγραφής	INSERT INTO πίνακας (στήλες...) VALUES (τιμές...);	INSERT INTO Students (ID, Name, Age) VALUES (1, 'Maria', 20);
Διαγραφή Εγγραφής	DELETE FROM πίνακας WHERE συνθήκη;	DELETE FROM Students WHERE ID = 1;
Ενημέρωση Εγγραφής	UPDATE πίνακας SET στήλη = τιμή WHERE συνθήκη;	UPDATE Students SET Age = 21 WHERE Name = 'Maria';
Επιλογή Δεδομένων	SELECT στήλες FROM πίνακας WHERE συνθήκη;	SELECT Name, Age FROM Students WHERE Age > 18;
Ανάθεση Δικαιωμάτων	GRANT δικαιώματα ON βάση/πίνακας TO 'χρήστης'@'host';	GRANT SELECT, INSERT ON SchoolDB.* TO 'student_user'@'localhost';

SQLite

- Η SQLite είναι ένα ελαφρύ (lightweight) σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων με τα ακόλουθα κύρια χαρακτηριστικά:
  - Το μέγεθος της βιβλιοθήκης SQLite είναι <1MB
  - Η βάση δεδομένων αποθηκεύεται σε ένα αρχείο που είναι μεταφέρσιμο μεταξύ διαφορετικών συστημάτων (π.χ., Windows, Linux, Android)
  - Δεν δημιουργεί νέα διεργασία για την εκτέλεση της αλλά εκτελείται στην ίδια διεργασία με την εφαρμογή που τη χρησιμοποιεί
  - Είναι κατάλληλη για εφαρμογές που εκτελούνται σε κινητά (Android, iOS) και σε ενσωματωμένα (embedded) συστήματα
  - Απαιτεί μηδενικές ρυθμίσεις (zero configuration)
  - Υποστηρίζει προχωρημένα χαρακτηριστικά (transactions, triggers, indexin, views)
  - Είναι γρήγορη και αξιόπιστη





Πρόσβαση στην SQLite από την Python

- Τα βήματα που θα ακολουθήσουμε για επίδειξη των δυνατοτήτων αλληλεπίδρασης Python και SQLite είναι:
  1. Δημιουργία Βάσης Δεδομένων και σύνδεση (π.χ. SCHOOLDB)
  2. Δημιουργία πινάκων (π.χ. STUDENTS, GRADES) και σύνδεση μεταξύ τους με ξένα κλειδιά
  3. Εισαγωγή εγγραφών
  4. Ενημέρωση εγγραφών
  5. Ανάκτηση δεδομένων

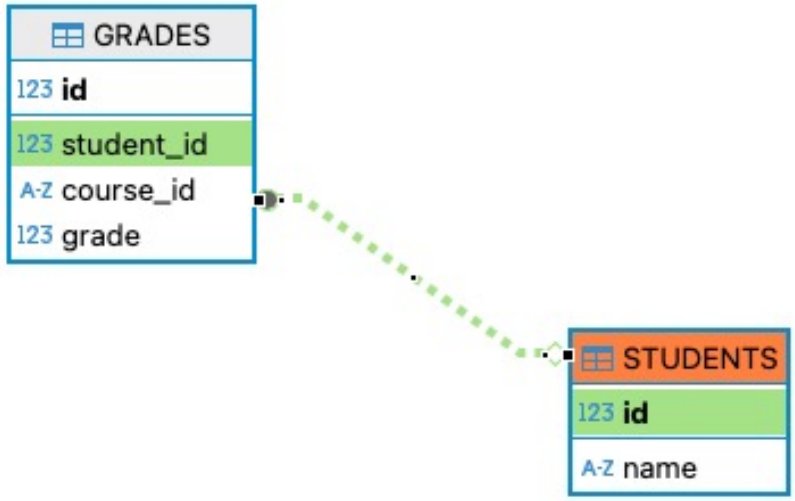
Python & SQLite (1/5): Δημιουργία Βάσης Δεδομένων

- Η Βάση Δεδομένων μπορεί να δημιουργηθεί χωρίς να γραφεί κώδικας Python, δίνοντας την κατάλληλη SQL εντολή στο περιβάλλον της SQLite ή μέσω κάποιου λογισμικού διαχείρισης της SQLite (π.χ. DB Browser for SQLite, DBeaver)
- Εδώ βλέπουμε τη δημιουργία της Βάσης Δεδομένων μέσω της Python

```python
import sqlite3

conn = sqlite3.connect("school.db")
print("Η Βάση Δεδομένων δημιουργήθηκε και συνδέθηκε.")
conn.close()
```

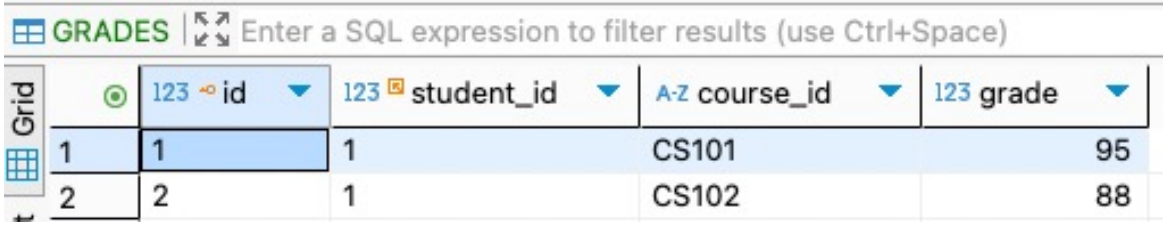
Python & SQLite (2/5): Δημιουργία πινάκων και σύνδεση τους μέσω ξένου κλειδιού

```
import sqlite3

conn = sqlite3.connect("school.db")
cursor = conn.cursor()

cursor.execute("""
CREATE TABLE IF NOT EXISTS STUDENTS (
    id INTEGER PRIMARY KEY,
    name TEXT
)
""")
cursor.execute("""
CREATE TABLE IF NOT EXISTS GRADES (
    id INTEGER PRIMARY KEY,
    student_id INTEGER,
    course_id TEXT,
    grade INTEGER,
    FOREIGN KEY(student_id) REFERENCES STUDENTS(id)
)
""")
conn.commit()
conn.close()
print("Οι πίνακες STUDENTS και GRADES δημιουργήθηκαν.")
```



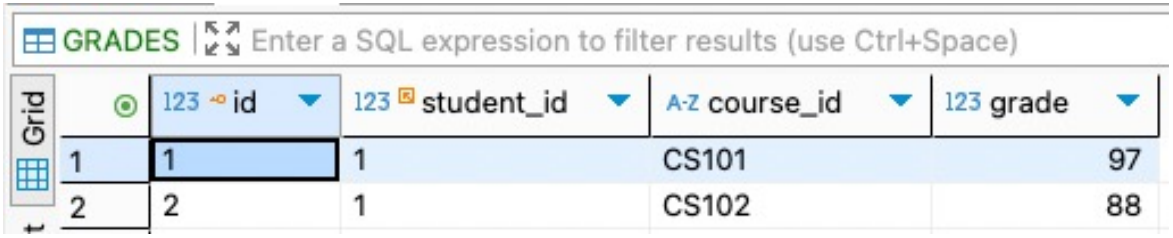
Python & SQLite (3/5): Εισαγωγή εγγραφών

```
import sqlite3

conn = sqlite3.connect("school.db")
cursor = conn.cursor()

cursor.execute("INSERT INTO STUDENTS (name) VALUES (?)", ("Αντώνης",))
cursor.execute("INSERT INTO GRADES (student_id, course_id, grade) VALUES (?, ?, ?)", (1, "CS101", 95))
cursor.execute("INSERT INTO GRADES (student_id, course_id, grade) VALUES (?, ?, ?)", (1, "CS102", 88))

conn.commit()
conn.close()
print("Εισήχθησαν εγγραφές στους πίνακες STUDENTS και GRADES.")
```



Python & SQLite (4/5): Ενημέρωση εγγραφών

```
import sqlite3


conn = sqlite3.connect("school.db")
cursor = conn.cursor()


cursor.execute(
"""UPDATE GRADES SET grade = ?
WHERE student_id = ? AND course_id = ?""",
(97, 1, "CS101"),
)


conn.commit()
conn.close()
print("Η εγγραφή βαθμολογίας ενημερώθηκε.")
```



Python & SQLite (5/5): Ανάκτηση δεδομένων

```
import sqlite3

conn = sqlite3.connect("school.db")
cursor = conn.cursor()

cursor.execute("""
SELECT STUDENTS.name, GRADES.course_id, GRADES.grade
FROM STUDENTS
JOIN GRADES ON STUDENTS.id = GRADES.student_id
""")

for row in cursor.fetchall():
    print(row)

conn.close()
```

```
('Αντώνης', 'CS101', 97)
('Αντώνης', 'CS102', 88)
```

Άσκηση #6 (SQLite): Εκφώνηση

- Γράψτε ένα πρόγραμμα που προσομοιώνει έναν μετρητή θερμοκρασίας και καταγράφει 20 τυχαίες μετρήσεις σε μια βάση δεδομένων SQLite με όνομα temperature.db. Το πρόγραμμα θα πρέπει να δημιουργεί τον πίνακα Measurements με την παρακάτω SQL εντολή:

```
CREATE TABLE Measurements (
    id INTEGER PRIMARY KEY,
    timestamp TEXT,
    temperature REAL
);
```

- Κάθε μέτρηση θα εισάγεται μαζί με την τρέχουσα χρονική σήμανση (timestamp) και μια τυχαία θερμοκρασία μεταξύ 20 και 30 βαθμών Κελσίου. Αφού εισαχθούν και οι 20 μετρήσεις, με καθυστέρηση 0.2 δευτερολέπτων η μια από την άλλη, το πρόγραμμα θα υπολογίζει και θα εμφανίζει τον μέσο όρο όλων των θερμοκρασιών χρησιμοποιώντας κατάλληλη εντολή SQL.

Άσκηση #6: Λύση

```
Εισαγωγή μέτρησης: 2025-10-01 09:44:16, 20.18 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:16, 29.12 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:17, 29.34 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:17, 21.86 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:17, 23.32 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:17, 20.19 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:17, 28.32 °C
...
Εισαγωγή μέτρησης: 2025-10-01 09:44:19, 21.38 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:20, 20.91 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:20, 21.06 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:20, 21.22 °C
Εισαγωγή μέτρησης: 2025-10-01 09:44:20, 29.25 °C
Μέση θερμοκρασία: 24.03 °C
```

```python
import sqlite3
import time
import random

conn = sqlite3.connect("temperature.db")
c = conn.cursor()
c.execute("DROP TABLE IF EXISTS Measurements")
c.execute("CREATE TABLE Measurements(id INTEGER PRIMARY KEY, timestamp TEXT, temperature REAL)")
conn.commit()
for _ in range(20):
    temp = round(random.uniform(20, 30), 2)
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
    c.execute("INSERT INTO Measurements (timestamp, temperature) VALUES (?, ?)", (timestamp, temp))
    print(f"Εισαγωγή μέτρησης: {timestamp}, {temp} °C")
    conn.commit()
    time.sleep(0.2)

c.execute("SELECT AVG(temperature) FROM Measurements")
avg_temp = c.fetchone()[0]
print(f"Μέση θερμοκρασία: {avg_temp:.2f} °C")
conn.close()
```

Το module sys

Διεπαφή με το περιβάλλον της Python

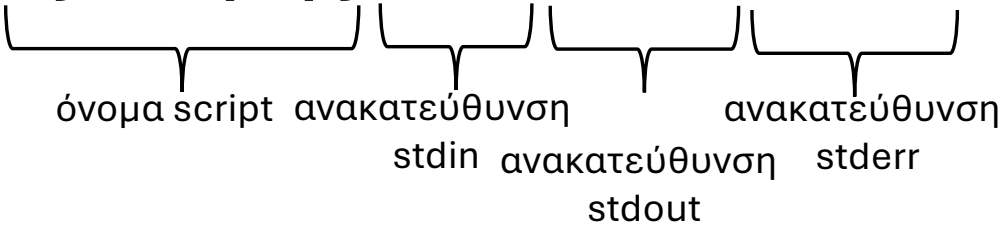
Βασικές πληροφορίες για το module sys

- Το module sys διαθέτει τα ακόλουθα αντικείμενα αρχείων:
  - sys.stdin => τυπική είσοδος
  - sys.stdout => τυπική έξοδος
  - sys.stderr => τυπική έξοδος σφαλμάτων
- Επίσης, διαθέτει χρήσιμα attributes όπως τα:
  - sys.path => λίστα φακέλων που χρησιμοποιεί η Python για να εντοπίσει τα modules που γίνονται import
  - sys.argv => για ορίσματα γραμμής εντολών
  - sys.version => τρέχουσα έκδοση της Python
  - sys.platform => υπολογιστικό σύστημα στο οποίο εκτελείται ο κώδικας
- Χρήσιμη είναι και η συνάρτηση sys.exit() που προκαλεί άμεση έξοδο (τερματισμό) του προγράμματος
  - Η sys.exit() δέχεται μια παράμετρο, που αν είναι μηδέν υποδηλώνει επιτυχή τερματισμό του προγράμματος, ενώ οποιαδήποτε μη μηδενική τιμή υποδηλώνει κατάσταση λάθους

Ανακατεύθυνση stdin, stdout, stderr

- Η ανακατεύθυνση των stdin, stdout, stderr μπορεί να γίνει είτε κατά την κλήση του προγράμματος Python είτε με εντολές μέσα στο ίδιο το πρόγραμμα
- Στο ακόλουθο παράδειγμα ο κώδικας του my_script.py, δέχεται είσοδο από το αρχείο "1.in" και εξάγει αποτελέσματα στα αρχεία "1.out" και "1.err"
- Η εντολή που πρέπει να δοθεί κατά την κλήση του προγράμματος για να γίνουν οι ανακατευθύνσεις είναι η:

python my_script.py < 1.in > 1.out 2>1.err



Παράδειγμα με ανακατεύθυνση stdin, stdout, stderr από τη γραμμή εντολών

my_script.py

```python
import sys
from math import sqrt

while True:
    x = input("Εισάγετε έναν αριθμό (-99 για έξοδο): ")
    if x == "-99":
        break
    try:
        r = sqrt(float(x))
        print(f"Η τετραγωνική ρίζα του {x} είναι {r}")
    except ValueError as e:
        sys.stderr.write(f"Σφάλμα: {e} - είσοδος: {x}\n")
        print("Παρακαλώ εισάγετε έναν μη αρνητικό αριθμό.")
    except TypeError as e:
        sys.stderr.write(f"Σφάλμα: {e} - είσοδος: {x}\n")
        print("Παρακαλώ εισάγετε έναν έγκυρο αριθμό.")
```

1.in

```
25
επτά
16
-1
36
-2
20
-99
```

```
$ python my_script.py < 1.in > 1.out 2>1.err
```

1.err

```
Σφάλμα: could not convert string to float: 'επτά' - είσοδος: επτά
Σφάλμα: math domain error - είσοδος: -1
Σφάλμα: math domain error - είσοδος: -2
```

1.out

```
Εισάγετε έναν αριθμό (-99 για έξοδο): Η τετραγωνική ρίζα του 25 είναι 5.0
Εισάγετε έναν αριθμό (-99 για έξοδο): Παρακαλώ εισάγετε έναν μη αρνητικό αριθμό.
Εισάγετε έναν αριθμό (-99 για έξοδο): Η τετραγωνική ρίζα του 16 είναι 4.0
Εισάγετε έναν αριθμό (-99 για έξοδο): Παρακαλώ εισάγετε έναν μη αρνητικό αριθμό.
Εισάγετε έναν αριθμό (-99 για έξοδο): Η τετραγωνική ρίζα του 36 είναι 6.0
Εισάγετε έναν αριθμό (-99 για έξοδο): Παρακαλώ εισάγετε έναν μη αρνητικό αριθμό.
Εισάγετε έναν αριθμό (-99 για έξοδο): Η τετραγωνική ρίζα του 20 είναι 4.47213595499958
Εισάγετε έναν αριθμό (-99 για έξοδο):
```

Το sys.argv

- Το sys.argv είναι μια λίστα με τα ορίσματα γραμμής εντολών που δόθηκαν για να εκτελεστεί το πρόγραμμα (script) Python
- Το πρώτο στοιχείο του sys.argv είναι το όνομα του προγράμματος και τα υπόλοιπα στοιχεία είναι τα ορίσματα που δίνονται από τον χρήστη

- Αν υποθέσουμε ότι ο ακόλουθος κώδικας έχει αποθηκευτεί με όνομα my_script.py

my_script.py

```
import sys

print("Όνομα script:", sys.argv[0])
print("Ορίσματα:", sys.argv[1:])
```

- Η εκτέλεση με:

```
$ python my_script.py ένα δύο 3
```

θα επιστρέψει:

```
Όνομα script: my_script.py
Ορίσματα: ['ένα', 'δύο', '3']
```

Άσκηση #7 (το module sys): Εκφώνηση

- Γράψτε ένα πρόγραμμα που χρησιμοποιεί το module sys για να χειριστεί στοιχεία εισόδου και εξόδου. Αρχικά, το πρόγραμμα πρέπει να διαβάζει όλη τη λίστα των ορισμάτων που δίνονται από τη γραμμή εντολών (sys.argv) και να εμφανίζει πόσα ορίσματα δόθηκαν και ποια είναι αυτά.
- Χρησιμοποιήστε το sys.exit() για να τερματίσετε το πρόγραμμα με διαφορετικούς κωδικούς εξόδου ανάλογα με το πλήθος των ορισμάτων. Αν δεν δόθηκε κανένα όρισμα, τερματίστε με κωδικό -1 και εμφανίστε μήνυμα σφάλματος, διαφορετικά συνεχίστε κανονικά και εμφανίστε μήνυμα επιτυχίας.
- Χρησιμοποιήστε το sys.version και το sys.platform για να εμφανίσετε στο χρήστη πληροφορίες για την τρέχουσα έκδοση της Python και την πλατφόρμα στην οποία εκτελείται το πρόγραμμα.

Άσκηση #7: Λύση

argv_example.py

```python
import sys


def main():
    args = sys.argv[1:]  # το πρώτο όρισμα που είναι το όνομα του script
    num_args = len(args)

    print(f"Αριθμός ορισμάτων: {num_args}")
    print("Ορίσματα:", args)

    if num_args == 0:
        print("Σφάλμα: Δεν δόθηκε κανένα όρισμα.")
        sys.exit(-1)
    else:
        print("Επιτυχής είσοδος ορισμάτων.")

    print(f"Python version: {sys.version}")
    print(f"Platform: {sys.platform}")


if __name__ == "__main__":
    main()
```

```
$ python argv_example.py 1 a True
Αριθμός ορισμάτων: 3
Ορίσματα: ['1', 'a', 'True']
Επιτυχής είσοδος ορισμάτων.
Python version: 3.10.12 | packaged by conda-forge |
(main, Jun 23 2023, 22:41:52) [Clang 15.0.7 ]
Platform: darwin
```

Εξαιρέσεις

Exceptions

Πως χειριζόμαστε μια κατάσταση λάθους καθώς εκτελείται ένα πρόγραμμα;

- Όταν συμβεί ένα σφάλμα (error) υπάρχουν πολλές προγραμματιστικές επιλογές για το τι πρόκειται να συμβεί όπως:
  - Αγνόηση του λάθους
  - Τερματισμός εκτέλεσης του προγράμματος
  - Εκτύπωση ενός μηνύματος λάθους και συνέχεια εκτέλεσης
  - Προσπάθεια επιδιόρθωσης του σφάλματος
- Οι περισσότερες γλώσσες προγραμματισμού διαθέτουν έναν μηχανισμό χειρισμού λαθών που λέγεται εξαιρέσεις (exceptions) και χειρισμός εξαιρέσεων (exceptions handling)
- Ο εντοπισμός ενός σφάλματος σηματοδοτείται από τη λεγόμενη πρόκληση εξαίρεσης (exception raise) που γίνεται με την εντολή:
  - raise <στιγμιότυπο εξαίρεσης>
- Ο χειρισμός της εξαίρεσης γίνεται με το σχήμα εντολών try/except

```
try:
   # εντολές που μπορεί να δημιουργήσουν εξαιρέσεις
except (Exception1, Exception2, ...):
   # εδώ γίνεται ο χειρισμός της εξαίρεσης
else:
   # εδώ εκτελείται αν το μπλοκ στον try κώδικα
   # δεν προκάλεσε εξαιρέσεις
finally:
   # εδώ ο κώδικας εκτελείται πάντα
```

Παράδειγμα χειρισμού εξαιρέσεων

```
def average_positive(numbers):
    """Υπολογίζει το μέσο όρο των θετικών αριθμών στη λίστα."""
    positives = [num for num in numbers if num > 0]
    return sum(positives) / len(positives)


alist = []
while True:
    x = input("Εισήγαγε έναν αριθμό (κενό για τερματισμό): ")
    if x == "":
        break
    try:
        x = float(x)
        alist.append(x)
    except ValueError:
        print("Εισήχθει μη αριθμητική τιμή, προσπαθήστε ξανά")

try:
    avg = average_positive(alist)
    print(f"Μέσος όρος θετικών αριθμών: {avg}")
except ZeroDivisionError:
    print("Δεν εισήχθησαν θετικοί αριθμοί.")
```

```
Εισήγαγε έναν αριθμό (κενό για τερματισμό): 5
Εισήγαγε έναν αριθμό (κενό για τερματισμό): -76
Εισήγαγε έναν αριθμό (κενό για τερματισμό): 4
Εισήγαγε έναν αριθμό (κενό για τερματισμό): 1
Εισήγαγε έναν αριθμό (κενό για τερματισμό): -2
Εισήγαγε έναν αριθμό (κενό για τερματισμό): πέντε
Εισήχθει μη αριθμητική τιμή, προσπαθήστε ξανά
Εισήγαγε έναν αριθμό (κενό για τερματισμό): 5
Εισήγαγε έναν αριθμό (κενό για τερματισμό):
Μέσος όρος θετικών αριθμών: 3.75
```

Παράδειγμα πρόκλησης και χειρισμού εξαιρέσεων

```
Δώσε κείμενο και επαναλήψεις: Hello 3
Αποτέλεσμα:  HelloHelloHello

Δώσε κείμενο και επαναλήψεις: Hello aaa
ValueError: invalid literal for int() with base 10: 'aaa'

Δώσε κείμενο και επαναλήψεις: Hello -2
ValueError: times must be non-negative.

Δώσε κείμενο και επαναλήψεις: Hello
ValueError: not enough values to unpack (expected 2, got 1)
```

```python
def repeat_text(text, times):
    if not isinstance(times, int):
        raise TypeError("times must be an integer.")
    if times < 0:
        raise ValueError("times must be non-negative.")
    return text * times


try:
    text, times = input("Δώσε κείμενο και επαναλήψεις: ").split()
    times = int(times)
    result = repeat_text(text, times)
    print("Αποτέλεσμα: ", result)
except TypeError as e:
    print("TypeError:", e)
except ValueError as e:
    print("ValueError:", e)
```

Ιεραρχία εξαιρέσεων

- Στην Python οι εξαιρέσεις είναι στιγμιότυπα κλάσεων εξαιρέσεων
- Υπάρχει ιεραρχία (από το γενικότερο στο ειδικότερο) στις κλάσεις εξαιρέσεων και στην κορυφή της ιεραρχίας βρίσκεται η κλάση Exception
- Ο χρήστης/προγραμματιστής μπορεί:
  - είτε να χρησιμοποιήσει υπάρχουσες κλάσεις εξαιρέσεων
  - είτε να δημιουργήσει τις δικές του κλάσεις εξαιρέσεων που κληρονομούν από υπάρχουσες κλάσεις εξαιρέσεων

```
BaseException
 ├── BaseExceptionGroup
 ├── GeneratorExit
 ├── KeyboardInterrupt
 ├── SystemExit
 └── Exception
      ├── ArithmeticError
      │    ├── FloatingPointError
      │    ├── OverflowError
      │    └── ZeroDivisionError
      ├── AssertionError
      ├── AttributeError
      ├── BufferError
      ├── EOFError
      ├── ExceptionGroup [BaseExceptionGroup]
      ├── ImportError
      │    └── ModuleNotFoundError
      ├── LookupError
      │    ├── IndexError
      │    └── KeyError
      ├── MemoryError
      ├── NameError
      │    └── UnboundLocalError
      ├── OSError
      │    ├── BlockingIOError
      │    ├── ChildProcessError
      │    ├── ConnectionError
      │    │    ├── BrokenPipeError
      │    │    ├── ConnectionAbortedError
      │    │    ├── ConnectionRefusedError
      │    │    └── ConnectionResetError
      │    ├── FileExistsError
      │    ├── FileNotFoundError
      │    ├── InterruptedError
      │    ├── IsADirectoryError
      │    ├── NotADirectoryError
      │    ├── PermissionError
      │    ├── ProcessLookupError
      │    └── TimeoutError
      ├── ReferenceError
      ├── RuntimeError
      │    ├── NotImplementedError
      │    ├── PythonFinalizationError
      │    └── RecursionError
      ├── StopAsyncIteration
      ├── StopIteration
      ├── SyntaxError
      │    └── IndentationError
      │         └── TabError
      ├── SystemError
      ├── TypeError
      ├── ValueError
      │    └── UnicodeError
      │         ├── UnicodeDecodeError
      │         ├── UnicodeEncodeError
      │         └── UnicodeTranslateError
      └── Warning
           ├── BytesWarning
           ├── DeprecationWarning
           ├── EncodingWarning
           ├── FutureWarning
           ├── ImportWarning
           ├── PendingDeprecationWarning
           ├── ResourceWarning
           ├── RuntimeWarning
           ├── SyntaxWarning
           ├── UnicodeWarning
           └── UserWarning
```

https://docs.python.org/3/library/exceptions.html#exception-hierarchy

Παράδειγμα με ορισμό προσαρμοσμένης (custom) κλάσης εξαίρεσης από το χρήστη

```
import math


class NegativeLogError(Exception):
    def __init__(self, value):
        self.value = value
        super().__init__(f"Logarithm undefined for x = {value}. x must be > 0.")

    def __str__(self):
        return (f"[NegativeLogError] Invalid input: {self.value}. Logarithm requires x > 0.")


def safe_log(x, base=10):
    if x <= 0:
        raise NegativeLogError(x)
    return math.log(x, base)


try:
    print(safe_log(100))  # log10(100) = 2
    print(safe_log(-5))  # θα γίνει raise η προσαρμοσμένη εξαίρεση
except NegativeLogError as e:
    print("Εξαίρεση: ", e)  # χρήση του __str__
    print("Η τιμή που προκάλεσε εξαίρεση ήταν: ", e.value)
```

```
2.0
Εξαίρεση: [NegativeLogError] Invalid input: -5. Logarithm requires x > 0.
Η τιμή που προκάλεσε εξαίρεση ήταν: -5
```

Άσκηση #8 (εξαιρέσεις): Εκφώνηση

- Γράψτε μια συνάρτηση read_numbers_and_divide() που ζητά από το χρήστη να εισάγει δύο αριθμούς σε μία γραμμή, χωρισμένους με κενό. Η συνάρτηση πρέπει να μετατρέπει τα δύο στοιχεία σε αριθμούς τύπου float και να επιστρέφει το πηλίκο του πρώτου διά του δεύτερου.
- Χρησιμοποιήστε μπλοκ try-except για να χειριστείτε τυχόν ValueError όταν τα δεδομένα δεν είναι αριθμοί και ZeroDivisionError όταν ο δεύτερος αριθμός είναι μηδέν, εμφανίζοντας κατάλληλα μηνύματα στον χρήστη
- Η συνάρτηση να συνεχίζει να ζητά είσοδο μέχρι ο χρήστης να δώσει έγκυρα δεδομένα.

Άσκηση #8: Λύση

```
def read_numbers_and_divide():
    while True:
        try:
            user_input = input("Εισάγετε 2 τιμές χωρισμένες με κενό: ").split()
            if len(user_input) != 2:
                raise ValueError("Παρακαλώ εισάγετε ακριβώς 2 τιμές.")
            a, b = map(float, user_input)  # μετατροπή σε float σε 1 εντολή
            return a / b

        except ValueError as e:
            print("Σφάλμα: ", e)
        except ZeroDivisionError:
            print("Σφάλμα: Η δεύτερη είσοδος δεν μπορεί να είναι μηδέν.")
```

```
quotient = read_numbers_and_divide()
print("Αποτέλεσμα: ", quotient)
```

```
Εισάγετε 2 τιμές χωρισμένες με κενό: 5 0
Σφάλμα: Η δεύτερη είσοδος δεν μπορεί να είναι μηδέν.
Εισάγετε 2 τιμές χωρισμένες με κενό: 1 2 3
Σφάλμα:  Παρακαλώ εισάγετε ακριβώς 2 τιμές.
Εισάγετε 2 τιμές χωρισμένες με κενό: 4 5
Αποτέλεσμα:  0.8
```

Κανονικές εκφράσεις

Regular Expressions

Τι είναι οι κανονικές εκφράσεις;

- Μια κανονική έκφραση είναι μια ακολουθία χαρακτήρων που ορίζει ένα μοτίβο αναζήτησης (search pattern) για ένα κείμενο
- Χρησιμοποιούνται για
  - Αναζήτηση μοτίβου σε κείμενο και εντοπισμό όλων των εμφανίσεών του
  - Έλεγχο αποδεκτής μορφής κειμένου που ορίζεται από ένα μοτίβο (π.χ., emails, τηλέφωνα, αριθμούς πιστωτικών καρτών)
  - Αντικατάσταση υποσυμβολοσειρών που ταιριάζουν με μοτίβο με άλλες σε κείμενο
  - Διαχωρισμό λεκτικών σε σημεία που ταιριάζουν με ένα μοτίβο
- Στην Python παρέχεται υποστήριξη για κανονικές εκφράσεις από το module re που είναι built-in, δηλαδή δεν χρειάζεται εγκατάσταση με το pip αλλά απλά γίνεται import για να χρησιμοποιηθεί
  - Ωστόσο υπάρχουν και βιβλιοθήκες τρίτων κατασκευαστών για κανονικές εκφράσεις (π.χ. regex, re2)

Το module re

- Οι βασικές συναρτήσεις του module re είναι οι ακόλουθες:
  - **re.match(pattern, text)** => ελέγχει αν ένα κείμενο (text) ξεκινά με ένα συγκεκριμένο μοτίβο (pattern)
  - **re.search(pattern, text)** => εντοπίζει την πρώτη εμφάνιση του μοτίβου στο κείμενο
  - **re.findall(pattern, text)** => δημιουργεί μια λίστα με όλες τις εμφανίσεις του μοτίβου στο κείμενο
  - **re.finditer(pattern, text)** => σαν το findall, αλλά επιστρέφει έναν iterator αντί για λίστα
  - **re.findall(pattern, rep, text)** => αντικαθιστά εμφανίσεις του μοτίβου με το rep
  - **re.split(pattern, text)** => διαχωρίζει ένα κείμενο χρησιμοποιώντας ένα μοτίβο ως διαχωριστικό

Συνήθη μοτίβα (1/4): βασικά σύμβολα

Μοτίβο	Περιγραφή	Παράδειγμα	Αποτέλεσμα
.	Οποιοσδήποτε χαρακτήρας (εκτός newline)	re.findall(r"a.c", "abc acb a-c a1c")	['abc', 'a-c', 'a1c']
^	Αρχή κειμένου	re.search(r"^Hello", "Hello World")	<re.Match object; span=(0, 5), match='Hello'>
$	Τέλος κειμένου	re.search(r"World$", "Hello World")	<re.Match object; span=(6, 11), match='World'>
\d	Ψηφίο	re.findall(r"\d", "A1B2")	['1', '2']
\D	Μη ψηφίο	re.findall(r"\D", "A1B2")	['A', 'B']
\w	Χαρακτήρας λέξης (γράμμα ή αριθμός ή κάτω παύλα)	re.findall(r"\w+", "Hi_123!")	['Hi_123']
\W	Όχι χαρακτήρας λέξης	re.findall(r"\W+", "Hi_123!")	['!']
\s	Κενό διάστημα	re.findall(r"\s", "Hello World")	[' ']
\S	Όχι κενό διάστημα	re.findall(r"\S+", "Hello World")	['Hello', 'World']

Συνήθη μοτίβα (2/4): ποσοδείκτες και ομάδες

Μοτίβο	Περιγραφή	Παράδειγμα	Αποτέλεσμα
*	Μηδέν ή περισσότερες εμφανίσεις	re.findall(r"a*", "aaab")	['aaa', '']
+	Μία ή περισσότερες εμφανίσεις	re.findall(r"a+", "aaab")	['aaa']
?	Μηδέν ή μία εμφάνιση	re.findall(r"a?", "aaab")	['a', 'a', 'a', '', '']
{m}	Ακριβώς m εμφανίσεις	re.findall(r"a{2}", "aaab")	['aa']
{m,n}	Από m έως n εμφανίσεις	re.findall(r"a{2,3}", "aaaa")	['aaa']
{m,}	Τουλάχιστον m εμφανίσεις	re.findall(r"a{2,}", "aaaa")	['aaaa']
(...)	Ομάδα για αναφορά	re.findall(r"(ab)+", "abab")	['ab']

Συνήθη μοτίβα (3/4): σύνολα χαρακτήρων

Μοτίβο	Περιγραφή	Παράδειγμα	Αποτέλεσμα
[abc]	Οποιοσδήποτε χαρακτήρας από αυτούς που βρίσκονται μέσα στις αγκύλες	re.findall(r"[aeiou]", "hello")	['e', 'o']
[^abc]	Οποιοσδήποτε χαρακτήρας εκτός από αυτούς που βρίσκονται μέσα στις αγκύλες	re.findall(r"[^0-9]", "A1B2")	['A', 'B']

Συνήθη μοτίβα (4/4): Όρια λέξεων

Μοτίβο	Περιγραφή	Παράδειγμα	Αποτέλεσμα
\b	Όριο λέξης	re.findall(r"\bword\b", "a word wordy")	['word']
\B	Μη όριο λέξης	re.findall(r"\Bend", "bend send")	['end', 'end']

Παράδειγμα με κανονικές εκφράσεις

- Δίνεται το ακόλουθο κείμενο που περιέχει ημερομηνίες: "Η συνάντηση θα γίνει στις 5/9/2025 και η επόμενη στις 12/10/2025. Μια άλλη επιλογή είναι οι συναντήσεις να γίνουν 1/10/2025 και 17/10/2025 αντίστοιχα."
- Ο κώδικας εντοπίζει με κανονικές εκφράσεις όλες τις ημερομηνίες στο κείμενο και τις εκτυπώνει

```
import re

text = """
Η συνάντηση θα γίνει στις 5/9/2025 και
η επόμενη στις 12/10/2025. Μια άλλη
επιλογή είναι οι συναντήσεις να γίνουν
1/10/2025 και 17/10/2025 αντίστοιχα.
"""

date_pattern = r"\b\d{1,2}/\d{1,2}/\d{4}\b"
matches = re.findall(date_pattern, text)

for i, match in enumerate(matches):
    print(f"Εμφάνιση ημερομηνίας {i+1}: {match}")
```

```
Εμφάνιση ημερομηνίας 1: 5/9/2025
Εμφάνιση ημερομηνίας 2: 12/10/2025
Εμφάνιση ημερομηνίας 3: 1/10/2025
Εμφάνιση ημερομηνίας 4: 17/10/2025
```

Λύση του προηγούμενου παραδείγματος με χρήση ομάδων (groups) στο μοτίβο

- Με τη χρήση ομάδων, μπορούμε να συλλάβουμε ξεχωριστά την ημέρα , τον μήνα και το έτος στις ημερομηνίες του κειμένου
- Έτσι η κανονική έκφραση αντί για: "\b\d{1,2}/\d{1,2}/\d{4}\b" γίνεται: r"\b(\d{1,2})/(\d{1,2})/(\d{4})\b"
- Προσέξτε τις έξτρα παρενθέσεις στη νέα κανονική έκφραση

```python
import re

text = """
Η συνάντηση θα γίνει στις 5/9/2025 και
η επόμενη στις 12/10/2025. Μια άλλη
επιλογή είναι οι συναντήσεις να γίνουν
1/10/2025 και 17/10/2025 αντίστοιχα.
"""

date_pattern = r"\b(\d{1,2})/(\d{1,2})/(\d{4})\b"

matches = re.findall(date_pattern, text)

for day, month, year in matches:
    print(f"Ημέρα: {day}, Μήνας: {month}, Έτος: {year}")
```

```
Ημέρα: 5, Μήνας: 9, Έτος: 2025
Ημέρα: 12, Μήνας: 10, Έτος: 2025
Ημέρα: 1, Μήνας: 10, Έτος: 2025
Ημέρα: 17, Μήνας: 10, Έτος: 2025
```

Άσκηση #9 (κανονικές εκφράσεις): Εκφώνηση

- Δίνεται το κείμενο δεξιά που περιέχει πράξεις πρόσθεσης και αφαίρεσης ακεραίων με μέχρι τρία ψηφία. Γράψτε πρόγραμμα που να εντοπίζει όλες τις πράξεις να τις εκτελεί και να εμφανίζει το αποτέλεσμα κάθε πράξης
- Χρησιμοποιήστε το regular expression: \b(\d{1,3})([+-])(\d{1,3})\b για να εντοπίσετε τις πράξεις

Κείμενο που θα χρησιμοποιηθεί στην άσκηση

Στο μάθημα αριθμητικής κάναμε τις παρακάτω πράξεις: 5+7, 123-45, 9+15, 250-100, 12+88, 99-50, 7+8, 300-150, 45+55, 200-75. Μετά κάναμε τις πράξεις 101+21, 15+85 και κάναμε μια τελευταία αφαίρεση 400-123 για επανάληψη.

Άσκηση #9: Λύση

```python
import re

text = """
Στο μάθημα αριθμητικής κάναμε τις παρακάτω πράξεις:
5+7, 123-45, 9+15, 250-100, 12+88, 99-50, 7+8,
300-150, 45+55, 200-75. Μετά κάναμε τις πράξεις
101+21, 15+85 και κάναμε μια τελευταία αφαίρεση
400-123 για επανάληψη.
"""

pattern = r"\b(\d{1,3})([+-])(\d{1,3})\b"

matches = re.findall(pattern, text)
for a, op, b in matches:
    a, b = int(a), int(b)
    result = a + b if op == "+" else a - b
    print(f"{a}{op}{b} = {result}")
```

```
5+7 = 12
123-45 = 78
9+15 = 24
250-100 = 150
12+88 = 100
99-50 = 49
7+8 = 15
300-150 = 150
45+55 = 100
200-75 = 125
101+21 = 122
15+85 = 100
400-123 = 277
```