33. ΚΛΑΣΕΙΣ – LAMBDA FUNCTIONS

Επανάληψη κεφαλαίων 11 - 13

33.0.1 Λύσεις των προηγούμενων ασκήσεων

Άσκηση 1

Χρησιμοποιήστε τη συνάρτηση range() για να δημιουργήσετε μια λίστα των αριθμών από το 0 μέχρι το 9. Ονομάστε τη λίστα newlist και χρησιμοποιήστε μόνο μια γραμμή κώδικα (list comprehension).

Λύση

```
newlist = [x for x in range(10)]
```

Δημιουργήσαμε μια λίστα από το 0 μέχρι και το 9 και της έχουμε δώσει και όνομα σε μια και μόνο γραμμή κώδικα.

Αν πάρουμε μια εκτύπωση της newlist θα έχουμε:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Άσκηση 2

Χρησιμοποιήστε σαν δεδομένα τα παραπάνω, αλλά δεχτείτε μόνο αριθμούς μικρότερους από το 5:

Λύση

```
newlist = [x for x in range(10) if x < 5]
```

Ζητώντας την εκτύπωση της παραπάνω λίστας έχουμε:

```
[0, 1, 2, 3, 4]
```

Άσκηση 3

Έστω η παρακάτω λίστα με ονόματα φρούτων:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
```

Μετατρέψτε τα στοιχεία της λίστας σε κεφαλαία γράμματα με μια και μόνο γραμμή κώδικα.

Λύση

```
newlist = [x.upper() for x in fruits]
```

Δηλαδή:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x.upper() for x in fruits]
print(newlist)
```

Έξοδος:

```
 ['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
```

Άσκηση 4

Αντικαταστήστε όλες τις τιμές των στοιχείων της λίστας με τη λέξη

```
“hello”:
```

Λύση

```
newlist = ['hello' for x in fruits]
```

Δηλαδή:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = ['hello' for x in fruits]
print(newlist)
```

Έξοδος:

```
['hello', 'hello', 'hello', 'hello', 'hello']
```

Άσκηση 5

Γράψτε μια γραμμή κώδικα για να πάρετε την παραπάνω λίστα, η οποία όμως να επιστρέφει "orange" αντί για "banana":

Λύση

```
newlist = [x if x != "banana" else "orange" for x in fruits]
```

Δηλαδή:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x if x != "banana" else "orange" for x in fruits]
print(newlist)
```

Η παραπάνω έκφραση λέει:

Επίστρεψε κάθε φρούτο αν δεν είναι μπανάνα, όμως αν είναι μπανάνα,

επίστρεψε πορτοκάλι

Έξοδος:

```
['apple', 'orange', 'cherry', 'kiwi', 'mango']
```

Άσκηση 6

Φτιάξτε μια λίστα η οποία να τυπώνει όλα τα τετράγωνα των αριθμών

από το 1 μέχρι και το 9.

Λύση:

```
squares_lst = [x**2 for x in range(10)]
print(squares_lst)
```

Έξοδος:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

33.1.1 Κλάσεις - Παραδείγματα κι εφαρμογές

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
```

33.1.2 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)
```

33.1.3 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 information2(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.

33.1.4 Παραδείγματα κληρονομικότητας και πολυμορφισμού

Παράδειγμα 1

Δημιουργία μια γονικής κλάσης "Animal" με μια μέθοδο "sound" που επιστρέφει τον ήχο που βγάζει το ζώο. Στη συνέχεια, δημιουργία τριών παιδικών (child) κλάσεων "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 κατάλληλος για κάθε παιδική κλάση. Αυτό μας επιτρέπει να δημιουργήσουμε ευέλικτο και επαναχρησιμοποιήσιμο κώδικα.

33.1.5 Κλάση Vehicle

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"
```

33.1.6 Κλάση Staff

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.

(class_Staff_less.13.py)

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 της %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.

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()

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 # στον IDLE
```

Για να αποκτήσουμε πρόσβαση στη μεταβλητή name πληκτρολογούμε:

```
Print(officeStaff1.name)
```

Αποτέλεσμα:

```
'Άννα'
```

Για να αποκτήσουμε πρόσβαση στη μεταβλητή position

πληκτρολογούμε

```
Print(officeStaff1.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 δοκιμάσουμε

```
Staff1.position
```

Αποτέλεσμα:

```
'Διευθυντής'
```

Για να αποκτήσουμε πρόσβαση στη μεταβλητή pay

πληκτρολογούμε

```
Print(officeStaff1.pay)
```

Αποτέλεσμα:

```
0
```

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

```
officeStaff1.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.

33.1.7 Κλάση Restaurant

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
```

33.1.8 Lambda Functions

Στην Python, οι συναρτήσεις lambda, επίσης γνωστές ως ανώνυμες συναρτήσεις, είναι μικρές συναρτήσεις που δημιουργούνται inline και δεν απαιτούν μια τυπική δήλωση def. Χρησιμοποιούνται συχνά όταν χρειαζόμαστε μια απλή συνάρτηση για ένα σύντομο χρονικό διάστημα, συνήθως ως όρισμα σε συναρτήσεις υψηλότερης προτεραιότητας ή σε καταστάσεις όπου η δημιουργία μιας ξεχωριστής συνάρτησης θα ήταν περιττή.

Παραδείγματα

1. Βασική χρήση

```
add = lambda x, y: x + y
result = add(3, 5)
print(result)  # Output: 8
```

2. Ταξινόμηση με lambda function

```
students = [('Alice', 25), ('Bob', 20), ('Charlie', 22)]
sorted_students = sorted(students, key=lambda x: x[1])
print(sorted_students)  # Output: [('Bob', 20), ('Charlie',
22), ('Alice', 25)]
```

3. Φιλτράρισμα

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4, 6, 8]
```

4. Ταξινόμηση λεξικού με Lambda

```
dictionary = {'one': 1, 'four': 4, 'three': 3, 'six': 6}
sorted_dict = dict(sorted(dictionary.items(), key=lambda x:
x[1])) # Βάζουμε x[1] ώστε να δηλώσουμε ότι η ταξινόμηση θα
γίνει σύμφωνα με το δεύτερο στοιχείο του ζευγαριού
«κλειδί:τιμή», δηλ. την τιμή
print(sorted_dict)  # Output: {'one': 1, 'three': 3, 'four':
4, 'six': 6}
```

5. Lambda με εκφράσεις συνθηκών

```
is_even = lambda x: True if x % 2 == 0 else False
print(is_even(4))  # Output: True
print(is_even(5))  # Output: False
```

33.2.0 Ασκήσεις

Άσκηση 1

Γράψτε μια συνάρτηση η οποία μας επιστρέφει μόνο τις μεγάλες λέξεις από τη λίστα:

```
long_words=['blog', 'Treehouse', 'Python_Rules', 'hi'])
```

Μπορούμε να πούμε πως οποιαδήποτε λέξη μεγαλύτερη από 7 χαρακτήρες είναι μια μεγάλη λέξη. Γράψτε την πρώτα με τον κλασικό τρόπο και κατόπιν χρησιμοποιήστε list comprehensions.

Άσκηση 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.

Άσκηση 3 (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.

Άσκηση 4

Χρησιμοποιήστε lambda function για να ταξινομήσετε την παρακάτω λίστα (με στοιχεία 3 tuples), σύμφωνα με το δεύτερο στοιχείο: my_list = [("apple", 50), ("banana", 10), ("cherry", 30)]

Άσκηση 5

Χρησιμοποιήστε lambda function για να ταξινομήσετε την παρακάτω λίστα σύμφωνα με τον τελευταίο χαρακτήρα της κάθε συμβολοσειράς: my_list = ["apple", "banana", "cherry"]