13. Κλασεις ΙΙ – παραδειγματα και εφαρμογες

13.0 Λύσεις προηγούμενων ασκήσεων

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)
```

Έξοδος:

```
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 "Νιαου"

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

# Παραδείγματα χρήσης
dog = Dog()
print(dog.sound())  # Εκτυπώνει "Γαβ"

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

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

Έξοδος:
Γαβ
Νιαου
Μουου

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")

# Παραδείγματα χρήσης
car = Car()
car.drive()  # Εκτυπώνει "Οδηγώ το αυτοκίνητο"

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

Έξοδος:
Οδηγώ το αυτοκίνητο
Οδηγώ τη μοτοσυκλέτα

13.1.1 Μελέτη των κλάσεων και της κληρονομικότητας

Για να προχωράμε την κατανόησή μας σε σχέση με τις κλάσεις, μπορούμε να πούμε πως κλάση είναι ένα πρότυπο το οποίο περιλαμβάνει μεταβλητές και μεθόδους που σχετίζονται μεταξύ τους.

Ας δούμε ένα παράδειγμα του πραγματικού κόσμου, στο οποίο μπορούμε να χρησιμοποιήσουμε μια και μόνο κλάση. Θα δημιουργήσουμε ένα πρόγραμμα το οποίο να εξυπηρετεί ένα εστιατόριο. Θα έχει μια κλάση MyRestaurant με μεταβλητές (χαρακτηριστικά) όπως menu_items, book_table και customer_orders και μεθόδους όπως add_item_to_menu, book_tables και customer_order. Αφού δημιουργήσουμε την κλάση κι ένα στιγμιότυπό της, θα πραγματοποιήσουμε τα παρακάτω:

Προσθήκη στοιχείων στο μενού

Κράτηση τραπεζιού

Παραγγελιοληψία

Εκτύπωση του μενού

Εκτύπωση των κρατήσεων των τραπεζιών

Εκτύπωση των παραγγελιών

Θα χρησιμοποιήσουμε λεξικά και λίστες για να αποθηκεύσουμε τα δεδομένα:

'''
Θα δημιουργήσουμε ένα πρόγραμμα το οποίο να εξυπηρετεί ένα εστιατόριο.
Θα έχει μια κλάση MyRestaurant με μεταβλητές (χαρακτηριστικά) όπως menu_items, book_table και customer_orders και μεθόδους όπως add_item_to_menu, book_tables και customer_order.
Αφού δημιουργήσουμε την κλάση κι ένα στιγμιότυπό της, θα πραγματοποιήσουμε
τα παρακάτω:
Προσθήκη στοιχείων στο μενού
Κράτηση τραπεζιού
Παραγγελιοληψία
Εκτύπωση του μενού
Εκτύπωση των κρατήσεων των τραπεζιών
Εκτύπωση των παραγγελιών
Θα χρησιμοποιήσουμε λεξικά και λίστες για να αποθηκεύσουμε τα δεδομένα:
'''

```
class Restaurant:
    def __init__(self):
        self.menu_items = {}
        self.book_table = []
        self.customer_orders = []

    def add_item_to_menu(self, item, price):
        self.menu_items[item] = price
```

```
    def book_tables(self, table_number):
        self.book_table.append(table_number)

    def customer_order(self, table_number, order):
        order_details = {'table_number': table_number, 'order': order}
        self.customer_orders.append(order_details)

    def print_menu_items(self):
        for item, price in self.menu_items.items():
            print("{}: {}".format(item, price))

    def print_table_reservations(self):
        for table in self.book_table:
            print("Τραπέζι {}".format(table))

    def print_customer_orders(self):
        for order in self.customer_orders:
            print("Τραπέζι {}: {}".format(order['table_number'],
order['order']))

restaurant = Restaurant()

# Προσθήκη στοιχείων στο μενού
restaurant.add_item_to_menu("Αστακομακαρονάδα", 9.99)
restaurant.add_item_to_menu("Σαλάτα Caesar's:", 8)
restaurant.add_item_to_menu("Φιλέτο σολωμού:", 19.99)
restaurant.add_item_to_menu("Τηγανητές πατάτες:", 3.99)
restaurant.add_item_to_menu("Τσιπούρα:", 15)

# Κράτηση τραπεζιού
restaurant.book_tables(1)
restaurant.book_tables(2)
restaurant.book_tables(3)

# Παραγγελίες
restaurant.customer_order(1, "Αστακομακαρονάδα")
restaurant.customer_order(1, "Φιλέτο σολωμού")
restaurant.customer_order(2, "Τσιπούρα")
restaurant.customer_order(2, "Σαλάτα Caesar's")
```

```
print("\nΔιάσημα πιάτα με τις τιμές τους:")
restaurant.print_menu_items()
print("\nΚρατημένα τραπέζια στο εστιατόριό μας:")
restaurant.print_table_reservations()
print("\nΕκτύπωση παραγγελιών:")
restaurant.print_customer_orders()
```

Έξοδος:

```
Διάσημα πιάτα με τις τιμές τους:
Αστακομακαρονάδα: 9.99
Σαλάτα Caesar's:: 8
Φιλέτο σολωμού:: 19.99
Τηγανητές πατάτες:: 3.99
Τσιπούρα:: 15

Κρατημένα τραπέζια στο εστιατόριό μας:
Τραπέζι 1
Τραπέζι 2
Τραπέζι 3

Εκτύπωση παραγγελιών:
Τραπέζι 1: Αστακομακαρονάδα
Τραπέζι 1: Φιλέτο σολωμού
Τραπέζι 2: Τσιπούρα
Τραπέζι 2: Σαλάτα Caesar's
```

Παράδειγμα 2

Συνεχίζουμε με το παρακάτω παράδειγμα για την κατανόηση των κλάσεων.
Ας υποθέσουμε ότι έχουμε μια κλάση που ονομάζεται Staff. Αυτή η κλάση μπορεί να χρησιμοποιηθεί ώστε να αποθηκεύει όλες τις σχετικές πληροφορίες σχετικά με το προσωπικό σε μια εταιρεία. Μέσα στην κλάση, μπορούμε να δηλώσουμε δύο μεταβλητές για να αποθηκεύσουμε το όνομα και τη θέση του υπαλλήλου. Επιπλέον, μπορούμε επίσης να κωδικοποιήσουμε μια μέθοδο που ονομάζεται calculPay() για να υπολογίζουμε τον μισθό του προσωπικού.
Ας δούμε πώς να το κάνουμε αυτό.

Δημιουργούμε ένα νέο αρχείο στον IDLE και προσθέτουμε τον παρακάτω κώδικα:

```
class Staff:
   def __init__ (self, pPosition, pName, pPay):
      self.position = pPosition
      self.name = pName
      self.pay = pPay
      print('Δημιουργία του αντικειμένου "Προσωπικό"')

   def __str__(self):
      return "Θέση = %s, Name = %s, Pay = %d" % (self.position,
self.name, self.pay)

   def calculatePay(self):
      prompt = '\nΕισάγετε τις ώρες εργασίας σας %s: ' % (self.name)
      hours = input(prompt)
      prompt = 'Εισάγετε το ωρομίσθιό σας %s: ' %(self.name)
      hourlyRate = input(prompt)
      self.pay = int(hours)*int(hourlyRate)
      return self.pay
```

Στον παραπάνω κώδικα, πρώτα ορίζουμε μια κλάση με το όνομα Staff γράφοντας class Staff:
Στη συνέχεια, ορίζουμε μια ειδική μέθοδο με το όνομα __init__ για την κλάση. Αυτή ονομάζεται αρχικοποιητής της κλάσης. Πάντα την ονομάζουμε init με δύο κάτω παύλες μπροστά και πίσω.

Η Python διαθέτει ένα μεγάλο αριθμό ειδικών μεθόδων. Όλες οι ειδικές μέθοδοι έχουν δύο κάτω παύλες μπροστά και πίσω από τα ονόματά τους.

Θα συζητήσουμε τις ειδικές μεθόδους αργότερα στο μάθημά μας.

Ένας αρχικοποιητής καλείται κάθε φορά που δημιουργείται ένα αντικείμενο της κλάσης. Για να αρχικοποιήσει τις μεταβλητές (δηλαδή να τους δώσει αρχικές τιμές) στην κλάση.
Στην κλάση μας, έχουμε τρεις μεταβλητές: position, name και pay.

Αυτές οι μεταβλητές ονομάζονται μεταβλητές στιγμιοτύπων, αντίθετα από τις τοπικές μεταβλητές και τις μεταβλητές κλάσης. Οι μεταβλητές στιγμιοτύπων είναι μεταβλητές που προηγούνται από μια λέξη-κλειδί self.
Θα διερευνήσουμε την έννοια του self λίγο αργότερα. Για τώρα, απλά ας γνωρίζουμε ότι όταν θέλουμε να αναφερθούμε σε μεταβλητές στιγμιοτύπου στην κλάση, πρέπει να προσθέσουμε το self μπροστά από τα ονόματα μεταβλητών. Επιπλέον, οι περισσότερες μέθοδοι σε μια κλάση έχουν το self ως πρώτη παράμετρο.
Οι τρεις παρακάτω δηλώσεις αναθέτουν τις τρεις παραμέτρους της μεθόδου __init__
(pPosition, pName και pPay) στις μεταβλητές στιγμιοτύπου για να τις αρχικοποιήσουν.

```
self.position = pPosition
self.name = pName
self.pay = pPay
```

Μετά την αρχικοποίηση των τριών μεταβλητών στιγμιοτύπου, εκτυπώνουμε ένα απλό μήνυμα: 'Δημιουργία του αντικειμένου ‘Προσωπικό’'. Αυτό είναι όλο όσο αφορά τον αρχικοποιητή.

Το να γράψουμε έναν αρχικοποιητή είναι προαιρετικό αν δεν επιθυμούμε να αρχικοποιήσουμε τις μεταβλητές στιγμιοτύπου κατά τη δημιουργία του αντικειμένου. Μπορούμε πάντα να τις αρχικοποιήσουμε αργότερα.

Ας προχωρήσουμε στην επόμενη μέθοδο - __str__.
Η __str__ είναι μια άλλη ειδική μέθοδος που συνήθως περιλαμβάνεται όταν κατασκευάζουμε μια κλάση. Τη χρησιμοποιούμε για να επιστρέψουμε μια αναγνώσιμη από άνθρωπο συμβολοσειρά που αναπαριστά την κλάση.
Στο παράδειγμά μας, απλά επιστρέφουμε μια συμβολοσειρά που παρέχει τις τιμές των τριών μεταβλητών. Θα δούμε πώς χρησιμοποιούμε αυτήν τη μέθοδο αργότερα.

Ας προχωρήσουμε τώρα στη μέθοδο calculatePay().
Η calculatePay() είναι μια μέθοδος που χρησιμοποιείται για τον υπολογισμό του μισθού ενός μέλους του προσωπικού. Θα

παρατηρήσουμε ότι είναι πολύ παρόμοια με μια συνάρτηση, εκτός από την παράμετρο self.
Μια μέθοδος είναι σχεδόν ταυτόσημη με μια συνάρτηση εκτός από το γεγονός ότι μια μέθοδος βρίσκεται μέσα σε μια κλάση και οι περισσότερες μέθοδοι έχουν τη self ως παράμετρο.

Μέσα στη μέθοδο calculatePay(), πρώτα ζητάμε από τον χρήστη να εισάγει τον αριθμό των εργατοωρών που εργάστηκε. Στη συνέχεια, ζητάμε το ωρομίσθιό του και υπολογίζουμε τον μισθό βάσει αυτών των δύο τιμών.
Στη συνέχεια, αναθέτουμε το αποτέλεσμα στη μεταβλητή στιγμιοτύπου self.pay και επιστρέφουμε την τιμή του self.pay.

Παρατηρούμε ότι σε αυτήν τη μέθοδο, δεν προσθέτουμε το self μπροστά από κάποιες μεταβλητές (όπως prompt, hours και hourlyRate). Αυτό συμβαίνει επειδή αυτές οι μεταβλητές είναι τοπικές και υπάρχουν μόνο μέσα στη μέθοδο calculatePay().
Δεν χρειάζεται να προσθέτουμε το self μπροστά από τις τοπικές μεταβλητές.

Η κλάση μας λοιπόν έχει τα ακόλουθα στοιχεία:

Μεταβλητές στιγμιοτύπου
position
name
pay

Μέθοδοι
__init__
__str__
calculatePay()

13.1.2 Δημιουργία στιγμιοτύπου /αντικειμένου

Για να χρησιμοποιήσουμε την κλάση, πρέπει να δημιουργήσουμε ένα αντικείμενο από αυτήν. Αυτό είναι γνωστό ως «δημιουργία αντικειμένου», “instantiating object”. Ένα αντικείμενο είναι επίσης γνωστό ως ένα στιγμιότυπο.
Αν και υπάρχουν κάποιες διαφορές μεταξύ ενός αντικειμένου και ενός στιγμιοτύπου, αυτές είναι περισσότερο σημασιολογικές και μπορούμε να χρησιμοποιούμε και τις δύο λέξεις.

Ας δημιουργήσουμε τώρα ένα αντικείμενο Staff.
Τρέχουμε τον κώδικα στο IDLE και τώρα είμαστε έτοιμοι να δημιουργήσουμε ένα αντικείμενο Staff.

Για να το κάνουμε αυτό, γράφουμε
officeStaff1 = Staff(Βασικός, Άννα, 0)

Αυτό είναι λίγο παρόμοιο με τον τρόπο που δηλώνουμε μια μεταβλητή όπου γράφουμε userAge = 10.

Σε αυτήν την περίπτωση, το officeStaff1 είναι το όνομα της μεταβλητής. Ωστόσο, αφού το officeStaff1 δεν είναι ένας ακέραιος αριθμός, δεν του αντιστοιχίζουμε έναν αριθμό. Αντ' αυτού, γράφουμε Staff(‘Βασικός’, ‘Άννα’, 0) στη δεξιά πλευρά. Όταν το κάνουμε αυτό, ουσιαστικά ζητάμε από την κλάση Staff να δημιουργήσει ένα αντικείμενο Staff και να χρησιμοποιήσει τη μέθοδο __init__ για να αρχικοποιήσει τις μεταβλητές στιγμιοτύπου στην κλάση.

Παρατηρήστε ότι έχουμε τρεις τιμές ‘Βασικός’, ‘Άννα’ και 0 μέσα στις παρενθέσεις. Αυτές είναι για τις παραμέτρους pPosition, pName και pPay στη μέθοδο __init__ που κωδικοποιήσαμε προηγουμένως.

Χρησιμοποιούμε αυτές τις τρεις τιμές για να αρχικοποιήσουμε αντίστοιχα τις μεταβλητές στιγμιοτύπου position, name και pay.

Τι συμβαίνει όμως με την πρώτη παράμετρο self; Δεν χρειάζεται να περάσουμε τίποτα για την παράμετρο self. Είναι μια ειδική παράμετρος και η Python θα την προσθέσει αυτόματα κατά την κλήση της μεθόδου.

Αφού δημιουργήσουμε αυτό το αντικείμενο, το αναθέτουμε στο officeStaff1.
Ας δοκιμάσουμε να πληκτρολογήσουμε την παρακάτω δήλωση στο Shell και να πατήσουμε enter.
officeStaff1 = Staff('Βασικός', 'Άννα', 0)
Θα δούμε το μήνυμα:

'Δημιουργία του αντικειμένου "Προσωπικό'

να εμφανίζεται στην οθόνη. Αυτό υποδηλώνει ότι η αρχικοποιητής καλείται.

Τώρα που έχουμε δημιουργήσει ένα αντικείμενο της κλάσης Staff, μπορούμε να το χρησιμοποιήσουμε για να έχουμε πρόσβαση στις μεταβλητές στιγμιοτύπου και τις μεθόδους μέσα στην κλάση. Για να το κάνουμε αυτό, χρησιμοποιούμε τον τελεστή τελείας μετά το όνομα του αντικειμένου για να έχουμε πρόσβαση σε οποιαδήποτε μεταβλητή στιγμιοτύπου ή μέθοδο στην κλάση Staff.

Για παράδειγμα, για να έχουμε πρόσβαση στη μεταβλητή στιγμιοτύπου name, πληκτρολογούμε
officeStaff1.name

Καθώς έχουμε πρόσβαση στη μεταβλητή χρησιμοποιώντας το Python Shell, δεν χρειάζεται να χρησιμοποιήσουμε τη συνάρτηση print() για να εμφανίσουμε την τιμή.
Διαφορετικά, αν δεν χρησιμοποιούμε το Python Shell, θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση print().

Δοκιμάστε να προσθέσετε τις παρακάτω γραμμές στο Shell για να δούμε τι συμβαίνει.
Για να αποκτήσουμε πρόσβαση στη μεταβλητή name πληκτρολογούμε Staff1.name

Αποτέλεσμα:
'Άννα'

Για να αποκτήσουμε πρόσβαση στη μεταβλητή position πληκτρολογούμε Staff1.position

Αποτέλεσμα:
‘Βασικός’

Για να αλλάξουμε την τιμή της μεταβλητής position και να την εκτυπώσουμε ξανά

#αλλαγή της μεταβλητής position
ας δοκιμάσουμε Staff1.position = 'Διευθυντής'

#εκτύπωση της position ξανά
ας δοκιμάσουμεStaff1.position

Αποτέλεσμα:
'Διευθυντής'

Για να αποκτήσουμε πρόσβαση στη μεταβλητή pay
πληκτρολογούμε Staff1.pay
Αποτέλεσμα:
0

Για να χρησιμοποιήσουμε τη μέθοδο calculatePay() για να υπολογίσουμε τον μισθό
πληκτρολογούμε Staff1.calculatePay()

Αποτέλεσμα:
Εισάγετε τις ώρες εργασίας για την Άννα: 10
Εισάγετε το ωρομίσθιο για την Άννα: 15

Για να εκτυπώσουμε ξανά τη μεταβλητή pay
πληκτρολογούμε Staff1.pay

Αποτέλεσμα:
150

Για να εκτυπώσουμε μια αναπαράσταση συμβολοσειράς του αντικειμένου officeStaff1

print(officeStaff1)

Αποτέλεσμα:
Θέση = Διευθυντής, Όνομα = Άννα, Μισθός = 150

Για να εκτυπώσουμε μια αναπαράσταση συμβολοσειράς του αντικειμένου, περνάμε το όνομα του αντικειμένου στην ενσωματωμένη συνάρτηση print().
Όταν το κάνουμε αυτό, η Python θα καλέσει τη μέθοδο __str__ που προγραμματίσαμε προηγουμένως.
Στο παράδειγμά μας, την προγραμματίσαμε ώστε να επιστρέφει τη θέση, το όνομα και τον μισθό του αντικειμένου officeStaff1.

13.1.3 Παράδειγμα κληρονομικότητας

Γνωρίζουμε ήδη, ότι μέσω της κληρονομικότητας μια κλάση μπορεί να κληρονομήσει τα χαρακτηριστικά (μεταβλητές, μεθόδους) μια άλλης (γονικής) κλάσης. Με την κληρονομικότητα λοιπόν, επαναχρησιμοποιούμε κώδικα, με μικρές αλλαγές, συνήθως μικρές προσθήκες. Η αρχική κλάση λοιπόν, ονομάζεται γονική (parent class) και η κλάση η οποία κληρονομεί απ' αυτήν, υποκλάση (subclass) ή κλάση-παιδί (child class).
Στον ορισμό μιας υποκλάσης το όνομα της γονικής κλάσης μπαίνει μέσα σε παρενθέσεις σαν όρισμα της κλάσης-παιδί.

Στο παρακάτω παράδειγμα έχουμε ένα πανεπιστήμιο με φοιτητές και καθηγητές. Οι φοιτητές και οι καθηγητές έχουν κάποια κοινά χαρακτηριστικά, π.χ. όνομα ή ηλικία, αλλά και κάποια μοναδικά χαρακτηριστικά που αφορούν την ιδιότητά τους, όπως είναι ο αριθμός μητρώου για τους φοιτητές και ο μισθός για τους καθηγητές.
Θα μπορούσαμε να δημιουργήσουμε δύο ανεξάρτητες κλάσεις, μία για τους καθηγητές και μία για τους φοιτητές, αλλά η προσθήκη ενός νέου κοινού χαρακτηριστικού θα απαιτούσε την προσθήκη του και στις δύο ανεξάρτητες κλάσεις, οπότε, μια καλύτερη λύση είναι να δημιουργήσουμε μια κοινή γονική κλάση (UniversityMember), και οι κλάσεις Καθηγητής (Professor) και Φοιτητής (Student) να κληρονομήσουν από αυτήν:

Παράδειγμα 1:

```
class UniversityMember:
        def __init__(self, name, age):
                self.name = name
                self.age = age
                print("Μέλος του Πανεπιστημίου:", self.name)
        def show(self):
                print('Όνομα: "{0}" Ηλικία: "{1}" ' .format(self.name,
self.age))

class Professor(UniversityMember):
   def __init__(self, name, age, misthos):
      UniversityMember.__init__(self, name, age)
      self.misthos = misthos
      print('Καθηγητής/τρια:', self.name)
   def show(self):
      UniversityMember.show(self)
      print('Μισθός:', self.misthos)

class Student(UniversityMember):
   def __init__(self, name, age, ar):
      UniversityMember.__init__(self, name, age)
      self.ar = ar
      print('Φοιτητής/τρια:', self.name)
   def show(self):
      UniversityMember.show(self)
      print('ΑΜ:', self.ar)

p = Professor('Κωνσταντίνου Γεώργιος', 48, 1700)
p.show()
s = Student('Γιαννάκου Ελένη', 20, 342)
s.show()
```

Έξοδος:

```
Μέλος του Πανεπιστημίου: Κωνσταντίνου Γεώργιος
Καθηγητής: Κωνσταντίνου Γεώργιος
Όνομα: "Κωνσταντίνου Γεώργιος" Ηλικία: "48"
Μισθός: 1700
Μέλος του Πανεπιστημίου: Γιαννάκου Ελένη
```

Φοιτητής: Γιαννάκου Ελένη
Όνομα: "Γιαννάκου Ελένη" Ηλικία: "20"
ΑΜ: 342

Συνεχίζουμε μελετώντας το παρακάτω παράδειγμα.

Παράδειγμα 2

```
'''
Φτιάξτε μια κλάση Person και δύο υποκλάσεις: Male και Female.
Όλες οι κλάσεις έχουν τη μέθοδο "getGender" η οποία μπορει να
τυπώσει "Άνδρας" για την κλάση Male και "Γυναικα" για την κλάση
Female.
Κατόπιν, φτιάξτε 2 αντικείμενα των κλάσεων και ζητήστε τις εκτυπώσεις
τους.
'''

class Person():
    def getGender(self):
        return "Unknown"

class Male(Person):
    def getGender(self):
        return "Άνδρας"

class Female(Person):
    def getGender(self):
        return "Γυναίκα"

Γιώργος = Male()
Μαρία= Female()
print('Ο Γιώργος είναι', Γιώργος.getGender())
print('Η Μαρία είναι', Μαρία.getGender())
```

Έξοδος:
Ο Γιώργος είναι Άνδρας
Η Μαρία είναι Γυναίκα

13.1.4 Χαρακτηριστικά / Ιδιότητες των κλάσεων και decorators (διακοσμητές)

Ιδιότητες (Properties) στις Κλάσεις της Python

Στην Python, οι κλάσεις είναι σαν σχέδια για τη δημιουργία αντικειμένων. Σκεφτείτε μια κλάση σαν ένα πρότυπο που ορίζει πώς θα μοιάζει και θα συμπεριφέρεται ένα αντικείμενο.

Οι ιδιότητες στις κλάσεις μας επιτρέπουν να ορίσουμε χαρακτηριστικά (μεταβλητές) που ανήκουν σε ένα αντικείμενο.

Ας θεωρήσουμε μια κλάση με το όνομα "Car" (Αυτοκίνητο). Ένα αυτοκίνητο έχει διάφορες ιδιότητες, όπως το χρώμα, η μάρκα και η ταχύτητα. Μπορούμε να ορίσουμε αυτές τις ιδιότητες μέσα στην κλάση. Παράδειγμα:

```
class Car:
    def __init__(self, color, brand):
        self.color = color
        self.brand = brand

# Δημιουργία αντικειμένου της κλάσης Car
my_car = Car("κόκκινο", "Toyota")

# Πρόσβαση στις ιδιότητες του αυτοκινήτου

print(my_car.color)  # Έξοδος: κόκκινο
print(my_car.brand)  # Έξοδος: Toyota
```

Σε αυτό το παράδειγμα, η κλάση `Car` έχει δύο ιδιότητες: `color` (χρώμα) και `brand` (μάρκα). Όταν δημιουργούμε ένα αντικείμενο (`my_car`) της κλάσης `Car`, μπορούμε να αντιστοιχίσουμε τιμές σε αυτές τις ιδιότητες.
Έπειτα, μπορούμε να αποκτήσουμε πρόσβαση στις ιδιότητες χρησιμοποιώντας τη σύνταξη της τελείας (`αντικείμενο.ιδιότητα`).

Διακοσμητές (Decorators) στις Κλάσεις της Python

Στην Python, οι decorators είναι ειδικές συναρτήσεις που τροποποιούν τη συμπεριφορά άλλων συναρτήσεων ή μεθόδων. Μας επιτρέπουν να προσθέσουμε λειτουργικότητα σε μια κλάση ή τις μεθόδους της χωρίς να αλλάξουμε τον αρχικό τους κώδικα.

Ας υποθέσουμε ότι έχουμε μια κλάση με το όνομα `MathOperations` (Μαθηματικές Λειτουργίες), και θέλουμε να προσθέσουμε έναν decorator στη μέθοδο `add_numbers` (προσθήκη αριθμών).

Ο decorator θα εκτυπώσει ένα μήνυμα πριν και μετά την εκτέλεση της μεθόδου.

Παράδειγμα:

```
def decorator(func):
    def wrapper(*args, **kwargs):
        print("Πριν την εκτέλεση της μεθόδου")
        result = func(*args, **kwargs)
        print("Μετά την εκτέλεση της μεθόδου")
        return result
    return wrapper

class MathOperations:
    @decorator
    def add_numbers(self, x, y):
        return x + y

# Δημιουργία αντικειμένου της κλάσης MathOperations
math_obj = MathOperations()

# Κλήση της μεθόδου add_numbers
result = math_obj.add_numbers(2, 3)
```

Έξοδος:

```
Πριν την εκτέλεση της μεθόδου
5
```

Μετά την εκτέλεση της μεθόδου

Σε αυτό το παράδειγμα, ορίσαμε μια συνάρτηση `decorator` που παίρνει μια συνάρτηση `func` ως όρισμα. Μέσα στον decorator, ορίζουμε μια συνάρτηση-περιτύλιγμα (`wrapper`) που προσθέτει επιπλέον λειτουργικότητα πριν και μετά την κλήση της αρχικής συνάρτησης. Η σύνταξη `@decorator` χρησιμοποιείται για να εφαρμόσουμε τον decorator στη μέθοδο `add_numbers` στην κλάση `MathOperations`.

Όταν καλούμε την `math_obj.add_numbers(2, 3)`, ο decorator εκτελείται πριν και μετά την κλήση της μεθόδου.
Εκτυπώνει τα μηνύματα και επιστρέφει το αποτέλεσμα της αρχικής μεθόδου.

Σε αυτήν την περίπτωση, ο decorator δεν αλλάζει τη λειτουργικότητα της μεθόδου, αλλά προσθέτει επιπλέον λειτουργικότητα.

13.1.5 Ιδιότητες των κλάσεων

Τώρα που έχουμε μια βασική κατανόηση των κλάσεων και των αντικειμένων, ας προχωρήσουμε στις ιδιότητες.

Στα παραπάνω παραδείγματα, παρατηρούμε ότι μπορούμε να αποκτήσουμε πρόσβαση στις μεταβλητές στιγμιοτύπου ενός αντικειμένου χρησιμοποιώντας τον τελεστή τελείας.
Αυτό μας διευκολύνει να διαβάσουμε τις μεταβλητές και να τις τροποποιήσουμε όταν χρειάζεται.

Ωστόσο, αυτή η ευελιξία προκαλεί κάποια προβλήματα. Για παράδειγμα, μπορεί να αλλάξουμε ακούσια τη θέση του officeStaff1 σε

μια μη υπαρκτή θέση. Ή μπορεί να αλλάξουμε τον μισθό του officeStaff1 σε λανθασμένο ποσό.

Για να αποτρέψουμε τέτοια σφάλματα, μπορούμε να χρησιμοποιήσουμε ιδιότητες. Οι ιδιότητες μας παρέχουν έναν τρόπο για να ελέγξουμε τις τιμές των αλλαγών που θέλουμε να πραγματοποιήσουμε πριν να επιτρέψουμε την αλλαγή να συμβεί.

Για να δείξουμε πώς λειτουργούν οι ιδιότητες, θα προσθέσουμε μια για τη μεταβλητή “θέση”.

Συγκεκριμένα, θα προσθέσουμε μια ιδιότητα για να διασφαλίσουμε ότι η μεταβλητή “θέση” μπορεί να οριστεί μόνο σε "Βασική" ή "Διευθυντή".

Ωστόσο, πριν το κάνουμε αυτό, θέλουμε να αλλάξουμε το όνομα της μεταβλητής στιγμιοτύπου από θέση (position) σε _position.

Η προσθήκη μιας μόνο κάτω παύλας μπροστά από το όνομα μιας μεταβλητής είναι ένας συμβατικός τρόπος να ενημερώσουμε άλλους προγραμματιστές ότι δεν πρέπει να αγγίζουν αυτήν τη μεταβλητή απευθείας.

Στην Python, υπάρχει μια συνήθης φράση που λέει "είμαστε όλοι συναινούντες ενήλικες εδώ". Αναμένεται από όλους να συμπεριφερόμαστε ως ενήλικες.

Η προσθήκη μιας μόνο κάτω παύλας μπροστά από μια μεταβλητή λέει σε άλλους προγραμματιστές ότι εμπιστευόμαστε ότι θα συμπεριφερθούν υπεύθυνα και δεν θα έχουν πρόσβαση σε αυτήν τη μεταβλητή απευθείας εκτός εάν έχουν έναν πειστικό λόγο.

Παρ’ όλα αυτά, τεχνικά, δεν τους εμποδίζει τίποτα από το να έχουν πρόσβαση σε αυτήν τη μεταβλητή. Εάν το επιθυμούν, μπορούν ακόμα να την προσπελάσουν γράφοντας
officeStaff1._position

Λαμβάνοντας αυτό υπόψη, ας κάνουμε τις εξής αλλαγές στο αρχείο classdemo.py για να ενημερώσουμε τους υπόλοιπους "συναινούντες ενήλικες" ότι δεν πρέπει να έχουν πρόσβαση στη θέση (position) απευθείας:

Αλλαγή της γραμμής
self.position = pPosition
στην __init__ σε
self._position = pPosition

και της γραμμής

return "Position = %s, Name = %s, Pay = %d" %(self.position, self.name, self.pay)

και της μεθόδου __str__ σε

return "Position = %s, Name = %s, Pay = %d" %(self._position, self.name, self.pay)

Στη συνέχεια, ας δούμε πώς να προσθέσουμε μια ιδιότητα για τη μεταβλητή _position:

Προσθέτουμε τις παρακάτω γραμμές στην κλάση Staff στο αρχείο classdemo.py.

```
@property
def position(self):
        print("Getter Method")
        return self._position

@position.setter
def position(self, value):
if value == 'Διευθυντής' or value == 'Βασική':
        self._position = value
else:
        print('Η θέση είναι άκυρη. Δεν έγιναν αλλαγές.')
```

Η πρώτη γραμμή παραπάνω (@property) ονομάζεται διακοσμητής (decorator). Δεν θα αναλυθούν λεπτομέρειες για το τι είναι ένας διακοσμητής, απλά αυτό που κάνει είναι να μας επιτρέπει να αλλάξουμε τη λειτουργικότητα της μεθόδου που ακολουθεί. Σε αυτήν την περίπτωση, αλλάζει την πρώτη μέθοδο position() σε μια ιδιότητα.

Αυτό σημαίνει ότι λέμε στον μεταγλωττιστή ότι όταν οι χρήστες πληκτρολογούν:

officeStaff1.position

θα πρέπει να χρησιμοποιεί την μέθοδο position() που ακολουθεί για να πάρει την τιμή.

Αυτή η μέθοδος απλά εκτυπώνει το μήνυμα «Getter Method» και επιστρέφει την τιμή της μεταβλητής _position.

Λόγω του διακοσμητή @property που μετατρέπει τη μέθοδο σε ιδιότητα, δεν χρειάζεται να πληκτρολογήσουμε officeStaff1.position() για να έχουμε πρόσβαση στη μέθοδο. Την προσπελαύνουμε όπως μια μεταβλητή χωρίς τις παρενθέσεις.

Στη συνέχεια, έχουμε έναν άλλο διακοσμητή @position.setter ακολουθούμενο από μια δεύτερη μέθοδο position().

Αυτός ο διακοσμητής λέει στο μεταγλωττιστή ότι όταν οι χρήστες προσπαθούν να ενημερώσουν την τιμή του _position γράφοντας κάτι σαν

officeStaff1.position = 'Διευθυντής'

θα πρέπει να χρησιμοποιήσει τη μέθοδο position() που ακολουθεί για να ενημερώσει την τιμή.

Αυτή η δεύτερη μέθοδος position() ονομάζεται μέθοδος ρύθμισης (setter method). Διαθέτει μια παράμετρο με το όνομα value που αντιστοιχεί στη μεταβλητή _position, εφόσον η τιμή είναι είτε 'Διευθυντής' είτε 'Βασική'.

Εάν η τιμή δεν είναι καμία από αυτές, εμφανίζεται το μήνυμα "Η θέση είναι άκυρη. Δεν έγιναν αλλαγές."

Τώρα, αποθηκεύουμε το αρχείο και το ξαναεκτελούμε.

Ας πληκτρολογήσουμε το παρακάτω στο shell:

officeStaff1 = Staff('Βασικός', "Άννα", 0)

Για να έχουμε πρόσβαση στη θέση officeStaff1, γράφουμε

officeStaff1.position

Θα πάρουμε

Getter Method
'Βασικός'
ως έξοδο.

Προηγουμένως, όταν πληκτρολογήσαμε:

officeStaff1.position

προσπελάσαμε απευθείας τη μεταβλητή position.
Τώρα, όταν πληκτρολογούμε:

officeStaff1.position

δεν προσπελαύνουμε πλέον τη μεταβλητή.
Αντ' αυτού, προσπελαύνουμε τη μέθοδο ανάκτησης (Getter method) της ιδιότητας position.

Αυτό φαίνεται από το γεγονός ότι λαμβάνουμε μια επιπλέον γραμμή (Getter method) στην έξοδο.

Δεν είναι τυχαίο ότι ονομάζουμε την ιδιότητα position, που είναι το αρχικό όνομα της μεταβλητής πριν την αλλάξουμε σε _position.

Με αυτόν τον τρόπο, οι χρήστες μπορούν να έχουν πρόσβαση στη θέση του προσωπικού με τον ίδιο τρόπο που είχαν συνηθίσει, πληκτρολογώντας officeStaff1.position.

Παρόλο που κάναμε αρκετές αλλαγές στο αρχείο classdemo.py στο παρασκήνιο, οι τελικοί χρήστες δεν επηρεάζονται από αυτές τις αλλαγές

(εκτός εάν προσπαθήσουν να αλλάξουν τη θέση του προσωπικού σε μια μη έγκυρη τιμή).

Τώρα, ας προσπαθήσουμε να αλλάξουμε τη θέση του officeStaff1. Πληκτρολογήστε το παρακάτω στο Shell:

officeStaff1.position = 'Διευθυντής'

Αυτό αλλάζει τη θέση του προσωπικού σε "Διευθυντής".

Επαληθεύστε το πληκτρολογώντας:

officeStaff1.position

Θα λάβουμε:
Setter method
'Διευθυντής'
ως έξοδο.

Κατόπιν, ας προσπαθήσουμε να αλλάξουμε τη θέση σε "CEO". Πληκτρολογήστε το παρακάτω στο Shell:
officeStaff1.position = 'Διευθυντής Εκτελεστικού Διευθυντή'

Θα λάβουμε:
Η θέση δεν είναι έγκυρη. Δεν έγιναν αλλαγές.
ως έξοδο.

Αυτό δείχνει ότι η μέθοδος ρύθμισης (setter method) μας έχει εμποδίσει να αλλάξουμε τη θέση του προσωπικού σε μια μη έγκυρη τιμή.
Μπορούμε να επαληθεύσουμε ότι η θέση δεν έχει αλλάξει πληκτρολογώντας:

officeStaff1.position

Θα λάβουμε:

Getter method
'Διευθυντής'
ως έξοδο.

13.2.0 Ασκήσεις

1. Φτιάξτε μια κλάση Calculator η οποία να έχει σαν μεθόδους τις βασικές αριθμητικές πράξεις (πρόσθεση, αφαιρεση, πολ/σμό, διαίρεση). Δημιουργήστε τα αντίστοιχα αντικειμενα και ζητήστε την εκτύπωση για κάθε μέθοδο. Σε περίπτωση διαιρεσης με το μηδέν, πρέπει το προγραμματάκι σας να προβλέπει την εκτύπωση ενός error.

2. Γράψτε ένα πρόγραμμα για να δημιουργήσετε μια κλάση Bank, η οποία να αντιπροσωπεύει μια τράπεζα. Δημιουργήστε μεθόδους όπως, δημιουργώ λογαριασμό, κάνω κατάθεση, κάνω ανάληψη, ελέγχω το balance.
Χρησιμοποιήστε if - else, σε παριπτώσεις όπως, σε περίπτωση ανάληψης, το ποσό ειναι μικρότερο από το διαθέσιμο. Ζητήστε εκτυπώσεις