Εισαγωγή στον Αντικειμενοστρεφή προγραμματισμό - OOP

Σημειώσεις μαθήματος

Ενότητα 6

Δρ.Ν.Μανδέλλος, Δρ.Φ.Δογάνης

Μάθημα 6 - Βασικά

Βασικές έννοιες και ορισμοί.

Απλά παραδείγματα.

Άσκηση.

Μάθημα 6 - Βασικά

- Ο Αντικειμενοστρεφής Προγραμματισμός (Object-Oriented Programming ή σύντομα OOP) είναι ένα μοντέλο ανάπτυξης λογισμικού όπου το πρόγραμμα οργανώνεται γύρω από αντικείμενα (objects).

Κάθε object:

- έχει **ιδιότητες (properties)**,
- έχει **λειτουργική συμπεριφορά (methods)**,
- συνεργάζεται με άλλα αντικείμενα.

Όπως και οι υπόλοιπες scripting languages όλα στην python είναι objects.

Μάθημα 6 - Βασικά

OBJECT

Ένα **object** στην python είναι μια οντότητα με:

- Properties (ιδιότητες/δεδομένα)
- Methods (λειτουργίες)
- Constructor (__init__) -> Αρχικοποίηση

Σε σχέση με άλλες γλώσσες διαφοροποιείται επειδή δεν υπάρχει εγγενώς ορισμένη η έννοια των event (συμβάντων) και του Destructor (καταστροφή του αντικειμένου). Να σημειωθεί ότι μπορούμε να κατασκευάσουμε objects που έχουν events & destructor, αλλά θα πρέπει να ακολουθήσουμε συγκεκριμένη μεθοδολογία.

Μάθημα 6 - Βασικά

Για να κατανοήσουμε την έννοια του **αντικειμένου** ας δούμε το εξής απλοποιημένο παράδειγμα:

Το πρόγραμμα μας είναι ας υποθέσουμε ένα simulation ενός αυτοκινήτου.

Το αυτοκίνητο είναι ένα αντικείμενο και έχει όλα τα χαρακτηριστικά που είπαμε προηγουμένως:

1. Properties

Κατάσταση:

- Ταχύτητα
- Κίνηση (μπροστά / πίσω / σταθμευμένο / ακίνητο)
- Σχέσεις μετάδοσης
- Χιλιόμετρα
- Στάθμη καυσίμου
- Άλλες ιδιότητες

Χαρακτηριστικά:

- Χρώμα
- Τύπος καυσίμου
- Τύπος οχήματος
- Τύπος κινητήρα
- Πλήθος θυρών
- Άλλες ιδιότητες

Συντήρηση:

- Αλλαγή λαδιών (περίοδος)
- Έλεγχος φρένων (περίοδος)
- Έλεγχος ελαστικών (χιλιόμετρα)
- Ετήσιος έλεγχος (ημερομηνία)
- Έλεγχος διεύθυνσης (χιλιόμετρα)
- Άλλες ιδιότητες

Μάθημα 6 - Βασικά

2. Methods

Μέθοδοι που επηρεάζουν την Κατάσταση:

- Αύξηση/Μείωση ταχύτητας
- Αλλαγή κίνησης σε όπισθεν
- Αλλαγή σχέσης
- Υπολογισμός διανυθέντων χιλιομέτρων
- Αναπλήρωση καυσίμων
- Άλλες μέθοδοι

Μέθοδοι που επηρεάζουν την κατάσταση συντήρησης:

- Αλλαγή λαδιών
- Εκτέλεση έλεγχου φρένων σε συνεργείο
- Εκτέλεση έλεγχου ελαστικών σε συνεργείο
- Εκτέλεση ετήσιου έλεγχου σε συνεργείο
- Εκτέλεση έλεγχου διεύθυνσης σε συνεργείο
- Άλλες μέθοδοι

Μάθημα 6 - Βασικά

3. Constructor - αρχικοποίηση

Η αρχικοποίηση επηρεάζει όλες τις καταστάσεις.

Παράδειγμα:

Μόλις έχουμε αγοράσει ένα καινούργιο αυτοκίνητο. Αυτό σημαίνει ότι τα χιλιόμετρα που έχει διανύσει είναι μηδέν, το όχημα είναι ακινητοποιημένο, δεν έχει γίνει κανένας έλεγχος συντήρηση (βρίσκεται στις εργοστασιακές ρυθμήσεις), οι δομικές του ιδιότητες είναι αυτές που έχει από το εργοστάσιο (χρώμα, τύπος, κλπ).

Μάθημα 6 - Βασικά

Ιεραρχία αντικειμένων

Το αντικείμενο του παραδείγματος μας, το αυτοκίνητο, αποτελείται από πολλά άλλα αντικείμενα τα οποία θα μπορούσαν να προστεθούν ως ιδιότητες (properties).

Ένα όχημα έχει ταχύμετρο, τιμόνι, σύστημα διεύθυνσης, τροχούς, κλπ.

Ταχύμετρο: Το ταχύμετρο έχει και αυτό ιδιότητες, μεθόδους και αρχικοποίηση.

Παράδειγμα:

Ιδιότητες: χιλιομετρητής, μέση ταχύτητα, μέγιστη ταχύτητα, στιγμιαία ταχύτητα. Είναι φωτειζόμενο, είναι μηχανικό, κλπ.

Μέθοδοι: Μέτρησε την στιγμιαία ταχύτητα, υπολόγισε την μέση ταχύτητα, reset τον χιλιομετρητή, κλπ.

Μάθημα 6 - Βασικά

Ιεραρχία αντικειμένων

Έτσι κάθε αντικείμενο που είναι ιδιότητα του αντικειμένου «Αυτοκίνητο» έχει ιδιότητες και μεθόδους. Το ερώτημα που τίθεται εδώ είναι πως επικοινωνούν μεταξύ τους τα αντικείμενα;

Η επικοινωνία των αντικειμένων ακολουθεί τους εξής κανόνες:

Κάθε αντικείμενο που βρίσκεται ψηλότερα στην ιεραρχία μπορεί να καλεί μεθόδους και να έχει πρόσβαση στις ιδιότητες των αντικειμένων που βρίσκονται χαμηλότερα στην ιεραρχία. Έτσι το αυτοκίνητο μπορεί να γνωρίζει την στιγμιαία ταχύτητα που αναγράφεται στο ταχύμετρο, να γνωρίζει ποιά είναι η στάθμη των καυσίμων, αν ο κινητήρας είναι ανοικτός, κλπ. Αντίθετα το ταχύμετρο δεν επιτρέπεται να γνωρίζει αν το όχημα έχει καύσιμα, ή αν ο κινητήρας είναι ανοικτός.

Οι ιδιότητες αυτές, που μπορούν να αναγνωστούν από αντικείμενα υψηλότερα στην ιεραρχία ονομάζονται **global**. Το ίδιο και οι μέθοδοι που μπορεί να κληθούν από αντικείμενα που βρίσκονται υψηλότερα στην ιεραρχία.

By default στην python όλες οι ιδιότητες και οι μέθοδοι είναι **public**. Μπορούμε όμως να αποκρύψουμε μεθόδους και ιδιότητες ώστε να μην υπάρχει πρόσβαση από άλλα αντικείμενα εαν τα σημειώσουμε ως **private**.

Class

Η πιο απλή δομή αντικειμένου στην Python είναι το να κατασκευάσει κάποιος ένα enumeration:

Παράδειγμα οι καταστάση κίνησης ενός οχήματος:

```
from enum import Enum

class CarState(Enum):
    STOPPED = 0
    NOT_RUNNING = 1
    FORWARD = 2
    BACKWARD = 3
    UNKNOWN = 4
```

Enumerations

Έτσι στο παραπάνω παράδειγμα έχουμε ορίσει ένα αντικείμενο που περιγράφει όλες τις δυνατές καταστάσεις ενός οχήματος. Έτσι μπορούμε να το χειριστούμε ως εξής:

```
carState = CarState.PARKED


action = float(input("Select an action: 1= stop at the gas station, 2 = stop at traffic lights, 3 = move forward, 4 = move backward"))
If (action == 0):
carState = CarState.STOPPED
elif (action == 2):
    carState = CarState.NOT_RUNNING
elif (action == 3):
    carState = CarState.FORWARD
elif (action == 4):
    carState = CarState.BACKWARD
else:
    carState = CarState.UNKNOWN
```

Class

Ορισμός μιας απλής κλάσης

```
class Car:
    #Constructor
    def __init__(self, velocity):
        self.velocity = velocity   #global property

#global function with arguments
    def set_velocity(self, velocity):
        self.velocity = velocity

#global function
    def report(self):
        if self.velocity > 0:
            print(“car is moving")
        else:
            print(“car is stopped")
```

Class

Εσωτερική μεταβλητή self

Το self είναι ένα όρισμα που αντιπροσωπεύει –όπως λέει και η λέξη- το object. Στην ουσία μόλις η κλάση έχει δημιουργηθεί το self είναι η μεταβλητή που απεικονίζει την κλάση.

Έτσι όταν ορίζουμε μια ιδιότητα (property) στον constructor της κλάσης, η τιμή της αρχικοποιείται με το self

Το ίδιο συμβαίνει όταν ορίσουμε μια συνάρτηση στην κλάση.

Ο παρακάτω κώδικας είναι λάθος:

```
class car:
def __init__(self, velocity):
        self.velocity = velocity   #global property

def set_velocity(self, argument):
    velocity = argument; # η μεταβλητή velocity είναι άγνωστη!
```

Διόρθωση:

```
def set_velocity(self, argument):
    self.velocity = argument;
```

Class

Χρήση της κλάσης

 # αρχικοποίηση της κλάσης car με ταχύτητα μηδέν

car = Car(0)

#ορίστε ταχύτητα 50χλμ/ω

car.set_velocity(50)

#εκτύπωση αναφοράς

car.report()

Enumerations

Ένα πιο σύνθετο παράδειγμα

```
from enum import Enum
class CarState(Enum):
    STOPPED = 0
    NOT_RUNNING = 1
    FORWARD = 2

class EngineState(Enum):
    OFF = 0
    ON= 1

def calculate_car_state(velocity, engineState):
     state = CarState.STOPPED

     if (velocity > 0):
          state = CarState.FORWARD
    else:
          if (engineState == EngineState.OFF):
                state = CarState.STOPPED
          else:
                state = CarState.NOT_RUNNING
```

Χρήση:

```
v = input("What is your velocity?")
engineState = EngineState.OFF
if (v == 0.0):
      isOn = input("The engine is on?")
      if (isOn):
            engineState = EngineState.ON
else:
      engineState = EngineState.ON

Print('The car is {calculate_car_state(v, engineState)}')
```

Μορφοποίηση μηνυμάτων – print

Άσκηση:

Να φτιαχτεί ένας απλός simulator αυτοκινήτου. Ο χειριστής θα μπορεί να ανοίξει ή να κλείσει τη μηχανή. Επίσης θα μπορεί να αυξήσει ή μειώσει την ταχύτητα του οχήματος.

Σε κάθε περίπτωση να ορίσετε τα σωστά μηνύματα στον χρήστη.

Συνέχεια της άσκησης στο σπίτι:

Προσθέστε νέες λειτουργίες.

Άσκηση

Ανάλυση:

Θα χρειαστώ δυο κλάσεις: Έναν enumerator για το state της μηχανής και μια κλάση για το αυτοκίνητο (object car).

Το car θα έχει τις εξής ιδιότητες:

engine (κατάσταση: ανοικτό ή κλειστό)

velocity (ταχύτητα)

Και τις εξής μεθόδους:

start_car, stop_car (ανοίγει – κλείνει τη μηχανή)

increase_speed / decrease_speed (αυξάνει - μειώνει την ταχύτητα)

Άσκηση

Ορισμός των κλάσεων

```
class EngineState:
    ON = 0
    OFF = 1


class Car:
#Constructor
    def __init__(self, velocity):
        self.velocity = velocity
        self.engine = EngineState.OFF
# start engine
    def start_car(self):
        self.engine = EngineState.ON
```

```
# stop engine
    def stop_car(self):
        self.engine = EngineState.OFF


# increase speed
    def increase_speed(self, increment):
        self.velocity += increment


# decrease speed
    def decrease_speed(self, increment):
        self.velocity -= increment
```

```
# provide messages
    def report(self):
        if (self.engine == EngineState.ON):
            if (self.velocity > 0):
                return f"I am moving, velocity {self.velocity}"
            else:
                return "I am stopped, engine on"
        elif (self.engine == EngineState.OFF):
            if (self.velocity > 0):
                return "Alarm, accident detected"
            else:
                return "I am stopped"
```

Άσκηση

Κώδικας που καλεί τις κλάσεις:

```
car = Car(0);
print(car.report())


print("/n")
print("Provide the driver an action: 0 = exit, 1= start engine, 2 = stop engine, 3 = increase speed, 4 = decrease speed \n")


while True:
    user_action = input()
    action = int(user_action)
    if (action == 0):
        break
    elif (action == 1):
        car.start_car()
    elif (action == 2):
        car.stop_car()
    elif (action == 3):
        user_inc = input("how much to increase speed? \n")
        inc = int(user_inc)
        car.increase_speed(inc)
    elif (action == 4):
        user_inc = input("how much to decrease speed? \n")
        inc = int(user_inc)
        car.decrease_speed(inc)
    else:
        print("unknown action, repeat")


print(car.report())
```

Άσκηση

Ένα πιο ολοκληρωμένο παράδειγμα για την κλάση car με περισσότερα μηνύματα:

```
class Car:
    def __init__(self, velocity):
        self.velocity = velocity
        self.engine = EngineState.OFF

    def start_car(self):
        if (self.engine == EngineState.ON):
            print("WARNING: engline is already on!")
            return

        if (self.velocity > 0):
            print("WARNING: back to safety!")

        self.engine = EngineState.ON

    def stop_car(self):
        if (self.engine == EngineState.OFF):
            print("WARNING: engline is already off!")
            return
        self.engine = EngineState.OFF
        if (self.velocity > 0):
            print("CRITICAL: You have switch off the engline while running")

    def increase_speed(self, increment):
        if (self.engine == EngineState.OFF):
            print("WARNING: I cannot increase speed, The engine is off!")
        else:
            self.velocity += increment

    def decrease_speed(self, increment):
        if (self.engine == EngineState.OFF):
            print("WARNING: I cannot decrease speed, The engine is off!")
        else:
            if self.velocity == 0:
                print("WARNING: I cannot decrease speed, The car is already stopped!")
            else:
                self.velocity -= increment

        if self.velocity < 0:
            self.velocity = 0;
            print("This car cannot move in a negative speed. The car is stopped!")

    def report(self):
        if (self.engine == EngineState.ON):
            if (self.velocity > 0):
                return f"I am moving, velocity {self.velocity}"
            else:
                return "I am stopped, engine on"
        elif (self.engine == EngineState.OFF):
            if (self.velocity > 0):
                return "Alarm, accident detected"
            else:
                return "I am stopped"
```