Технически Университет – София,
Факултет „Компютърни системи и технологии“

Упражнение 6 – Създаване на програми с класове и обекти. Наследяване

Какво представляват класовете и обектите ?

Python е обектно-ориентиран език. Това означава, че в него се използват класове и обекти. Най-общо казано обектите в Python представляват модели на реални обекти от действителността. Обектите могат да включват свойства и методи. Класовете пък могат да се разглеждат като „матрици” за производство на обекти.

Как се създават и използват класове и обекти в Python ?

За да създадем клас в Python използваме ключовата дума class. Например следният код: **`class MyFirstClass:`**

```
    x=5
```

създава клас с име MyFirstClass и едно свойство x, инициализирано със стойност 5 . Тук **x** е свойство(атрибут) на класа и неговата стойност е една и съща за всички инстанции на този клас .

За да създадем обект, базиран на този клас, използваме израза

```
MyFirstObject= MyFirstClass()
```

Атрибутите на класа достъпваме чрез използване на името на класа и името на атрибута, разделени с точка:

```
print(MyFirstClass.x)
```

Атрибутите на класа можем да достъпваме и чрез използване на обекта и името на атрибута, разделени с точка:

```
print(MyFirstObject.x)
```

Всеки клас в Python има вградена функция конструктур __init__(), която се изпълнява автоматично винаги, когато класът се инициализира, т. е. когато се създава обект на базата на класа. Тази функция може да се използва, за да

задаваме стойности на свойства при инициализация или да извършваме други действия, които са необходими при създаването на обектите.
Например за класа Person може да предефинираме функцията __init__(), така че тя да задава стойности на полетата name и age по следния начин:

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

MyPerson = Person("Ivan", 35)
print(MyPerson.name)
print(MyPerson.age)
```

Инстанциите на един клас могат да имат атрибути(свойства) ,като за всеки всеки обект стойността на атрибутите е различна. В горния пример това са **self.name** и **self.age.**
Атрибутите и методите на класа могат да бъдат публични , както в нашия пример и частни . Публичните методи могат да бъдат извикани , а към публичните атрибути можем да се обръщаме(да четем и да променяме стойностите им) извън пределите на класа. Частния метод може да бъде извикан само вътре в класа- в други негови методи, а към частния атрибут не можем да се обърнем пряко (не можем да го четем или да го променяме). За да бъде дефиниран като частен пред името на метода или атрибута се поставя два знака долна черта.
Пример:

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
        self.__w=10    #частен атрибут
```

Ако се опитате да се обърнете към частен атрибут в кода на програмата извън рамките на класа , ще получите съобщение за грешка.
За управлението на достъпа до частни атрибути се използват специални методи- за четене стойността на такъв частен атрибут (getter) и за модифициране стойността на такъв атрибут(setter)

Пример :

```
class Car:
    car_type = "sports"             # class attribute
    def __init__(self, color): # __init__дефинира конструктора на класа. self е препратка към текущия обект (инстанция), а color е параметър, който се подава при създаване на обекта.
        self.__color = color  # създава частен инстантен атрибут на конкретната инстанция и му присвоява стойността на color
    def print_car(self):            # дефинира метода print_car
        print(self.__color, "\t", self.car_type) #отпечатва стойностите на color и car_type
    def get_color(self):            # дефинира метод (getter), който приема self (инстанцията)
        return self.__color         # връща стойността на частния атрибут __color
    def set_color(self, color):     # дефинира методът setter. Приема стойност (color) и я записва в частния инстантен атрибут __color на текущия обект (self)
        self.__color = color
car2 = Car("yellow")        # създава нов обект car2 от класа Car и подава стойност "yellow" за параметъра color
car2.print_car()            # отпечатва двата атрибута yellow    sports
car2.color = "black"        # добавя нов публичен атрибут color в обекта car2. Той не засяга частния атрибут __color. Методи, които ползват __color, ще виждат стария цвят
car2.print_car()            # пак ще покаже yellow    sports
# print(car2.__color)     # това ще вдигне AttributeError (няма достъп отвън)
print(car2.get_color())     # ще отпечата стойността на частния атрибут __color

car2.set_color("green")     # setter променя частния атрибут __color
print(car2.get_color())     # getter връща текущата стойност green
car2.print_car()            # методът чете частния атрибут __color (сменен от setter-а на "green") и car_type ("sports") и връща резултат green    sports
```

Освен атрибути, обектите могат да съдържат и методи. Методите на обектите са функции, които принадлежат на обектите. Например за създадения вече клас може да създадем метод **greetings**, който отпечатва поздравление от съответното лице:

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

```
   def greetings(self):
      print("Hello, my name is " + self.name)
MyPerson = Person("Ivan", 35)
MyPerson.greetings()
```

Параметърът **self** е референция към текущия обект (инстанция) инстанция на класа. Използва се за достъп до свойствата и методите, принадлежащи на обекта. Не е задължително да се нарича **self,** но по конвенция се използва това име. Важното е да е първият параметър на всеки метод на обектите на класа.

Може да се дефинират и методи на класа, които не са обвързани с инстанциите на класа. Можем да извикваме такъв метод без да сме създали обекти от този клас, за целта се използва името на класа оператор **.** и име на метода.

Пример :

```
class FirstClass:
    x = 5
    def FirstClassMethod():
        print("This is class method")
FirstClass.FirstClassMethod()
```

Можем да **променяме стойности** на свойства на обекти, например

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

Myperson = Person("Ivan", 40)  # името на обекта е Myperson
Myperson.age = 41              # сменяме атрибута на ТАЗИ инстанция
print(Myperson.age)            # -> 41
```

Можем също да изтриваме свойства на обекти с ключовата дума del.

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

MyPerson = Person("Ivan", 35)
```

```
print(MyPerson.age)   # 35

del MyPerson.age      # изтриваме атрибута age САМО от този обект
# След изтриването достъпът до age вдига грешка:
print(MyPerson.age)   # AttributeError: 'Person' object has no attribute 'age'
```

Можем да изтриваме и цели обекти, отново с ключовата дума del.

```
class Person:
   def __init__(self, name):
      self.name = name
MyPerson = Person("Ivan")
print(MyPerson.name)   # Ivan
del MyPerson          # изтрива името MyPerson (прекъсваме препратката)
print(MyPerson)       # NameError: name 'MyPerson' is not defined
```

Дефиницията на клас не може да е празна, но ако по някакава причина искаме да оставим клас без съдържание, може да използваме ключовата дума pass.

```
class Person:
    pass
```

Какво представлява наследяването?

Наследяването е подход, който позволява един клас да наследи всички методи и свойства на друг клас. Наследяваният клас се нарича клас-родител. Наследяващият клас се нарича клас-дете.

Всеки клас може да бъде родителски. Създаването на родителски клас е същото, като на всеки друг клас.

Като пример създаваме клас, наречен Person, който има свойства firstname (малко име) и lastname (фамилно име) и метод printname:

```
class Person:
   def __init__(self, fname, lname):
      self.firstname = fname
      self.lastname = lname
   def printname(self):
      print(self.firstname, self.lastname)
```

```
p = Person("Ivan", "Ivanov")
p.printname()
```

Може да използваме този клас, за да създадем обект и да извикаме метода printname.

```
MyPerson=Person("Ivan","Petrov")
MyPerson.printname()
```

За да създадем клас, който наследява даден клас, подаваме родителския клас като параметър при създаването на класа-наследник.

Например, можем да създадем клас Student, който наследява методите и атрибутите на класа Person.

```
class Student(Person):
    pass
```

Тук е подадена ключовата дума pass, защото за момента не е необходимо да се добавят нови свойства или методи към този клас. Така създаден, класът наследник има същите свойства и методи като родителския клас. За него например може да създадем обект и да извикаме метода printname.

```
class Person:
   def __init__(self, fname, lname):
      self.firstname = fname
      self.lastname = lname
   def printname(self):
      print(self.firstname, self.lastname)
class Student(Person):
   pass
MyStudent = Student("Petar", "Vasilev")
MyStudent.printname()  # Petar Vasilev
```

В класа-наследник можем да предефинираме функцията __init__(). Тогава класът-наследник вече не наследява тази функция от родителя, а има свой собствен вариант за нея. Например, за да предефинираме в класа Student функцията __init__(), така че тя да прави същото като в родителския клас, може да използваме един от двата варианта:

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)
class Student(Person):
    def __init__(self, fname, lname):
        Person.__init__(self, fname, lname)   # извиква конструктора на родителя Person, за да инициализира firstname и lastname за този обект.
# тестване
s = Student("Petar", "Vasilev")
s.printname()   # Petar Vasilev
```

или

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)
class Student(Person):
    def __init__(self, fname, lname):
        super().__init__(fname, lname)   # по-чисто и правилно в Python 3
s = Student("Petar", "Vasilev")
s.printname()   # Petar Vasilev
```

Като използваме функцията super(), даваме указание на класа- наследник автоматично да взима наследените от родителя методи и свойства.

Наследяващият клас може да има свои добавени свойства и методи. Например към класа Student може да се добави свойство **graduationyear**.

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
```

```
        print(self.firstname, self.lastname)
class Student(Person):
    def __init__(self, fname, lname, graduationyear):
        super().__init__(fname, lname)   # викаме конструктора на базовия клас
        self.graduationyear = 2019
MyStudent = Student("Petar", "Vasilev", 2019)
MyStudent.printname()              # -> Petar Vasilev
print(MyStudent.graduationyear)    # -> 2019
```

За да се вземе под внимание новото свойство при инициализация, може да се промени методът __init__(), така че graduationyear да се задава при инициализация.

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)
class Student(Person):
    def __init__(self, fname, lname, year):
        super().__init__(fname, lname)
        self.graduationyear = year
MyStudent = Student("Petar", "Vasilev", 2019)
MyStudent.printname()              # -> Petar Vasilev
print(MyStudent.graduationyear)    # -> 2019
```

Така при създаването на обекта се подава коректната година.

Към наследяващия клас може да добавим нови методи. Например, към класа Student може да добавим метода welcome:

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)
class Student(Person):
```

```
    def __init__(self, fname, lname, year):
        super().__init__(fname, lname)
        self.graduationyear = year
    def welcome(self):
        print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)
MyStudent = Student("Petar", "Vasilev", 2019)
MyStudent.printname()             # -> Petar Vasilev
print(MyStudent.graduationyear)    # -> 2019
MyStudent.welcome()               # -> Welcome Petar Vasilev to the class of 2019
```

При добавяне на нови методи към класа-наследник с имена, съвпадащи с имена на методи в родителя, родителският метод се припокрива (override) от метода на наследника.

Специални методи :

Класовете в Python поддържат специални методи , които започват с две подчертаващи черти (вече разгледахме един от тези специални методи конструктор __init__()).

Ще обърнем внимание на специалния метод **_ _str_ _** . Използва се при преобразуване на обекта в низ. Извиква се при опит обектът да бъде преобразуван в низ, когато се извежда чрез функция **`print().`**

Пример:

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def __str__(self):
        return f"{self.firstname} {self.lastname}"
person1 = Person("Polina", "Koleva")
print(person1)       # подаваме обект на функция print  Polina Koleva
```

Задачи :

1. Да се напише код на Python, който дефинира клас Person с полета име (name), фамилия (family), възраст (age), националност (nationality). Да се дефинира конструктор, който инициализира полетата на класа. Да се добави метод print, който отпечатва имената и националността на съответното лице. Да се създадат обекти-инстанции на класа. За тях да се извика методът print.

2. Да се добави към кода от Задача 1 клас Student, наследник на класа Person с две нови полета – университет (university) и година на обучение (yearofstudy). Да се предифинира за него методът print, така, че да отпечатва и тези две полета. Да се създадат инстанции на новия клас и за тях да се извика методът print.

3. Да се добави към кода от Задача 1 и Задача 2 клас Lecturer, наследник на класа Person с две нови полета – университет (university) и опит (experience) – брой години преподавателски стаж. Да се предифинира за него методът print, така, че да отпечатва тези две полета. Да се създадат инстанции на новия клас и за тях да се извика методът print.