14. Κλασεις III – ΠΟΛΥΜΟΡΦΙΣΜΟΣ & ΕΝΘΥΛΑΚΩΣΗ

14.0 Λύσεις προηγούμενων ασκήσεων

1.

'''

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

'''

```
class Bank:
    def __init__(self):
        self.customers = {}

    def create_account(self, account_number, initial_balance=0):
        if account_number in self.customers:
            print("Ο λογαριασμός υπάρχει ήδη.")
        else:
            self.customers[account_number] = initial_balance
            print("Ο λογαριασμός δημιουργήθηκε επιτυχώς.")

    def make_deposit(self, account_number, amount):
```

```
        if account_number in self.customers:
            self.customers[account_number] += amount
            print("Η κατάθεση πραγματοποιήθηκε επιτυχώς.")
        else:
            print("Ο αριθμός λογαριασμού δεν υπάρχει.")

    def make_withdrawal(self, account_number, amount):
        if account_number in self.customers:
            if self.customers[account_number] >= amount:
                self.customers[account_number] -= amount
                print("Η ανάληψη ήταν επιτυχής.")
            else:
                print("Μη επαρκές διαθέσιμο υπόλοιπο.")
        else:
            print("Ο αριθμός λογαριασμού δεν υπάρχει.")

    def check_balance(self, account_number):
        if account_number in self.customers:
            balance = self.customers[account_number]
            print(f"Διαθέσιμο κεφάλαιο: {balance}")
        else:
            print("Ο αριθμός λογαριασμού δεν υπάρχει.")


# Παράδειγμα χρήσης
bank = Bank()
```

```
acno1= "SB-123"
dep_amt_1 = 1000


print("Νέος αρ. λογ.: ",acno1,"Ποσό κατάθεσης:",dep_amt_1)
bank.create_account(acno1, dep_amt_1)

acno2= "SB-124"
damt2 = 1500

print("Νέος αρ. λογ.: ",acno2,"Deposit Amount:",damt2)
bank.create_account(acno2, damt2)
wamt1 = 600
print("\nΚατάθεση €:",wamt1,"σε αρ. λογ.",acno1)
bank.make_deposit(acno1, wamt1)
wamt2 = 350
print("Ανάληψη €:",wamt2,"από αρ. λογ.",acno2)
bank.make_withdrawal(acno2, wamt2)
print("Αρ. λογ.",acno1)
bank.check_balance(acno1)
print("Αρ. λογ.",acno2)
bank.check_balance(acno2)
wamt3 = 1200
print("Ανάληψη €:",wamt3,"Από αρ. λογ..",acno2)
bank.make_withdrawal(acno2, wamt3)
```

```
acno3 = "SB-134"
print("Αρ. λογ.",acno3)
bank.check_balance(acno3)  # Μη-υπαρκτός αριθμός λογαριασμού
```

2.

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

class Calculator:
    def add(self, x, y):
        return x + y

    def subtract(self, x, y):
        return x - y

    def multiply(self, x, y):
        return x * y

    def divide(self, x, y):
        if y != 0:
            return x / y
```

```
        else:
            return ("Δεν μπορει να γίνει διαιρεση με το '0'.")

# Δημιουργία στιγμιοτύπου
calculator = Calculator()

# Πρόσθεση
result = calculator.add(7, 5)
print("7 + 5 =", result)

# Αφαίρεση
result = calculator.subtract(34, 21)
print("34 - 21 =", result)

# Πολλαπλασιασμός
result = calculator.multiply(54, 2)
print("54 * 2 =", result)

# Διαίρεση
result = calculator.divide(144, 2)
print("144 / 2 =", result)

# Διαίρεση με το μηδέν (τυπώνει error)
result = calculator.divide(45, 0)
print("45 / 0 =", result)
```

14.1 ΚΛΑΣΕΙΣ – ΟΡΟΙ ΚΑΙ ΕΝΝΟΙΕΣ - ΕΜΠΕΔΩΣΗ

Μάθαμε για τις βασικές έννοιες των κλάσεων στην Python. Ας δούμε μερικά πράγματα σε σχέση με τα χαρακτηριστικά και τις μεθόδους. Θα δούμε πώς μπορούμε να ορίσουμε χαρακτηριστικά, να αρχικοποιήσουμε αντικείμενα, να έχουμε πρόσβαση στις τιμές τους και να εκτελούμε λειτουργίες χρησιμοποιώντας μεθόδους.

14.1.0 Ορισμός Χαρακτηριστικών και Αρχικοποίηση Αντικειμένων

Ας μιλήσουμε για τα χαρακτηριστικά. Τα χαρακτηριστικά είναι τα γνωρίσματα ή οι πληροφορίες που σχετίζονται με ένα αντικείμενο. Στην Python, ορίζουμε χαρακτηριστικά μέσα σε μια κλάση χρησιμοποιώντας τη μέθοδο `__init__`, γνωστή και ως constructor. Καλείται αυτόματα όταν δημιουργείται ένα αντικείμενο από την κλάση.

Παράδειγμα.

Θα δημιουργήσουμε μια κλάση με το όνομα `Person` με χαρακτηριστικά όπως το `name` και το `age`. Ανοίξτε τον IDLE και πληκτολογήστε:

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

person1 = Person("Άννα", 25)

Ορίσαμε την κλάση `***Person***` με τη μέθοδο `***__init__***` που παίρνει τα `***name***` και `***age***` ως παραμέτρους. Χρησιμοποιούμε τη λέξη-κλειδί `***self***` για να αναφερθούμε στο αντικείμενο που δημιουργείται. Μέσα στη μέθοδο, αντιστοιχίζουμε τις τιμές που περνάνε στα χαρακτηριστικά του αντικειμένου χρησιμοποιώντας τη σύνταξη `***self.attribute_name***`.

Κατόπιν δημιουργούμε ένα αντικείμενο `***person1***` της κλάσης `***Person***`, περνώντας το όνομα "***Άννα***" και την ηλικία ***25*** ως ορίσματα. Αυτό θα αρχικοποιήσει το αντικείμενο με τις καθορισμένες τιμές των χαρακτηριστικών.

14.2 Πρόσβαση σε Χαρακτηριστικά και Μεθόδους

Μόλις δημιουργήσουμε ένα αντικείμενο, μπορούμε να έχουμε πρόσβαση στα γνωρίσματά του χρησιμοποιώντας την σημείωση με τελεία (`***αντικείμενο.χαρακτηριστικό***`). Ας δοκιμάσουμε να έχουμε πρόσβαση στα χαρακτηριστικά του `***person1***`.

print(person1.name)

print(person1.age)

Εκτελώντας αυτόν τον κώδικα, μπορούμε να δούμε ότι μπορούμε να έχουμε πρόσβαση και να εκτυπώσουμε τις τιμές των γνωρισμάτων

`***name***` και `***age***` για το `***person1***`. Τα χαρακτηριστικά παρέχουν έναν τρόπο αποθήκευσης και ανάκτησης δεδομένων που σχετίζονται με ένα αντικείμενο.

Εκτός από τα χαρακτηριστικά, τα αντικείμενα μπορούν επίσης να έχουν μεθόδους, που είναι συναρτήσεις που ορίζονται μέσα σε μια κλάση. Οι μέθοδοι μπορούν να εκτελούν λειτουργίες στα δεδομένα του αντικειμένου ή να πραγματοποιούν συγκεκριμένες ενέργειες. Ας προσθέσουμε μια μέθοδο στην κλάση `***Person***` μας.

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

    def greet(self):
        print(f"Γεια σας, το όνομά μου είναι {self.name}.")

person1 = Person("Άννα", 25)
person1.greet()
```

Σε αυτήν την έκδοση της κλάσης `**Person**`, προσθέσαμε μια μέθοδο `**greet*()***` που εκτυπώνει ένα μήνυμα χαιρετισμού μαζί με το όνομα του προσώπου. Παρατηρήστε ότι η παράμετρος `**self**` χρησιμοποιείται μέσα στη μέθοδο για να έχουμε πρόσβαση στα χαρακτηριστικά του αντικειμένου. Μπορούμε να καλέσουμε τη μέθοδο `***greet()***` στο αντικείμενο `***person1***` και να εκτυπώσουμε το σχετικό μήνυμα χαιρετισμού.

Συνοψίζοντας, τα χαρακτηριστικά αναπαριστούν τα δεδομένα που σχετίζονται με ένα αντικείμενο, ενώ οι μέθοδοι παρέχουν λειτουργίες και ενέργειες που μπορούν να εκτελέσουν τα αντικείμενα.

Τώρα, όταν καλούμε τη μέθοδο `greet` στο αντικείμενο `person1`, θα εκτυπωθεί το μήνυμα χαιρετισμού, συμπεριλαμβανομένου του ονόματος του ατόμου.

14.3 Τροποποίηση των τιμών των χαρακτηριστικών

Συχνά, μπορεί να χρειαστεί να τροποποιήσουμε τις τιμές των χαρακτηριστικών μετά τη δημιουργία του αντικειμένου. Μπορούμε να το κάνουμε αυτό αντιστοιχίζοντας άμεσα νέες τιμές στα χαρακτηριστικά χρησιμοποιώντας τη σημείωση με τελεία(dot notation) . Ας τροποποιήσουμε το χαρακτηριστικό age του `***person1***`.

person1.age = 26

print(person1.age)

Με την αντιστοίχιση μιας νέας τιμής στο γνώρισμα `age`, έχουμε ενημερώσει την τιμή του γνωρίσματος για το `person1`. Όταν εκτυπώνουμε `person1.age`, αντικατοπτρίζει την ενημερωμένη τιμή του: 26.

Οπότε, σε αυτήν την ενότητα, εξερευνήσαμε πώς να ορίσουμε χαρακτηριστικά, να αρχικοποιήσουμε αντικείμενα, να έχουμε πρόσβαση στις τιμές τους και να πραγματοποιήσουμε λειτουργίες χρησιμοποιώντας μεθόδους.

Αυτές οι έννοιες αποτελούν τα θεμέλια των κλάσεων στην Python.

14.4.0 Άσκηση εμπέδωσης

Ας δημιουργήσουμε μια κλάση με το όνομα `***Car***` με χαρακτηριστικά όπως `***make***`, `***model***` και `***year***`.

Αρχικοποιήστε ένα αντικείμενο της κλάσης με τις λεπτομέρειες του αγαπημένου μας αυτοκινήτου.

Στη συνέχεια, προσθέστε μια μέθοδο με το όνομα `start_engine` που εκτυπώνει ένα μήνυμα όπως

"***Το [year] [make] [model] " ξεκινάει τη μηχανή."***

Μια δυνατή λύση για την άσκηση είναι:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def start_engine(self):
        print(f"To {self.year} {self.make} {self.model} ξεκινάει τη μηχανή")

# Αρχικοποίηση ενός αντικειμένου και κλήση της μεθόδου start_engine
my_car = Car("Toyota", " Corolla", 2022)
```

my_car.start_engine()

Έξοδος:

Το 2022 Toyota Corolla ξεκινάει τη μηχανή.

Σε αυτήν τη λύση, καθορίζουμε την κλάση `***Car***` με τα χαρακτηριστικά `***make***`, `***model***` και `***year***` στον κατασκευαστή (`***__init__***`**).** Ορίζεται επίσης η μέθοδος `***start_engine***`, η οποία απλώς εκτυπώνει ένα μήνυμα με τις λεπτομέρειες του αυτοκινήτου χρησιμοποιώντας τις τιμές των χαρακτηριστικών.

Στη συνέχεια, δημιουργούμε ένα αντικείμενο `***my_car***` με τη μάρκα "***Toyota***", το μοντέλο "***Corolla***" και το έτος ***2022***. Τέλος, καλούμε τη μέθοδο `***start_engine***` στο `***my_car***`, η οποία εκτυπώνει το επιθυμητό μήνυμα.

Μπορείτε ελεύθερα να τροποποιήσετε τον κώδικα ανάλογα με τις προτιμήσεις σας και να δοκιμάσετε διάφορες τιμές χαρακτηριστικών για να εξερευνήσετε περαιτέρω.

14.5 Κληρονομικότητα

Στην προηγούμενη ενότητα, καλύψαμε τις βασικές αρχές των κλάσεων και τον τρόπο εργασίας με χαρακτηριστικά και μεθόδους. Τώρα, ας εξερευνήσουμε μια ισχυρή έννοια που ονομάζεται κληρονομικότητα, η οποία μας επιτρέπει να δημιουργούμε νέες κλάσεις βασισμένες σε υπάρχουσες. Η κληρονομικότητα προάγει την επαναχρησιμοποίηση κώδικα και διευκολύνει τη δημιουργία ιεραρχιών σχετικών κλάσεων.

14.5.1 Επισκόπηση - Ορισμός

Η κληρονομικότητα είναι ένας μηχανισμός στον αντικειμενοστραφή προγραμματισμό που επιτρέπει σε μια κλάση (η κλάση-κόρη ή υποκλάση) να κληρονομεί τα χαρακτηριστικά και τη συμπεριφορά από μια άλλη κλάση (την κλάση-μητέρα ή υπερκλάση). Η κλάση-κόρη μπορεί στη συνέχεια να επεκτείνει ή να τροποποιήσει τα κληρονομημένα χαρακτηριστικά και μεθόδους ανάλογα με τις ανάγκες της.

Ας δούμε ένα απλό παράδειγμα. Ας υποθέσουμε ότι έχουμε μια κλάση που ονομάζεται " Animal " με χαρακτηριστικά όπως "***name***" και "***age***", καθώς και μεθόδους όπως "***food()"*** και "***sleep()***". Τώρα, θέλουμε να δημιουργήσουμε μια πιο συγκεκριμένη κλάση, ας πούμε "***Dog***", η οποία μοιράζεται μερικά κοινά χαρακτηριστικά με το "***Animal***", αλλά έχει επίσης τα δικά της μοναδικά χαρακτηριστικά και μεθόδους.

14.5.2 Δημιουργία Υποκλάσης

Για να δημιουργήσουμε μια υποκλάση στην Python, καθορίζουμε την υπερκλάση στον ορισμό της υποκλάσης.

Ας δούμε πώς λειτουργεί αυτό με το παρακάτω παράδειγμά.

Πληκτρολογούμε στον IDLE τα παρακάτω.

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

    def eat(self):
        print("Το ζώο τρώει.")

    def sleep(self):
        print("Το ζώο κοιμάται.")

class Dog(Animal):
    def __init__(self, name, age, breed):
        super().__init__(name, age)
        self.breed = breed

    def bark(self):
        print("Ο σκύλος γαβγίζει.")
```

Έχουμε καθορίσει την κλάση "***Animal***" με τα χαρακτηριστικά "***name***" και "***age***", καθώς και τις μεθόδους "***food()***" και "***sleep()***".

Στη συνέχεια, δημιουργούμε την κλάση "***Dog***" ως υποκλάση της "***Animal***" δηλώνοντας την "***Animal***" μέσα στις παρενθέσεις μετά το όνομα της κλάσης.

Η κλάση "***Dog***" έχει το δικό της γνώρισμα με το όνομα "***breed***", το οποίο συμπεριλαμβάνουμε στον κατασκευαστή χρησιμοποιώντας το "init". Χρησιμοποιούμε τη συνάρτηση "super()" για να καλέσουμε τον κατασκευαστή της υπερκλάσης και να περάσουμε τις απαραίτητες παραμέτρους. Αυτό μας επιτρέπει να κληρονομήσουμε και να αρχικοποιήσουμε τα χαρακτηριστικά "***name***" και "***age***" από την κλάση "***Animal***".

14.5.3 Υπερφόρτωση (override) μεθόδων

*Σε ορισμένες περιπτώσεις, η υποκλάση μπορεί να χρειαστεί να τροποποιήσει τη συμπεριφορά μιας μεθόδου που κληρονομήθηκε από τη γονική κλάση. Αυτό ονομάζεται "**υπερφόρτωση» μεθόδου "method override"**.*

Ας δούμε τι συμβαίνει με την τροποποιημένη μέθοδο `***sleep()***` στην κλάση `***Dog***` παρακάτω:

```
class Dog(Animal):
    # ... (προηγούμενος κώδικας)

    def sleep(self):
        print("Ο σκύλος κοιμάται με ανοιχτά τα μάτια.")
```

Σε αυτήν την ενημερωμένη έκδοση της κλάσης `***Dog***`, έχουμε ξαναορίσει τη μέθοδο `***sleep()***`. Με αυτόν τον τρόπο, έχουμε κάνει override της μεθόδου `***sleep()***` που κληρονομήθηκε από την κλάση `***Animal***`. Τώρα, όταν η μέθοδος `***sleep()***` καλείται σε ένα αντικείμενο `***Dog***`, θα εκτελεί τον τροποποιημένο κώδικα που είναι συγκεκριμένος για την κλάση `***Dog***`.

14.5.4 Πρόσβαση σε μεθόδους της γονικής κλάσης χρησιμοποιώντας τη super()

Εκτός από την αντικατάσταση μεθόδων, μια υποκλάση μπορεί επίσης να έχει πρόσβαση και να καλεί μεθόδους από τη γονική κλάση χρησιμοποιώντας τη συνάρτηση `***super()***`.

Αυτό μας επιτρέπει να επαναχρησιμοποιούμε και να επεκτείνουμε τη λειτουργικότητα της γονικής κλάσης. Ας τροποποιήσουμε (κάνουμε override) τη μέθοδο `***bark()***` στην κλάση `***Dog***` για να το δείξουμε.

```
class Dog(Animal):
    # ... (προηγούμενος κώδικας)

    def bark(self):
        super().sleep()  # Κλήση της μεθόδου της γονικής κλάσης
        print("Ο σκύλος γαβγίζει.")
```

Εδώ, μέσα στη μέθοδο `***bark()***`, χρησιμοποιούμε την `***super().sleep()***` για να καλέσουμε τη μέθοδο `***sleep()***` από την κλάση `***Animal***`. Αυτό μας επιτρέπει να εκτελέσουμε τη συμπεριφορά της γονικής κλάσης πριν προσθέσουμε επιπλέον λειτουργικότητα που είναι συγκεκριμένη για την κλάση `***Dog***`.

14.5.5 Πολλαπλή Κληρονομικότητα

Επιπλέον, η Python υποστηρίζει την πολλαπλή κληρονομικότητα, που επιτρέπει σε μια κλάση να κληρονομεί από πολλές γονικές κλάσεις. Αυτό σημαίνει ότι μια υποκλάση μπορεί να παραλαμβάνει χαρακτηριστικά και μεθόδους από πολλές πηγές.

Αν και μπορεί να είναι ένα ισχυρό εργαλείο, πρέπει να χρησιμοποιείται προσεκτικά για να διατηρηθεί η σαφήνεια του κώδικα και να αποφεύγονται πιθανές συγκρούσεις.

Ωστόσο, λόγω της πολυπλοκότητας που συνεπάγεται η πολλαπλή κληρονομικότητα, δεν θα εμβαθύνουμε περισσότερο σε αυτήν σε αυτό το εισαγωγικό μάθημα.

Απλά ας γνωρίζουμε ότι υπάρχει ως ένα προηγμένο θέμα που μπορείτε να διερευνήσετε στο μέλλον.

Παρ' όλα αυτά ας έχουμε ένα παράδειγμα:

```
# Κλάση γονέας 1
class Animal:
    def __init__(self, name):
        self.name = name

    def eat(self):
        print(f"Ο {self.name} τρώει.")

# Κλάση γονέας 2
class CanFly:
    def fly(self):
        print(f"Ο {self.name} πετάει.")
```

```
# Κλάση παιδί που κληρονομεί από τις κλάσεις Animal και CanFly
class Bird(Animal, CanFly):
    def __init__(self, name):
        # Καλούμε τους κατασκευαστές των γονικών κλάσεων
        Animal.__init__(self, name)
        CanFly.__init__(self)

bird = Bird("Σπουργίτης")
bird.eat()  # Έξοδος: Ο Σπουργίτης τρώει.
bird.fly()  # Έξοδος: Ο Σπουργίτης πετάει.
```

Σε αυτό το παράδειγμα, έχουμε τρεις κλάσεις: `***Animal***`, `***CanFly***` και `***Bird***`.

Η κλάση `***Animal***` αναπαριστά ένα ζώο και έχει έναν μέθοδο `***__init__***` που αρχικοποιεί το χαρακτηριστικό `name` και έναν μέθοδο `***eat***`.

Η κλάση `***CanFly***` αναπαριστά την ικανότητα να πετάει και έχει μια μέθοδο `***fly***`.

Η κλάση `Bird` είναι μια παιδική κλάση που κληρονομεί και από τις δύο κλάσεις, `***Animal***` και `***CanFly***`. Έχει μια μέθοδο `***__init__***` που καλεί τους κατασκευαστές και των δύο γονικών κλάσεων. Αυτό της επιτρέπει να κληρονομήσει τα χαρακτηριστικά και τις μεθόδους και από τις δύο κλάσεις.

Σ’ αυτό το παράδειγμα, δημιουργούμε μια έκδοση της κλάσης `***Bird***` που ονομάζεται `***bird***` και δίνουμε το όνομα "***Σπουργίτης***" ως όρισμα στον κατασκευαστή της.

Στη συνέχεια, καλούμε τη μέθοδο `eat` στο αντικείμενο `***bird***`, η οποία εκτυπώνει το μήνυμα "***Ο Σπουργίτης τρώει***".

Στη συνέχεια, καλούμε τη μέθοδο `***fly***` στο αντικείμενο `***bird***`, η οποία εκτυπώνει το μήνυμα "***Ο Σπουργίτης πετάει***".

Η κλάση `***Bird***` μπορεί να έχει πρόσβαση και να χρησιμοποιήσει τόσο τη μέθοδο `eat` από την κλάση `***Animal***`, όσο και τη μέθοδο `***fly***` από την κλάση `***CanFly***` χάρη στην πολλαπλή κληρονομικότητα.

Συμπεράσματα

Σε αυτήν την ενότητα, μάθαμε για την κληρονομικότητα, που μας επιτρέπει να δημιουργούμε υποκλάσεις που κληρονομούν χαρακτηριστικά και μεθόδους από γονικές κλάσεις. Εξερευνήσαμε τη δημιουργία υποκλάσεων, την αντικατάσταση μεθόδων και την πρόσβαση σε μεθόδους της γονικής κλάσης χρησιμοποιώντας το `***super()***`.

14.5.6 Άσκηση εμπέδωσης

Δημιουργήστε μια υποκλάση με όνομα `Cat` που κληρονομεί από την κλάση `Animal`. Προσθέστε μια μοναδική μέθοδο που είναι συγκεκριμένη για την κλάση `Cat` και δοκιμάστε τον κώδικά σας δημιουργώντας ένα αντικείμενο `Cat` και καλώντας τις μεθόδους του.

14.6.0 Προηγμένα θέματα κλάσεων

Ας δούμε κάποια προηγμένα θέματα που σχετίζονται με τις κλάσεις στην Python.

Θα δούμε τις μεταβλητές κλάσης, την ενθυλάκωση (encapsulation) και τον πολυμορφισμό. Αυτές οι έννοιες θα ενισχύσουν ακόμα περισσότερο την κατανόησή μας για τον αντικειμενοστραφή προγραμματισμό. Ας αρχίσουμε!

14.6.1 Μεταβλητές Κλάσης

Α. Οι μεταβλητές κλάσης κοινοποιούνται μεταξύ όλων των περιπτώσεων μιας κλάσης.

Β. Ορίζονται εντός της κλάσης, αλλά έξω από κάθε μέθοδο.

Γ. Οι μεταβλητές κλάσης προσπελαύνονται χρησιμοποιώντας το όνομα της κλάσης ή τα αντικείμενα της κλάσης.

14.6.1 Ενθυλάκωση και Μεταβλητές Πρόσβασης

Α. Η ενθυλάκωση (encapsulation) είναι η πρακτική της συσκευασίας δεδομένων και μεθόδων εντός μιας κλάσης.

Β. Οι μεταβλητές πρόσβασης ελέγχουν την ορατότητα και την προσβασιμότητα των μελών μιας κλάσης.

Γ. Η Python χρησιμοποιεί συμβάσεις ονομασίας για να υποδηλώνει επίπεδα πρόσβασης:

- Δημόσια μέλη: Καμία σύμβαση ονομασίας (προσπελάσιμα από οπουδήποτε).

- Προστατευμένα μέλη: Ξεκινούν με ένα underscore (σύμβαη, δεν επιβάλλεται).

- Ιδιωτικά μέλη: Ξεκινούν με διπλό underscore, δηλ. dunder (ονοματοποίηση ονομάτων – name mangling).

Παράδειγμα:

```
# Δημιουργία μιας κλάσης με όνομα 'Person'
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age  # Ενθυλακωμένη μεταβλητή

    def display_info(self):
        print("Όνομα:", self.name)
        print("Ηλικία:", self.__age)

# Δημιουργία ενός αντικειμένου της κλάσης 'Person'
```

person = Person("Γιάννης", 25)

 # Πρόσβαση στη συνδεδεμένη μεταβλητή έμμεσα χρησιμοποιώντας μια μέθοδο της κλάσης

person.display_info()

- Δημιουργούμε μια κλάση με το όνομα `Person`, η οποία αναπαριστά ένα άτομο και έχει δύο χαρακτηριστικά: `***name***` και `***__age***`.

- Γνωρίζουμε ότι μέθοδος `***__init__***` είναι μια ειδική μέθοδος (κατασκευαστής) που καλείται όταν δημιουργούμε ένα αντικείμενο της κλάσης. Παίρνει τα `***name***` και `***age***` ως παραμέτρους και τα αναθέτει στα αντίστοιχα χαρακτηριστικά του αντικειμένου.

- Το χαρακτηριστικό `***__age***` είναι συνδεδεμένο χρησιμοποιώντας διπλά υπογράμμιση (`__`) στην αρχή του ονόματός του. Αυτή η συνήθεια στην Python υποδηλώνει ότι το χαρακτηριστικό προορίζεται να είναι ιδιωτικό και δεν πρέπει να προσπελαύνεται απευθείας από έξω της κλάσης.

- Ορίζουμε μια μέθοδο με το όνομα `***display_info***` που εκτυπώνει το όνομα και την ηλικία του ατόμου.

- Στη μέθοδο `***display_info***`, μπορούμε να αποκτήσουμε πρόσβαση στην ενθυλακωμένη μεταβλητή `***__age***` απευθείας επειδή βρισκόμαστε μέσα στην κλάση.

- Δημιουργούμε ένα αντικείμενο `***person***` της κλάσης `***Person***`, περνώντας τις τιμές "***Γιάννης***" και ***25*** ως ορίσματα στον κατασκευαστή.

- Τέλος, καλούμε τη μέθοδο `***display_info***` στο αντικείμενο `***person***` για να εκτυπωθούν οι πληροφορίες του ατόμου.

14.6.2 Πολυμορφισμός και Υπερφόρτωση Μεθόδων

Α. Ο πολυμορφισμός επιτρέπει τη χρήση αντικειμένων από διάφορες κλάσεις με ανταλλάξιμο τρόπο.

Β. Η υπερφόρτωση μεθόδων είναι η δυνατότητα ορισμού πολλαπλών μεθόδων με το ίδιο όνομα, αλλά διαφορετικές παραμέτρους.

Γ. Στην Python, η υπερφόρτωση μεθόδων επιτυγχάνεται μέσω προεπιλεγμένων τιμών παραμέτρων και `*args` ή `**kwargs`.

14.6.3 Άσκηση: Εφαρμογή Προηγμένων Εννοιών Κλάσεων

Σενάριο – Άσκηση (class_BankAccount.py):

Έχετε προσληφθεί για να δημιουργήσετε ένα απλό σύστημα τραπεζικού λογαριασμού. Το σύστημα πρέπει να υποστηρίζει δύο τύπους λογαριασμών: `***SavingsAccount***` και `***CheckingAccount***`. Και οι δύο τύποι λογαριασμών πρέπει να έχουν κοινές λειτουργίες όπως `***deposit()***`, `***withdraw()***` και `***get_balance()***`, αλλά κάθε τύπος λογαριασμού έχει κάποια μοναδικά χαρακτηριστικά.

1. Ορίστε μια βασική κλάση με το όνομα `***BankAccount***` με τις ακόλουθες μεθόδους:

- `***__init__(self, account_number, initial_balance)***`: Αρχικοποιεί τον λογαριασμό με έναν αριθμό λογαριασμού και έναν αρχικό υπόλοιπο.

- `***deposit(self, amount)***`: Προσθέτει το καθορισμένο ποσό στο υπόλοιπο του λογαριασμού.

- `***withdraw(self, amount)***`**:** Αφαιρεί το καθορισμένο ποσό από το υπόλοιπο του λογαριασμού.

- `***get_balance(self)***`: Επιστρέφει το τρέχον υπόλοιπο του λογαριασμού.

2. Δημιουργήστε μια υποκλάση με το όνομα `***SavingsAccount***` που κληρονομεί από την `***BankAccount***`. Προσθέστε τα ακόλουθα χαρακτηριστικά:

- Μια μεταβλητή κλάσης με το όνομα `***interest_rate***` που είναι ίση με 0.05 (5%).

- Αντικαταστήστε τη μέθοδο `***get_balance()***` για να υπολογίζει και να επιστρέφει το υπόλοιπο συν το επιτόκιο που έχει αποκτηθεί βάσει του επιτοκίου.

3. Δημιουργήστε μια άλλη υποκλάση με το όνομα `***CheckingAccount***` που κληρονομεί από την `***BankAccount***`. Προσθέστε τα ακόλουθα χαρακτηριστικά:

- Μια μεταβλητή κλάσης με το όνομα `***transaction_fee***` που είναι ίση με 1.0.

- Αντικαταστήστε τη μέθοδο `***withdraw()***` για να αφαιρεί το τέλος συναλλαγής από το υπόλοιπο του λογαριασμού.

4. Πραγματοποιήστε διάφορες λειτουργίες όπως καταθέσεις, αναλήψεις και ερωτήσεις υπολοίπου και παρατηρήστε τα αποτελέσματα.

ΚΑΛΗ ΜΕΛΕΤΗ!
