Τελευταία ενημέρωση 17/10/2025

Python – Μέρος Α
ΕΚΔΔΑ

7.1 Εισαγωγή στον αντικειμενοστραφή προγραμματισμό με την Python

Εβδομάδα 7/7

Οκτώβριος 2025

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

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

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

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 #1 (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 #1: 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__

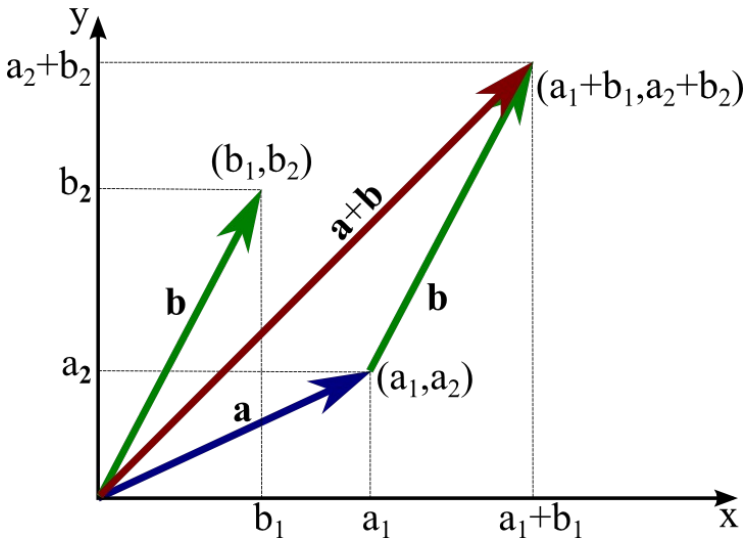
```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    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)}
```

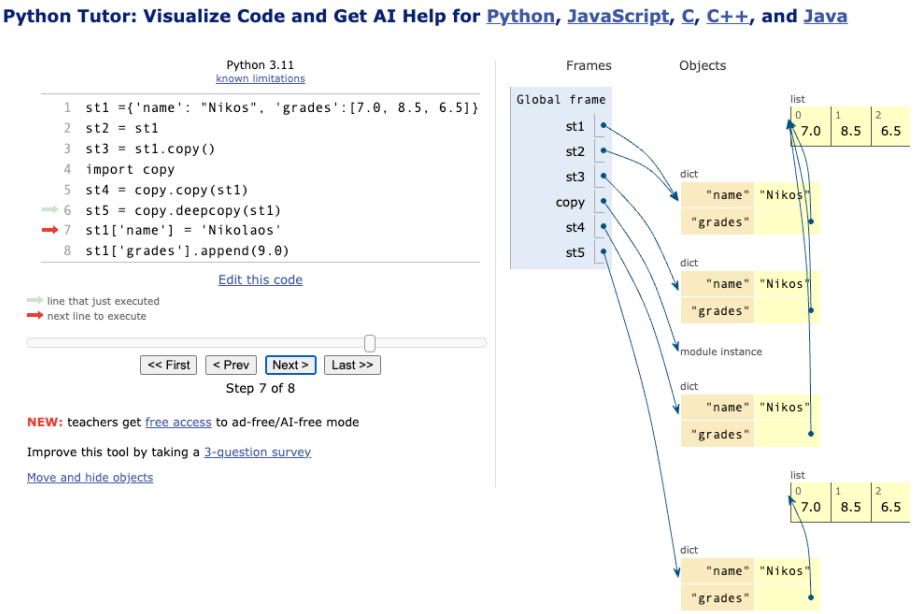
Αντιγραφή αντικειμένων

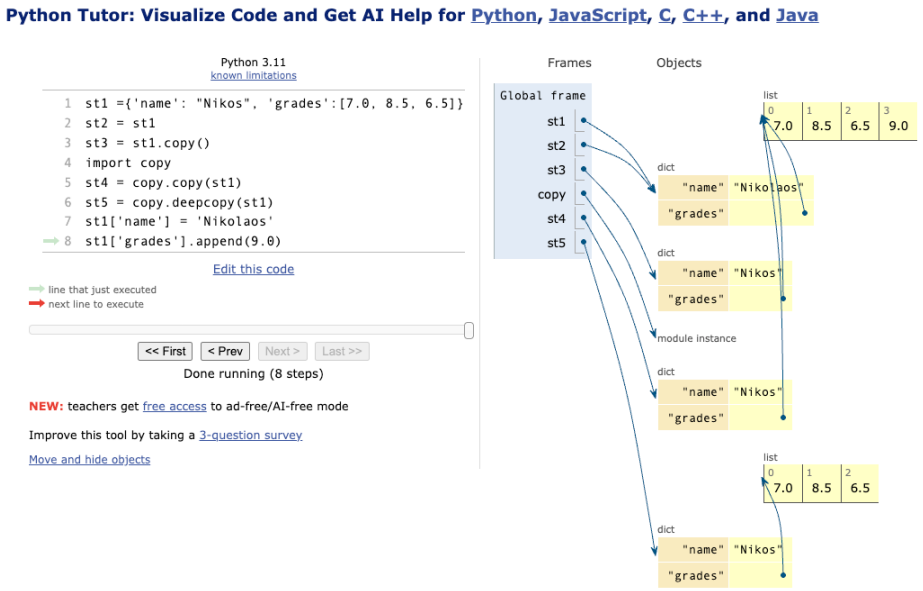
- Η αντιγραφή αντικειμένων είναι ένα θέμα που απαιτεί προσοχή
- Η απλή ανάθεση μιας μεταβλητής που είναι αναφορά σε ένα αντικείμενο σε μια άλλη μεταβλητή, δεν κάνει αντιγραφή, αλλά δύο αναφορές προς το ίδιο αντικείμενο
- Για να γίνει πραγματική αντιγραφή πρέπει να χρησιμοποιηθεί το module copy
- Η συνάρτηση 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 #2 (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 #2: 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)

```
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 #3 (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 #3: Λύση

```
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