Klasy

Tworzenie klasy

Tworzenie klasy:

- wykorzystujemy instrukcję ‘class’,
- każda klasa dziedziczy pośrednio lub bezpośrednio po klasie ‘object’
- Python 3.x – dziedziczenie po klasie może być niejawne lub jawne, Python 2.x – jawne

Tworzenie klasy

Przykład 1:

```
class k1:                #domyślne dziedziczenie po ‘object’
   pass                  #instrukcja, która nic nie robi – w definicji jest wymagana
                         #jakakolwiek  instrukcja


obj_k1=k1()
print(dir(obj_k1))       #wydruk dostępnych (odziedziczonych) metod w klasie

#output:
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

Tworzenie klasy

Przykład 2:

```
class k2(object):    #jawne dziedziczenie po wbudowanej klasie ‘object’
   pass

obj_k2=k2()
print(dir(obj_k2))

#output:
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

Tworzenie klasy

Przykład 3:

```
class k3(object):
    pass

obj_k3=k3()
print(obj_k3.__doc__)    #wydruk notki dokumentacyjnej


#output:
None                     #domyślnie brak zdefiniowanej notki dokumentacyjnej
```

Tworzenie klasy

Przykład 4:

```
class k4(object):
    '''Przykładowy opis
klasy'''
    pass

obj_k4=k4()
print(obj_k4.__doc__)    #wydruk notki dokumentacyjnej

#output:
Przykładowy opis
klasy
```

Definicja metody w klasie

Przykład 5:

```
class k5:           #definicja klasy
   '''Klasa k5'''   #definicja notki inf.
   def fun(self):   #definicja metody fun – obowiązkowy pierwszy argument
      print("Metoda 'fun' klasy k5")

obj_k5=k5()    #utworzenie instancji (obiektu) klasy k5
obj_k5.fun()   #wywołanie metody

#output:
Metoda 'fun' klasy k5
```

Argument „self” :

- każda metoda danej klasy wykorzystuje **pierwszy argument** jako informację, na którym **obiekcie** tej klasy 'pracuje' (metody są 'wspólne' dla wszystkich obiektów danej klasy)
- przyjęło się używać jako pierwszy argument metody nazwę: „self” – **nie jest to obowiązek** – można używać dowolnej nazwy.

Definicja metody w klasie

Przykład 6:

```
class k6:
    def fun1(self, zmienna):
        print("fun1: metoda z argumentem '%s'" %zmienna)
    def fun2(self):
        print("fun2: metoda bez argumentu")

obj_k6=k6()

obj_k6.fun1('Hello')    #argument 'self' jest przekazywany automatycznie
obj_k6.fun2()           # i zawiera wskaźnik do obiektu 'obj_k6'

#output:
fun1: metoda z argumentem 'Hello'
fun2: Metoda bez argumentu
```

Klasy, instancje, atrybuty

Atrybuty klasy :

- wszystko w Python'ie jest obiektem – także definicja klasy; istnieje ona cały czas w pamięci i można się do niej odwoływać, tak jak do każdego innego obiektu,
- w ramach definicji klasy można definiować tzw. atrybuty (zmienne) klasy – zwykle umieszcza się je zaraz po definicji nazwy klasy (nie jest to jednak obowiązkowe),
- atrybuty klasy można także dodawać 'w locie' odwołując się do nazwy klasy.
- do atrybutów klasy mają dostęp wszystkie jej instancje (są współdzielone).

Atrybuty obiektu :

- obiekt danej klasy jest tworzony na podstawie definicji tej klasy i ma dostęp do jej atrybutów oraz metod (nie są one kopiowane tylko współdzielone),
- atrybuty danej instancji klasy (obiektu) można definiować w różny sposób i będą one tylko dostępne tylko w obrębie tej instancji,
- najczęściej atrybuty instancji definiuje się w specjalnej metodzie '__init__', która jest automatycznie uruchamiania podczas tworzenia instancji; można także tworzyć nowe atrybuty obiektu 'w locie'.

Klasy, instancje, atrybuty

Przykład 7 - atrybuty klasy i obiektu:

```
class klasa:
    ak='atrybut klasy'
    def __init__(self):
        self.ao='atrybut obiektu'

obj=klasa()
print(obj.ao)
print(obj.ak)       #obiekt ma dostęp do atrybutów klasy
print(klasa.ak)
#print(klasa.ao)  #błąd - klasa nie ma dostępu do zmiennych obiektu

klasa.ak='zmieniony atrybut klasy'
print(obj.ak)       #obiekt 'widzi' zmiany atrybutów klasy
print(klasa.ak)
```

```
#output:
atrybut obiektu
atrybut klasy
atrybut klasy
zmieniony atrybut klasy
zmieniony atrybut klasy
```

Klasy, instancje, atrybuty

Przykład 8 – atrybuty klasy:

```
class klasa:
    ak1='atrybut klasy ak1'

print(klasa.ak1)
obj=klasa()
print(obj.ak1)

klasa.ak2='atrybut klasy dodany "w locie"'
print(klasa.ak2)
print(obj.ak2)
```

```
#output:
atrybut klasy ak1
atrybut klasy ak1
atrybut klasy dodany "w locie"
atrybut klasy dodany "w locie"
```

Klasy, instancje, atrybuty

Przykład 9 – atrybuty obiektu:

```
class klasa:
   ak1='1 atrybut klasy'
   def __init__(self):
      self.ao1='1 atrybut obiektu'
      zm='Hello'          #zmienna lokalna metody __init__
   ak2='2 atrybut klasy'

obj=klasa()
print(obj.ak1)    #atrybut klasy
print(obj.ak2)    #atrybut klasy

print(obj.ao1)    #atrybut obiektu
obj.ao2='2 atrybut obiektu'    #nowy atrybut obiektu stworzony w locie
print(obj.ao2)
obj.ak1='3 atrybut obiektu'    #nowy atrybut obiektu który 'przesłania' atrybut klasy
print(obj.ak1)              #teraz to atrybut obiektu

print(klasa.ak1)            #oryginalny atrybut klasy pozostaje bez zmian
```

```
#output:
1 atrybut klasy
2 atrybut klasy
1 atrybut obiektu
2 atrybut obiektu
3 atrybut obiektu
1 atrybut klasy
```

Klasy - zasięg zmiennych

Atrybuty prywatne i publiczne:

- atrybuty są domyślnie publiczne, chyba że nazwa poprzedzona „__” (podwójne podkreślenie), wtedy zasięg tylko w obrębie klasy/obiektu.

Przykład 10:

```
class klasa:
   atr1 = 'atrybut publiczny klasy'        #atrybut publiczny klasy
   __atr2 = 'atrybut prywatny klasy'       #atrybut „prywatny” klasy
   def __init__ (self):
      self.atr4 = 'atrybut publiczny obiektu'
      self.__atr3 = 'atrybut prywatny obiektu'
   def set_atr (self, zm1,zm2):     #metoda - zmiana prywatnych atrybutów
      self.__atr3 = zm1
      klasa.__atr2 = zm2
   def get_atr (self):              #dostęp do prywatnych atrybutów
      return klasa.__atr2, self.__atr3
#ciąg dalszy na następnym slajdzie…
```

Klasy - zasięg zmiennych

Przykład 10 - kont.:

```
print(klasa.atr1)
#print(klasa.__atr2)  #błąd - dostęp bezpośredni jest zabroniony

obj=klasa()
print(obj.atr1)
#print(obj.__atr2)   #błąd - dostęp bezpośredni jest zabroniony
#print(obj.__atr3)   #błąd - dostęp bezpośredni jest zabroniony

print(obj.get_atr())
obj.set_atr('nowy atrybut obiektu', 'nowy atrybut klasy')
print(obj.get_atr())
```

```
#output:
atrybut publiczny klasy
atrybut publiczny klasy
('atrybut prywatny klasy', 'atrybut prywatny
obiektu')
('nowy atrybut klasy', 'nowy atrybut obiektu')
```

Dziedziczenie klas

Dziedziczenie :

- tworzenie nowych klas na podstawie już istniejących. W Pythonie dziedziczenie realizuje się przez podanie klas bazowych oddzielanych przecinkami w definicji klasy pochodnej.

Przykład 11:

```
class samochod:
    def set_kolor (self, kolor):
        self.__kolor = kolor
    def get_color(self):
        return self.__kolor

class osobowy(samochod):        #dziedziczenie po klasie samochód
    def set_marka (self, marka):
        self.__marka = marka
    def get_marka (self):
        return self.__marka

sam=osobowy()
sam.set_kolor ("niebieski")
sam.set_marka ("Ford")
print ("To jest %s %s." %(sam.get_color(), sam.get_marka()))
```

```
#output:
To jest niebieski Ford.
```

Dziedziczenie klas - funkcja ‘super’

Funkcja ‘super’ :

- jeśli w klasie potomnej implementujemy metodę zdefiniowaną już w klasie bazowej, a chcemy wywołać metodę z klasy bazowej, używamy funkcji ‘super’.

Przykład 12:

```
class A( object ):
    def funkcja ( self ):
        print("Wywolanie  A")

class B(A):
    def funkcja ( self ):
        print("Wywolanie  B")
        super(B, self ).funkcja()    #wywołanie metody z klasy nadrzędnej

kb=B()
kb.funkcja()
```

```
#output:
Wywolanie  B
Wywolanie  A
```

Dziedziczenie klas - funkcja ‘super’

Przykład 13 - przykład z samochodem:

```
class samochod:
    def set_kolor(self, kolor):
        self.__kolor = kolor
    def get_info(self):
        return self.__kolor

class osobowy(samochod):
    def set_marka(self, marka):
        self.__marka = marka
    def get_info(self):
        return super().get_info(), self.__marka    #f. 'super' wywołanie bez parametrów.

sam=osobowy()
sam.set_kolor ("niebieski")
sam.set_marka ("Ford")
print ("To jest %s %s." %(sam.get_info()))
```

```
#output:
To jest niebieski Ford.
```

Klasy - metody specjalne

Metody specjalne:

- są to metody, których nazwy zaczynają się i kończą podwójnym podkreśleniem: ‘**__nazwa__’,**
- metody o takich nazwach są wywoływane automatyczne w pewnych sytuacjach,
- można je implementować we własnych klasach (niektóre są już zdefiniowane w nadrzędnej klasie 'object', ale można je redefiniować),
- typowy przykład metod specjalnych to metody do przeciąża operatorów, np.: __add__, __mul__,
- inne przykłady metod specjalnych: __str__, __repr__, __init__, __call__.

Klasy - metody specjalne

Przykład 14:

```
class A:
    def __init__ (self, zmienna):    #metoda uruchamiana przy tworzeniu obiektów klasy A
        self.zmienna = zmienna
    def __add__ (self, other):       #metoda uruchamiana przy dodawania obiektów klasy A
        return A(self.zmienna + other.zmienna)
    def __str__ (self):              #metoda uruchamiana przy  'drukowaniu' obiektów klasy A
        return str(self.zmienna)

a = A(5)
b = A(8)

print(a+b)
```

```
#output:
13
```

Klasy - metody specjalne

Przykład 15:

```
class samochod:
    def __init__(self, kolor):
        self.__kolor = kolor
    def set_kolor(self, kolor):
        self.__kolor = kolor
    def get_info(self):
        return self.__kolor

class osobowy(samochod):
    __marka='Ford'
    def __init__(self, kolor, model):
        self.__model = model
        super().__init__(kolor)
    def set_marka(self, model):
        self.__model = model
    def get_info(self):
        return (super().get_info(),
        self.__marka, self.__model)
```

...

```
mustang1=osobowy('niebieski','Mustang')
mustang2=osobowy('czerwony','Mustang')
fiesta1=osobowy('czarny','Fiesta')

print ("To jest %s %s %s." %(mustang1.get_info()))
print ("To jest %s %s %s." %(mustang2.get_info()))
print ("To jest %s %s %s." %(fiesta1.get_info()))
```

```
#output:
To jest niebieski Ford Mustang.
To jest czerwony Ford Mustang.
To jest czarny Ford Fiesta.
```

Zadanie 1

Zaimplementuj klasę ”AddMul”. Klasa przechowuje liczbę i ma mieć możliwość dodawania i mnożenia z wykorzystaniem odpowiednich operatorów.

Po przekazaniu obiektu do funkcji „print” liczba reprezentowana przez dany obiekt powinna zostać wydrukowana.

Przykład (wywołanie):

```
a=AddMul(5)
b=AddMul(5)
c=a*b
print(c, type(c))
print(a+b, type(a+b))
```

Przykład (wyjście):

```
25 <class '__main__.AddMul'>
10 <class '__main__.AddMul'>
```

Zadanie 2

Zaimplementuj cztery klasy A, B, C, i D.

W każdej z tych klas zdefiniuj metodę info(), która drukuje adekwatny napis "Klasa A", "Klasa B"...

Dodatkowo klasa D dziedziczy po klasach A, B i C. Wykorzystaj funkcję super, do tego aby osiągnąć następujący efekt:

Przykład (wywołanie):

```
D_obj=D()
D_obj.info()
```

Przykład (wyjście):

```
Klasa D
Klasa A
Klasa B
Klasa C
```

Zadanie 3

Zaimplementuj klasę ”LiczbaZespolona”. Klasa ma mieć możliwość dodawania, odejmowania, mnożenia oraz dzielenia liczb zespolonych z wykorzystaniem standardowych operatorów.

Dodatkowo powinna posiadać funkcję ”modul” obliczającą moduł liczby zespolonej oraz możliwość porównywania przy pomocy operatora '='. Po przekazaniu obiektu do funkcji „print” liczba zespolona reprezentowana przez dany obiekt powinna zostać wyświetlona na ekranie (nie korzystamy z żadnych dodatkowych/gotowych modułów).

cd Zadania 3

Przykład (wywołanie):

```
za=LZ(1,1)
zb=LZ(2,2)

print("za=%s" %za)
print("zb=%s" %zb)
print("%s+%s=%s" %(za,zb,(za+zb)))
print("%s-%s=%s" %(za,zb,(za-zb)))
print("%s*%s=%s" %(za,zb,(za*zb)))
print("modul(za) = %s" %(za.modul()))
print("modul(zb) = %s" %(zb.modul()))
print("%s/%s=%s" %(za,zb,(za/zb)))
```

Przykład (wyjście):

```
za=(1.00,1.00)
zb=(2.00,2.00)
(1.00,1.00)+(2.00,2.00)=(3.00,3.00)
(1.00,1.00)-(2.00,2.00)=(-1.00,-1.00)
(1.00,1.00)*(2.00,2.00)=(0.00,4.00)
modul(za) = 1.4142135623730951
modul(zb) = 2.8284271247461903
(1.00,1.00)/(2.00,2.00)=(0.50,0.00)
```

Zadanie 4

Zaimplementuj dwie klasy: Punkt2D i rozszerzającą ją klasę Punkt3D. Każda z klas powinna mieć możliwość obliczania odległości między dwoma punktami zrealizowaną w postaci operatora odejmowania (nie korzystamy z żadnych dodatkowych/gotowych modułów).

Zadanie 5

Zamodeluj klasę „samochod” dziedziczącą po trzech różnych podzespołach (klasy bazowe - np. koło, silnik, skrzynia, itp.).

Każda klasa bazowa powinna mieć:

- co najmniej dwa prywatne pola (tzn. atrybuty opisujące dany obiekt) oraz odpowiednie metody do ich modyfikowania,
- zdefiniowaną metodę (np. get_info) zwracającą wszystkie swoje atrybuty w postaci słownika,
- metodę inicjującą wszystkie atrybuty przy tworzeniu instancji.

Klasa potomna (samochod) powinna posiadać:

- także minimum dwa pola prywatne i odpowiednie metody publiczne do ich modyfikacji,
- metodę inicjującą wszystkie atrybuty (także z klas bazowych) przy tworzeniu instancji
- publiczną metodę (get_info), która zwraca zmienną słownikową zawierającą wszystkie dane opisujące cały samochód (także atrybut z klas bazowych).

cd Zadania 5

Przykład (wywołanie):

```
print('\nSam 1:\n')
sam1=samochod("Fiat","sedan")
print(sam1)

print('\nSam 2:\n')
sam2=samochod(
marka_samochodu='Ford',
typ_nadwozia='sedan',
pojemnosc_silnika=1600,
rodzaj_paliwa="benzyna",
rozmiar_kola=17,
rodzaj_felgi='alu',
typ_skrzyni="manual",
biegi_skrzyni=6
)
print(sam2)
```

Przykład (wyjscie):

```
Sam 1 :

marka samochodu: Fiat
typ nadwozia: sedan
pojemnosc silnika: ----
rodzaj paliwa: ----
rozmiar_kola: ----
rodzaj felgi: ----
typ skrzyni: ----
biegi skrzyni: ----

Sam 2:

marka samochodu: Ford
typ nadwozia: sedan
pojemnosc silnika: 1600
rodzaj paliwa: benzyna
rozmiar_kola: 17
rodzaj felgi: alu
typ skrzyni: manual
biegi skrzyni: 6
```