12. ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ - ΚΛΑΣΕΙΣ

12.0 Λύσεις προηγούμενων ασκήσεων

Ασκήσεις (lambda_ex1_lesson12.py)

1. Γράψτε ένα πρόγραμμα για να δημιουργήσετε μια συνάρτηση λάμδα που προσθέτει τον αριθμό 15 σε έναν αριθμό που μεταβιβάζεται ως όρισμα, αφού ζητηθεί από τον χρήστη. Δημιουργήστε επίσης μια συνάρτηση λάμδα που πολλαπλασιάζει το όρισμα x με το όρισμα y (τα ζητάει και τα δύο από τον χρήστη) και εκτυπώνει το αποτέλεσμα.

Λύση

```
a = int(input("Δώστε έναν αριθμό: "))
r = lambda a : a + 15
print(r(a))
x = int(input("Δώστε έναν αριθμό: "))
y = int(input("Δώστε ακόμα έναν: "))
r = lambda x, y : x * y
print(r(x, y))
```

Άσκηση 2 (lambda_ex2_lesson12.py)

Γράψτε ένα πρόγραμμα το οποίο υψώνει εις το τετράγωνο κάθε αριθμό μιας δεδομένης λίστας ακεραίων. Κατόπιν υψώνει εις τον κύβο τους ίδιους αριθμούς. Χρησιμοποιήστε lambda function καθώς και τη συνάρτηση map(), σαν όρισμα της list() (list(map(lambda….).

Τυπώστε μία-μία σε ξεχωριστές γραμμές την κάθε λίστα

Η αρχική λίστα των ακεραίων είναι η: nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Λύση

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Αρχική λίστα ακεραίων:")
print(nums)
print("\nΎψωση κάθε αριθμού της λίστας εις το τετράγωνο:")
square_nums = list(map(lambda x: x ** 2, nums))
print(square_nums)
print("\nΎψωση κάθε αριθμού της λίστας εις τον κύβο:")
cube_nums = list(map(lambda x: x ** 3, nums))
print(cube_nums)
```

Άσκηση 3 (lambda_ex3_lesson12.py)

Γράψτε ένα πρόγραμμα για να μετρήσετε τους άρτιους και τους περιττούς αριθμούς σε μια δεδομένη λίστα ακεραίων χρησιμοποιώντας τις lambda functions.

Αρχικός πίνακας:

[1, 2, 3, 5, 7, 8, 9, 10]

Η εκτύπωση πρέπει να είναι:

Αριθμός ζυγών αριθμών στην παραπάνω λίστα: 3

Αριθμός περιττών αριθμών στην παραπάνω λίστα: 5

Σημείωση: Χρησιμοποιήστε τις συναρτήσεις len(list(filter(lambda….

Λύση

```
array_nums = [1, 2, 3, 5, 7, 8, 9, 10]
print("Αρχική λίστα:")
```

```
print(array_nums)
odd_ctr = len(list(filter(lambda x: (x%2 != 0) , array_nums)))
even_ctr = len(list(filter(lambda x: (x%2 == 0) , array_nums)))
print("\nΑριθμός περιττών αριθμών στην παραπάνω λίστα: ",
even_ctr)
print("\nΑριθμός περιττών αριθμών στην παραπάνω λίστα: ", odd_ctr)
```

Άσκηση 4 (lambda_ex4_lesson12.py)

Γράψτε μια συνάρτηση η οποία υπολογίζει το Μέγιστο Κοινό Διαιρέτη δύο δεδομένων αριθμών. Χρησιμοποιήστε τα ζευγάρια αριθμών 12 και 17, 4 και 6, 336 και 360

Λύση

```
def gcd(x, y):
  gcd = 1
  if x % y == 0:
     return y
  for k in range(int(y / 2), 0, -1):
     if x % k == 0 and y % k == 0:
        gcd = k
        break
  return gcd
print("Ο ΜΚΔ των 12 & 17 είναι ο",gcd(12, 17))
print("Ο ΜΚΔ των 4 & 6 είναι ο",gcd(4, 6))
print("Ο ΜΚΔ των 336 & 360 είναι ο",gcd(336, 360))
```

12.1 Ορισμός και έννοιες

Ο αντικειμενοστραφής προγραμματισμός είναι μια παραδοχή ή μεθοδολογία προγραμματισμού που επιτρέπει την οργάνωση κώδικα σε αυτόνομες μονάδες που ονομάζονται "αντικείμενα". Ένα αντικείμενο είναι ένα πρότυπο ή μια "εκδοχή", ή ένα στιγμιότυπο μιας κλάσης και περιέχει δεδομένα (μεταβλητές) και μεθόδους (συναρτήσεις) που δρουν πάνω σε αυτά τα δεδομένα.

12.1.2 Πλεονεκτήματα του αντικειμενοστραφούς προγραμματισμού

* Ανακλαστικότητα (Reflection): Η δυνατότητα ενός αντικειμένου να "κατανοεί" τον εαυτό του και τις ιδιότητές του.

* Επαναχρησιμοποίηση κώδικα: Η δυνατότητα να δημιουργούμε κλάσεις και αντικείμενα που μπορούν να επαναχρησιμοποιηθούν σε διάφορα μέρη του προγράμματος.

* Κληρονομικότητα (Inheritance): Η δυνατότητα μιας κλάσης να κληρονομεί τις ιδιότητες και τις συμπεριφορές μιας άλλης κλάσης.

* Πολυμορφισμός (Polymorphism): Η δυνατότητα μιας μεθόδου να έχει διαφορετική συμπεριφορά, ανάλογα με τον τύπο του αντικειμένου που την καλεί.

12.2 Κλάσεις

2.1 Ορισμός μιας κλάσης

Μια κλάση είναι ένα πρότυπο ή ένα καλούπι για τη δημιουργία αντικειμένων. Ορίζουμε τις μεταβλητές και τις συναρτήσεις που ανήκουν σε αυτήν την κλάση.

2.2 Αναγνώριση κλάσεων και αντικειμένων

Μια κλάση αναγνωρίζεται από το όνομά της, ενώ ένα αντικείμενο είναι μια συγκεκριμένη εκδοχή (ή ένα στιγμιότυπο) της κλάσης. Ένα αντικείμενο δημιουργείται από μια κλάση και έχει τη δυνατότητα να προσπελαστεί και να τροποποιηθεί.

2.3 Χαρακτηριστικά και συμπεριφορά μιας κλάσης

Ένα χαρακτηριστικό μιας κλάσης είναι μια μεταβλητή που αποθηκεύει δεδομένα για το αντικείμενο. Μια συμπεριφορά μιας κλάσης είναι μια συνάρτηση που εκτελεί ενέργειες πάνω στα δεδομένα του αντικειμένου.

2.4 Ορίζοντας μια κλάση στην Python

Για να ορίσουμε μια κλάση στην Python, χρησιμοποιούμε τη λέξη-κλειδί "class" ακολουθούμενη από το όνομα της κλάσης. Συνήθως, ορίζουμε τα χαρακτηριστικά και τις συμπεριφορές μέσα σε έναν κατασκευαστή (constructor) και άλλες μεθόδους μέσα στην κλάση.

12.3: Δημιουργία δικής μας κλάσης

3.1 Ορισμός της κλάσης

Για να δημιουργήσουμε μια κλάση, χρησιμοποιούμε τη λέξη-κλειδί "class" ακολουθούμενη από το όνομα της κλάσης. Μέσα στην κλάση ορίζουμε τα χαρακτηριστικά και τις μεθόδους που θα έχει η κλάση.

3.2 Συναρτήσεις μέλη

Οι συναρτήσεις μέλη είναι μέθοδοι που ανήκουν σε μια κλάση. Αυτές οι μέθοδοι μπορούν να εκτελέσουν ενέργειες σε αντικείμενα της κλάσης και να αλληλεπιδρούν με τα χαρακτηριστικά τους.

3.3 Μεταβλητές μέλη

Οι μεταβλητές μέλη είναι μεταβλητές που ανήκουν σε μια κλάση. Αυτές οι μεταβλητές αποθηκεύουν δεδομένα για τα αντικείμενα της κλάσης και μπορούν να προσπελαστούν και να τροποποιηθούν από τις μεθόδους της κλάσης.

3.4 Αρχικοποίηση αντικειμένων

Η αρχικοποίηση αντικειμένων γίνεται μέσω του κατασκευαστή (constructor) της κλάσης. Ο κατασκευαστής είναι μια ειδική μέθοδος που καλείται κατά τη δημιουργία ενός αντικειμένου και χρησιμοποιείται για την αρχικοποίηση των μεταβλητών μελών.

3.5 Πρόσβαση σε μέλη κλάσης

Για να προσπελάσουμε τα μέλη μιας κλάσης, χρησιμοποιούμε τη σύνταξη "αντικείμενο.μέλος". Μπορούμε να αναφερθούμε στα χαρακτηριστικά ή να καλέσουμε τις μεθόδους του αντικειμένου.

12.4: Παραδείγματα και εφαρμογές

4.1 Παράδειγμα 1: Κλάση Car

Ας δημιουργήσουμε μια κλάση με όνομα "Car" που θα αναπαριστά ένα αυτοκίνητο. Ορίζουμε τα χαρακτηριστικά όπως το μοντέλο (model), το χρώμα (color) και την ταχύτητα (speed), καθώς και μεθόδους όπως την επιτάχυνση (accelerate) και το φρενάρισμα (brake).

```
# Ορισμός της κλάσης Car
class Car:
    def __init__(self, model, color):
        # Χαρακτηριστικά της κλάσης Car
        self.model = model
        self.color = color
        self.speed = 0

    def accelerate(self, increment):
        # Μέθοδος για επιτάχυνση του αυτοκινήτου
        self.speed += increment

    def brake(self, decrement):
        # Μέθοδος για φρενάρισμα του αυτοκινήτου
        if self.speed >= decrement:
            self.speed -= decrement
        else:
            self.speed = 0
```

4.2 Παράδειγμα 2: Κλάση φοιτητής

Ας δημιουργήσουμε μια κλάση με όνομα "Student" που θα αναπαριστά ένα φοιτητή. Ορίζουμε τα χαρακτηριστικά όπως το όνομα, το επώνυμο και τον βαθμό, καθώς και μεθόδους όπως ο υπολογισμός του μέσου όρου και η εκτύπωση των στοιχείων του φοιτητή.

```
# Ορισμός της κλάσης Student
class Student:
    def __init__(self, name, surname):
        # Χαρακτηριστικά της κλάσης Student
        self.name = name
        self.surname = surname
        self.grade = 0

    def calculate_average(self, grades):
        # Μέθοδος για υπολογισμό μέσου όρου βαθμολογίας
        total = sum(grades)
        self.grade = total / len(grades)

    def print_info(self):
        # Μέθοδος για εκτύπωση πληροφοριών φοιτητή
        print("Όνομα:", self.name)
        print("Επώνυμο:", self.surname)
        print("Βαθμός:", self.grade)
```

Ενότητα 5.1: Εισαγωγή στην κληρονομικότητα

Η κληρονομικότητα αποτελεί ένα σημαντικό στοιχείο του αντικειμενοστραφούς προγραμματισμού. Μέσω της κληρονομικότητας, μπορούμε να οργανώσουμε τις κλάσεις σε ιεραρχίες, επιτρέποντας την κοινή χρήση χαρακτηριστικών και μεθόδων ανάμεσα σε αυτές.

Η ιεραρχία κλάσεων αποτελείται από μια γονική κλάση και μία ή περισσότερες υποκλάσεις. Η γονική κλάση είναι η κλάση από την οποία κληρονομούνται τα χαρακτηριστικά και οι μέθοδοι, ενώ οι υποκλάσεις είναι οι κλάσεις που κληρονομούν τα χαρακτηριστικά και τις μεθόδους από τη γονική κλάση.

Ας δούμε ένα παράδειγμα κώδικα που περιγράφει την κληρονομικότητα:

```
# Δημιουργία της γονικής κλάσης
class Vehicle:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def information(self):
        print("Make:", self.make)
        print("Model:", self.model)

# Δημιουργία της υποκλάσης
class Car(Vehicle):
    def __init__(self, make, model, displacement):
        # Κλήση του __init__ της γονικής κλάσης
```

```
        super().__init__(make, model)
        self.displacement = displacement

    def information(self):
        # Κλήση της μεθόδου της γονικής κλάσης
        super().information()
        print("Displacement:", self.displacement)

# Δημιουργία αντικειμένων
Car1 = Car("Ford", "Focus", 1600)
Car2 = Car("Toyota", "Corolla", 1800)

# Κλήση μεθόδων αντικειμένων
Car1.information()
Car2.information()
```

Στο παραπάνω παράδειγμα, η κλάση "Vehicle" είναι η γονική κλάση και περιέχει το χαρακτηριστικό "make" και τη μέθοδο "information". Η κλάση "Car" είναι μια υποκλάση της "Vehicle" και κληρονομεί το χαρακτηριστικό και τη μέθοδο από τη γονική κλάση. Η υποκλάση επίσης έχει ένα δικό της χαρακτηριστικό "displacement" και μια μέθοδο "information" που επεκτείνει τη μέθοδο της γονικής κλάσης.

Όταν δημιουργούμε αντικείμενα της υποκλάσης "Car", μπορούμε να καλούμε τη μέθοδο "information" τόσο από τη γονική κλάση όσο και από την υποκλάση. Η μέθοδος "super()" χρησιμοποιείται για να κληθεί η μέθοδος της γονικής κλάσης. Με αυτόν τον τρόπο, μπορούμε να επεκτείνουμε τη λειτουργικότητα της γονικής κλάσης στην υποκλάση.

Εκτέλεση του παραπάνω κώδικα θα παράγει την ακόλουθη έξοδο:

Make: Ford

Model: Focus

Displacement: 1600

Make: Toyota

Model: Corolla

Displacement: 1800

Όπως φαίνεται, οι μέθοδοι τόσο της γονικής κλάσης όσο και της υποκλάσης καλούνται και παρέχουν τις αντίστοιχες πληροφορίες (information) για τα αυτοκίνητα.

Έτσι, η κληρονομικότητα μας επιτρέπει να δημιουργήσουμε ιεραρχίες κλάσεων και να κληρονομήσουμε τα χαρακτηριστικά και τις μεθόδους των γονικών κλάσεων στις υποκλάσεις, επεκτείνοντας και προσαρμόζοντας τη λειτουργικότητα σύμφωνα με τις ανάγκες.

6.1 Παραδείγματα κληρονομικότητας και πολυμορφισμού:

Μια κλάση είναι ένα καλούπι από το οποίο μπορούμε να δημιουργήσουμε αντικείμενα. Η κληρονομικότητα επιτρέπει τη δημιουργία νέων κλάσεων που βασίζονται σε ήδη υπάρχουσες κλάσεις, επαυξάνοντας ή τροποποιώντας τη λειτουργικότητά τους.

Για να ορίσουμε μια κλάση, χρησιμοποιούμε τη λέξη-κλειδί "class" και το όνομα της κλάσης, ακολουθούμενο από δύο άνω και κάτω τελείες. Εδώ έχουμε ένα παράδειγμα μιας κλάσης "Person":

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

    def greet(self):
        print("Γεια σας, είμαι ο", self.name)
```

Στο παραπάνω παράδειγμα, η κλάση "Person" έχει έναν constructor (__init__) που δέχεται το όνομα του ατόμου και αποθηκεύει αυτό το όνομα στη μεταβλητή "name". Επίσης, η κλάση έχει μια μέθοδο "greet" που εκτυπώνει ένα μήνυμα χαιρετισμού με το όνομα του ατόμου.

Για να δημιουργήσουμε ένα αντικείμενο από μια κλάση, χρησιμοποιούμε τη σύνταξη "όνομα = Κλάση()" με τυχόν παραμέτρους. Παρακάτω έχουμε ένα παράδειγμα χρήσης της κλάσης "Person":

```
person = Person("Γιάννης")
person.greet()
```

Το παραπάνω κομμάτι κώδικα δημιουργεί ένα νέο αντικείμενο "person" από την κλάση "Person" και καλεί τη μέθοδο "greet". Έτσι, εκτυπώνεται το μήνυμα χαιρετισμού με το όνομα "Γιάννης".

Συμπεράσματα και επόμενα βήματα:

Η κληρονομικότητα μας επιτρέπει να δημιουργούμε νέες κλάσεις βασισμένες σε ήδη υπάρχουσες κλάσεις, επεκτείνοντας ή τροποποιώντας τη λειτουργικότητά τους. Επίσης, είδαμε πως να ορίζουμε και να χρησιμοποιούμε μια κλάση στην Python.

Άσκηση 1 (class_rectangle_12less.py):

Δημιουργήστε μια γονική κλάση "Shape" με μια μέθοδο "area" που επιστρέφει το εμβαδόν ενός σχήματος. Στη συνέχεια, δημιουργήστε δύο παιδικές (child) κλάσεις "Rectangle" και "Circle" που κληρονομούν από την κλάση "Shape" και υλοποιούν τη μέθοδο "area" με τον κατάλληλο τρόπο για κάθε σχήμα.

```
class Shape:
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius * self.radius

# Αυτό που λείπει τώρα από τον κώδικά μας είναι η δημιουργία
#στιγμιοτύπων των κλάσεών μας και ο υπολογισμός των εμβαδών

# Δημιουργία παραλληλόγραμμου και υπολογισμός εμβαδού
```

```
rectangle = Rectangle(5, 10)
rectangle_area = rectangle.area()
print("Το εμβαδόν του παραλληλόγραμμου είναι:", rectangle_area)

# Δημιουργία κύκλου και υπολογισμός εμβαδού
circle = Circle(3)
circle_area = circle.area()
print("Το εμβαδόν του κύκλου είναι:", circle_area)
```

Η έξοδος είναι:

Το εμβαδόν του παραλληλόγραμμου είναι: 50

Το εμβαδόν του κύκλου είναι: 28.259999999999998

Άσκηση 2 (class_animal_12less.py):

Δημιουργήστε μια γονική κλάση "Animal" με μια μέθοδο "sound" που επιστρέφει τον ήχο που βγάζει το ζώο. Στη συνέχεια, δημιουργήστε τρεις παιδικές (child) κλάσεις "Dog", "Cat" και "Cow" που κληρονομούν από την κλάση "Animal" και υλοποιούν τη μέθοδο "sound" με τον κατάλληλο τρόπο για κάθε ζώο.

```
class Animal:
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Γαβ"
```

```
class Cat(Animal):
    def sound(self):
        return "Μιαου"

class Cow(Animal):
    def sound(self):
        return "Μου"

# Δημιουργία στιγμιοτύπων και κλήση μεθόδων
dog = Dog()
print(dog.sound())  # Εκτυπώνει "Γαβ"

cat = Cat()
print(cat.sound())  # Εκτυπώνει "Μιαου"

cow = Cow()
print(cow.sound())  # Εκτυπώνει "Μου"
```

Σε αυτήν την άσκηση, όπως και στην προηγούμενη, χρησιμοποιούμε την έννοια του πολυμορφισμού.

Η γονική κλάση "Animal" ορίζει μια μέθοδο με το όνομα "sound" (ήχος) που δεν κάνει τίποτα. Οι παιδικές κλάσεις "Dog", "Cat" και "Cow" κληρονομούν από την κλάση "Animal" και υλοποιούν τη μέθοδο "sound" με διαφορετικό τρόπο για κάθε ζώο.

Αυτό σημαίνει ότι κάθε ζώο μπορεί να επιστρέψει τον δικό του ήχο.

Στη συνέχεια, δημιουργούμε αντικείμενα για κάθε ζώο και εκτυπώνουμε τον ήχο που βγάζει κάθε ζώο. Κάθε φορά που καλούμε τη μέθοδο "sound" σε ένα αντικείμενο, επιστρέφεται ο σωστός ήχος για το εκάστοτε ζώο.

Έτσι, χρησιμοποιώντας την πολυμορφική φύση της Python, μπορούμε να δημιουργήσουμε κλάσεις που κληρονομούν από μια γονική κλάση και υλοποιούν τις μεθόδους με τρόπο που είναι κατάλληλο για κάθε παιδική κλάση. Αυτό μας επιτρέπει να δημιουργήσουμε ευέλικτο και επαναχρησιμοποιήσιμο κώδικα.

Άσκηση 3 (class_vehicle_12less.py):

Δημιουργήστε μια γονική κλάση "Vehicle" με μια μέθοδο "drive" που εκτυπώνει ένα μήνυμα. Στη συνέχεια, δημιουργήστε δύο παιδικές κλάσεις "Car" και "Motorcycle" που κληρονομούν από την κλάση "Vehicle" και υλοποιούν τη μέθοδο "drive" με τον κατάλληλο τρόπο για κάθε όχημα.

```
class Vehicle:
    def drive(self):
        pass

class Car(Vehicle):
    def drive(self):
        print("Οδηγώ το αυτοκίνητο")

class Motorcycle(Vehicle):
    def drive(self):
        print("Οδηγώ τη μοτοσυκλέτα")
```

```
# Δημιουργία στιγμιοτύπων και κλήση μεθόδων
car = Car()
car.drive()  # Εκτυπώνει "Οδηγώ το αυτοκίνητο"

motorcycle = Motorcycle()
motorcycle.drive()  # Εκτυπώνει "Οδηγώ τη μοτοσυκλέτα"
```

7.1 Απλά παραδείγματα κατανόησης

Παράδειγμα 1 (class_StudentMarks_less12.py):

Γράψτε ένα πρόγραμμα για να δημιουργήσετε δύο κενές κλάσεις, Student και Marks. Τώρα δημιουργήστε μερικά στιγμιότυπα και ελέγξτε αν είναι στιγμιότυπα των εν λόγω κλάσεων ή όχι. Επίσης, ελέγξτε εάν οι εν λόγω κλάσεις είναι υποκλάσεις της ενσωματωμένης (built-in) κλάσης object ή όχι (θα χρησιμοποιήσουμε τις συναρτήσεις isistance() και issubclass() για τους παραπάνω ελέγχους.

```
class Student:
    pass
class Marks:
    pass
student1 = Student()
marks1 = Marks()
print(isinstance(student1, Student))
print(isinstance(marks1, Student))
print(isinstance(marks1, Marks))
print(isinstance(student1, Marks))
```

Έξοδος:

True

False

True

False

Παράδειγμα 2 (class_IOstring_less12.py):

Γράψτε μια κλάση που έχει δύο μεθόδους: get_String και print_String . Η get_String δέχεται μια συμβολοσειρά από τον χρήστη και η print_String εκτυπώνει τη συμβολοσειρά με κεφαλαία.

```
class IOString():
    def __init__(self):
        self.str1 = ""

    def get_String(self):
        self.str1 = input()

    def print_String(self):
        print(self.str1.upper())

str1 = IOString()
str1.get_String()
str1.print_String()
```

Παράδειγμα εξόδου:

fsdfasdf

FSDFASDF