# Лабораторно упражнение №7

# Обектно-ориентирано програмиране в Python. Класове, обекти, наследяване.

Целта на упражнението е да демонстрира средствата на програмния език Python за реализация на основните принципи на обектно-ориентираното програмиране: капсулиране на данни, наследяване на класове, припокриване на методи.

Основни понятия:

- Class - шаблон, който може да се използва за конструиране на обект. Определя атрибутите и методите, които ще съставят обекта.
- Атрибут (attribute) - променлива, която е част от клас.
- Клас наследник (child class) - нов клас, създаден при разширяване на родителския клас. Новият клас наследява всички атрибути и методи на родителския клас.
- Конструктор (constructor) - незадължителен специално именуван метод (__init__), който се извиква при създаване на обект в клас. Обикновено се използва за задаване на начални стойности за обекта.
- Деструктор (destructor) - незадължителен специално именуван метод (__del__), който се извиква преди унищожаване на обект. Рядко се използват.
- Наследяване (inheritance) – създаване на нов клас (наследник) чрез разширяване на съществуващ клас. Класът наследник има всички атрибути и методи на родителя плюс допълнителни атрибути и методи, определени от класа наследник.
- Родителски клас - класът, който се разширява, за да създаде нов клас наследник.
- Метод - функция, която се съдържа в клас и обектите, които са конструирани от класа.
- Обект – екземпляр (инстанция) от клас. Обектът съдържа всички атрибути и методи, които са определени от класа.

## 1. Синтаксис на дефиницията на клас

Най-простата форма на дефиниция на клас изглежда така:

```
class ClassName:
    <оператор-1>
    ...
    <оператор-N>
```

## 2. Клас обекти

Клас обектите поддържат два вида операции: обръщения към атрибути (attribute references) и създаване на инстанции (instantiation).Обръщенията към атрибути използват стандартния синтаксис, употребяван за всички обръщения към атрибути в Python — ***обект.име***. Валидни имена на атрибути са всички онези, които са се намирали в пространството на имената на класа, когато клас обектът е бил създаден. Така че ако дефиницията на класа е изглеждала по този начин,

***Пример 1:***

```
class MyClass:
    "Един прост примерен клас"
    i = 12345def
    f(x):
        return ’Успех’
```

то **MyClass.i** и **MyClass.f** са валидни обръщения към атрибути, които съответно връщат цяло число и обект от тип метод. Атрибутите на класовете също могат да бъдат присвоявани, така че може да се промени стойността на **MyClass.i** чрез присвояване. (**__doc__**също е валиден атрибут, който връща документационния символен низ, принадлежащ на класа: "Един прост примерен клас"). При създаването на инстанция на клас се използва нотацията на функция. Клас обектът е функция без параметри, която връща нова инстанция на класа. Например:

```
x = MyClass()
```

създава нова инстанция на класа и присвоява този обект към локалната променлива **x**.

Операцията за създаване на инстанция създава празен обект. За предпочитане е повечето класове да създават обекти в определено начално състояние. Затова класът може да дефинира специален метод, наречен **__init__():**

```
def __init__(self):
    self.data = []
```

Когато даден клас дефинира метод **__init__()**, тогава при създаването на инстанцията автоматично се извиква__init__() за новосъздадената инстанция на класа. В горния пример една нова инициализирана инстанция може да бъде получена чрез:

```
x = MyClass()
```

Методът__init__()може да има аргументи. Аргументите, подадени на оператора за създаване на инстанция на класа, се предават на__init__().

***Пример 2:***

```
>>> class Complex:
        def __init__(self, realpart, imagpart):
            self.r = realpart
            self.i = imagpart
>>> x = Complex(3.0,-4.5)
>>> print(x.r, x.i)
```

### 3. Обекти-инстанции

Съществуват два вида валидни имена на атрибути - данни и методи. Атрибутите-данни няма нужда да се декларират; подобно на локалните променливи, те се появяват, когато за първи път им се присвои стойност.

Вторият вид обръщения към атрибутите са методите. Методът е функция, която "принадлежи на" даден обект. (В Python, терминът "метод" не е характерен само за инстанциите на класове — други типове обекти също могат да притежават методи. Например обектите от тип списък притежават методи - append, insert, remove, sort и др.). Валидните имена на методи за даден обект-инстанция зависят от неговия клас. По дефиниция всички атрибути на един клас, които са обекти от тип функция, дефинират съответните методи на инстанциите на класа. Например **x.f** е валидно обръщение към метод, тъй като **MyClass.f** е функция, а **x.i** не е, тъй като **MyClass.i** не е. Но **x.f** не е същото като **MyClass.f** — то е обект от тип метод, а не обект от тип функция.

### 4. Обекти от тип метод

Обикновено метод се вика непосредствено, например: x.f(). В горния пример той ще върне символния низ 'Успех'.

***Пример 4:***

```
xf = x.f
print x.f()
```

### 5. Помощна документация за клас

Функциите type и dir могат да се използват за класове.

***Пример 5:***

```
class PartyAnimal:
    x = 0
    def party(self) :
        self.x = self.x + 1
        print("So far",self.x)
```

```
an = PartyAnimal()
print ("Type", type(an))
print ("Dir ", dir(an))
print ("Type", type(an.x))
print ("Type", type(an.party))
```

Чрез функцията dir може да се види както атрибута x integer, така и метод в обекта.

Изход:

***Резултат:?***

## 6. Жизнен цикъл на обекта

В предишните примери се дефинира клас (шаблон), който се използва за създаване на екземпляр (обект) от този клас и след това се използва инстанцията. Когато програмата приключи, всички променливи се унищожават. Препоръчително е унищожаване на обекта след приключване на работата с него.

Към обекта се добавят специално посочени методи:

***Пример 6:***

```
class PartyAnimal:
    x = 0
    def __init__(self):
        print('I am constructed')
    def party(self) :
        self.x = self.x + 1
        print('So far',self.x)
    def __del__(self):
        print('I am destructed', self.x)
an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)
```

Изход:

***Резултат:?***

Когато Python създава обекта, той извиква метод **__init__**, за да се зададат някои стойности по подразбиране или първоначални стойности за обекта. Когато Python срещне оператора:

```
an = 42
```

, той всъщност „откъсва нашия обект", така че може да използва повторно променливата, за да запази стойността 42. Точно в момента, когато обекта се „унищожава", се извиква кодът на деструктора (**__del__**). Не може да се спре променливата да бъде унищожена.

При разработването на обекти е добре да се добавя конструктор към обект, за да се зададат начални стойности за обекта. Сравнително рядко е необходим деструктор за даден обект.

## 7. Множество инстанции

Когато се конструират няколко обекта от клас, могат да се зададат различни начални стойности за всеки от обектите. Може да се предават данни на конструкторите, за да се даде на всеки обект различна начална стойност:

***Пример 7:***

```
class PartyAnimal:
    x = 0
    name = ''
    def __init__(self, nam):
        self.name = nam
        print(self.name,'constructed')
    def party(self) :
        self.x = self.x + 1
        print(self.name,'party count',self.x)
s = PartyAnimal('Sally')
j = PartyAnimal('Jim')
s.party()
j.party()
s.party()
```

Конструкторът има както самостоятелен параметър, който сочи към обекта на обекта, така и допълнителни параметри, които се предават в конструктора, докато обектът е конструиран:

```
s = PartyAnimal('Sally')
```

В конструктора вторият ред копира параметъра (nam), който се предава в атрибута name в рамките на обекта на обект.

```
self.name = nam
```

Резултатът от програмата показва, че всеки от обектите (s и j) съдържа собствени независими копия на x и nam:

***Резултат:?***

## 8. Наследяване

Синтаксисът за дефиниция на наследен клас изглежда така:

```
class DerivedClassName(BaseClassName):
    <оператор 1>
    .
    .
```

.

<оператор N>

Името BaseClassName трябва да е дефинирано в областта на видимост на наследения клас. Допустим е и израз вместо име на базов клас. Това е полезно, когато базовият клас е дефиниран в друг модул, например:

class DerivedClassName(modname.BaseClassName):

Друга мощна характеристика на обектно-ориентираното програмиране е възможността за създаване на нов клас чрез разширяване на съществуващ клас. Когато се разширява клас, оригиналния клас се нарича родителски клас, а новия клас - наследник.

За пример ще се премести клас PartyAnimal в собствен файл. След това може да се "импортира" класа PartyAnimal в нов файл и да се разшири, както следва:

***Пример 8:***

```
from party import PartyAnimal
class CricketFan(PartyAnimal):
    points = 0
    def six(self):
        self.points = self.points + 6
        self.party()
print(self.name,"points",self.points)
s = PartyAnimal("Sally")
s.party()
j = CricketFan("Jim")
j.party()
j.six()
print(dir(j))
```

Когато се дефинира класа CricketFan, се показва, че се разширява класа PartyAnimal. Това означава, че всички променливи (x) и методи (party) от класа PartyAnimal се наследяват от клас CricketFan.

Докато програмата се изпълнява, се създават s и j като независими случаи на PartyAnimal и CricketFan. Обектът j има допълнителни възможности извън s-обекта.

***Резултат:?***

```
Sally constructed
Sally party count 1
Jim constructed
Jim party count 1
Jim party count 2
Jim points 6
['__class__', '__delattr__', ... '__weakref__',
        'name', 'party', 'points', 'six', 'x']
```

В изхода dir за j обекта (екземпляр от клас CricketFan) се вижда, че той има атрибутите и методите на родителския клас, както и атрибутите и методите, които се добавят при разширяването на класа за създаване на клас CricketFan.

## 9. Множествено наследяване

Python поддържа и една ограничена форма на множествено наследяване. Дефиницията на клас с няколко базови класове изглежда както следва:

```
class DerivedClassName(Base1, Base2, Base3):
<оператор-1>
.
.
.
<оператор-N>
```

При обръщения към атрибути на клас, ако даден атрибут не е намерен в DerivedClassName, за него се претърсва в Base1, после (рекурсивно) в базовите класове на Base1, и само ако атрибутът не е открит там, тогава се претърсва Base2, и така нататък.

**Самостоятелна задача:**

Създайте клас Person, който симулира опростена форма на родословно дърво.

Всеки човек трябва да има атрибути за име (name), година на раждане (birth_year) и пол (gender).

- Родители

Всеки човек може да има родители (баща и майка) също от тип Person. Минималната разлика в годините между родител и дете е 18.

- Братя и сестри

Всеки човек може да има (но не задължително) братя и сестри. Списък с братята и сестрите трябва да се връща от методите съответно get_brothers() и get_sisters(). При липсата на братя или сестри съответният метод трябва да връща празен списък.

Имплементирайте метод children, който връща списък с децата на даден човек. Добавете и незадължителен аргумент, според който методът да връща деца само от единия от двата пола. Добавете и метод is_direct_successor, който проверява дали двама души са с директна роднинска връзка и връща булева стойност.

## Допълнителни задачи

1. Напишете програма на Python, която използва клас за конвертиране на цяло число от десетична в римска бройна система.

Пример:

1 -> I

4000 -> MMMM

2. Напишете програма на Python, която използва клас за конвертиране на число от римска в десетична бройна система.

Пример:

MMMCMLXXXVI -> 3986

MMMM -> 4000

C -> 100

3. Напишете програма на Python, която чрез клас проверява съответствие на скоби '(', ')', '{', '}', '[' и ']. Скобите трябва да бъдат затворени в правилен ред, напр. "()" и "()[]{}" са валидни, а "[)", "({[)]" и "{{{" са невалидни.

Пример:

(){}[] -> True

()[{)} -> False

() -> True

4. Създайте Python клас, който извежда всички уникални подмножества на множество от неповтарящи се цели числа.

Примерен вход: [4, 5, 6]

Изход: [[], [6], [5], [5, 6], [4], [4, 6], [4, 5], [4, 5, 6]]

Решение:

5. Създайте Python клас, който намира индексите на всички двойки елементи от масив от цели числа, чиято сума е равна на въведена от потребителя стойност.

Примерен вход:

numbers= [10,20,10,40,50,60,70], target=50

Изход:

3, 4

6. Създайте Python клас, намиращ три елемента от множество от n реални числа, чиято сума е равна на нула.

Примерен вход:

Input array : [-25, -10, -7, -3, 2, 4, 8, 10]

Изход:

```
[[-10, 2, 8], [-7, -3, 10]]
```

7. Създайте Python, който реализира функцията pow(x, n).
8. Създайте Python клас, който обръща символен низ дума по дума

Примерен вход: 'hello .py'

Изход: '.py hello'

9. Създайте Python клас с два метода – get_String и print_String. Методът get_String приема символен низ от потребител, а методът print_String извежда символния низ с главни букви (upper_case).
10. Създайте Python клас Rectangle, който съдържа дължина, височина на правоъгълник и метод за изчисляване на площта на правоъгълника.
11. Създайте Python клас Circle, който съдържа радиус на окръжност и два метода за изчисляване на площта и периметъра на окръжността.