Wyklad12

January 10, 2020

1 Dziedziczenie

[1]:
```
class Parent:
    def __init__(self, a):
        self.a = a

    def metoda1(self):
        print(self.a*2)

    def metoda2(self):
        print(self.a+ ' !!! ' )

class Child(Parent):
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def metoda1(self):
        print(self.a*7)

    def metoda3(self):
        print(self.a + self.b)
```

[2]:
```
p = Parent('Hi')
c = Child('Hi', 'Bye')
```

[3]:
```
type(p)
```

[3]:
```
__main__.Parent
```

[4]:
```
type(c)
```

[4]:
```
__main__.Child
```

[5]:
```
print('Parent-metoda 1: ',end='')
p.metoda1()
print('Parent-metoda 2: ',end='')
```

```
p.metoda2()
print()
print('Child-metoda 1: ',end='')
c.metoda1()
print('Child-metoda 2: ',end='')
c.metoda2()
print('Child-metoda 3: ',end='')
c.metoda3()
```

```
Parent-metoda 1: HiHi
Parent-metoda 2: Hi !!!

Child-metoda 1: HiHiHiHiHiHiHi
Child-metoda 2: Hi !!!
Child-metoda 3: HiBye
```

Nadpisywanie funkcji

[6]:
```
class A:
    def f(self):
        return self.g()

    def g(self):
        return 'A'

class B(A):
    def g(self):
        return 'B'

a = A()
b = B()
```

[7]:
```
print(a.g(), a.f())
```

```
A A
```

[8]:
```
print(b.g(), b.f())
```

```
B B
```

[ ]:
```
Klasa Child może zawierać wiecej atrybutow z klasy Parent
```

[9]:
```
class Parent:
    def __init__(self, a):
        self.a = a

    def print_var(self):
        print("The value of this class's variables are:")
        print(self.a)

class Child(Parent):
    def __init__(self, a, b):
        Parent.__init__(self, a)
        self.b = b

    def print_var(self):
        Parent.print_var(self)   #metoda z klasy Parent
        print(self.b)
```

[10]:
```
p = Parent('hi')
c = Child('Hi', 'Bye')
```

[11]:
```
p.print_var()
```

```
The value of this class's variables are:
hi
```

[12]:
```
c.print_var()
```

```
The value of this class's variables are:
Hi
Bye
```

[ ]:
```
Przyklad zastosowania
```

[13]:
```
class Student:

    last_index = 1234  # atrybut klasy

    def __init__(self, name):
        Student.last_index +=1 # aktualizuj numer indeksu

        self.name = name
       self.index = Student.last_index # przypisz studentowi imię i numer indeksu

    def __str__(self):  # Student [imię] (nr indeksu)
        return "Student {} (nr indeksu {})".format(self.name, self.index)
```

[14]:
```
student = Student("Anna Nowak")
print(student)
```

```
Student Anna Nowak (nr indeksu 1235)
```

[16]:
```
class StudentISSP(Student):  # też Student
```

```
    def __init__(self, name, przedmioty):
        Student.__init__(self,name)  #super().__init__(name)
        self.przedmioty = przedmioty

    def __str__(self):
        return super().__str__() + \
               ": " + ", ".join(self.przedmioty)
```

[17]:
```
studentISSP = StudentISSP("Jan Kowalski", ["Python", "Algebra"])
print(studentISSP)
```

```
Student Jan Kowalski (nr indeksu 1236): Python, Algebra
```

[18]:
```
student2 = Student("Tomasz Bąk")
print(student2)
```

```
Student Tomasz Bąk (nr indeksu 1237)
```

Dziedziczenie wielokrotne

[19]:
```
class Samochod:
    def run(self):
        print("Jadę...")

class Lodz:
    def run(self):
        print("Płynę...")

class Amfibia1(Samochod, Lodz): pass
class Amfibia2(Lodz, Samochod): pass
class Amfibia3(Samochod, Lodz):
    def run(self):
        print("Jadę i płynę...")
```

[20]:
```
amifibia1 = Amfibia1()
amifibia1.run()
```

```
Jadę…
```

[21]:
```
amifibia2 = Amfibia2()
amifibia2.run()
```

```
Płynę…
```

[22]:
```
amifibia3 = Amfibia3()
amifibia3.run()
```

```
Jadę i płynę…
```

2 Kopiowanie obiektów

[23]:
```
class Point:
    def __init__(self, x, y):
        self.x=x
        self.y=y

class Rectangle:
    def __init__(self, a, b, x0, y0):
        self.width=a
        self.hight=b
        self.corner = Point(x0,y0)
```

[24]:
```
box=Rectangle(100,200,0,0)
```

[25]:
```
print(box.width,box.hight)
```

```
100 200
```

[26]:
```
print(box.corner.x,box.corner.y)
```

```
0 0
```

[27]:
```
import copy
```

[28]:
```
box2=copy.copy(box)
```

[29]:
```
print(box2.width,box2.hight,box2.corner.x,box2.corner.y)
```

```
100 200 0 0
```

[30]:
```
box2 is box
```

[30]:
```
False
```

[31]:
```
box2.corner is box.corner
```

[31]:
```
True
```

[32]:
```
box3=copy.deepcopy(box)
```

[33]:
```
box3.corner is box.corner
```

[33]:
```
False
```

3 Iteratory

[34]:
```
for i in [1, 2, 3]:
    print(i)
```

```
1
2
3
```

[35]:
```
x=iter([1, 2, 3])
x
```

[35]:
```
<list_iterator at 0x7f4d90a48bd0>
```

[36]:
```
next(x)
```

[36]:
```
1
```

[37]:
```
next(x)
```

[37]:
```
2
```

[38]:
```
next(x)
```

[38]:
```
3
```

[38]:
```
next(x)
```

```
       ␣
↪---------------------------------------------------------------------------

     StopIteration                             Traceback (most recent call last)

     <ipython-input-38-92de4e9f6b1e> in <module>
----> 1 next(x)


     StopIteration:
```

[39]:
```
class yrange:
    def __init__(self, n):
        self.start = 0
        self.stop = n

    def __iter__(self):
        return self

    def __next__(self):
        if self.start < self.stop:
            i = self.start
            self.start += 1
            return i
        else:
            raise StopIteration()
```

[40]:
```
y=yrange(3)
```

[41]:
```
print(next(y),next(y),next(y))
```

```
0 1 2
```

[42]:
```
next(y)
```

```
       ␣
↪---------------------------------------------------------------------------

        StopIteration                             Traceback (most recent call last)

        <ipython-input-42-81b9d2f0f16a> in <module>
    ----> 1 next(y)


        <ipython-input-39-19c2b2240afd> in __next__(self)
         13             return i
         14         else:
    ---> 15             raise StopIteration()


        StopIteration:
```

[43]:
```
y=yrange(3)
list(y)
```

[43]:
```
[0, 1, 2]
```

[44]:
```
list(y)
```

[44]:
```
[]
```

[45]:
```
class yrange_new:
    def __init__(self, n):
        self.n = n

    def __iter__(self):
        return yrange(self.n)

class yrange:
    def __init__(self, n):
        self.start = 0
        self.stop = n

    def __iter__(self):
        return self

    def __next__(self):
        if self.start < self.stop:
            i = self.start
            self.start += 1
            return i
        else:
            raise StopIteration()
```

[46]:
```
y_new=yrange_new(4)
list(y_new)
```

[46]:
```
[0, 1, 2, 3]
```

[47]:
```
list(y_new)
```

[47]:
```
[0, 1, 2, 3]
```

Po co **iter**?

[48]:
```
class yrange2:
    def __init__(self, n):
        #self.start = 0
        self.stop = n

    def __iter__(self):
        self.start = 0
        return self

    def __next__(self):
        if self.start < self.stop:
            i = self.start
            self.start += 1
            return i
        else:
            raise StopIteration()
```

[49]:
```
y2=yrange2(3)
```

[50]:
```
next(y2)
```

```
      ␣
↪---------------------------------------------------------------------------

       AttributeError                            Traceback (most recent call last)

       <ipython-input-50-3bc212bc78c2> in <module>
   ----> 1 next(y2)


       <ipython-input-48-a196b7f44f45> in __next__(self)
         9
        10     def __next__(self):
   ---> 11         if self.start < self.stop:
        12             i = self.start
        13             self.start += 1


       AttributeError: 'yrange2' object has no attribute 'start'
```

[51]:
```
y2_new=iter(y2)
print(next(y2_new),next(y2_new),next(y2_new))
```

```
0 1 2
```

[52]:
```
next(y2_new)
```

```
      ␣
↪---------------------------------------------------------------------------

       StopIteration                             Traceback (most recent call last)

       <ipython-input-52-5acb01e121b0> in <module>
   ----> 1 next(y2_new)


       <ipython-input-48-a196b7f44f45> in __next__(self)
```

```
     14             return i
     15         else:
---> 16             raise StopIteration()


    StopIteration:
```

```
[53]: next(iter(y2))
```

```
[53]: 0
```

4 Generatory

```
[54]: def zrange(n):
          i = 0
          while i < n:
              yield i
              i += 1
```

```
[55]: z=zrange(3)
```

```
[56]: z
```

```
[56]: <generator object zrange at 0x7f4d9018e7d0>
```

```
[57]: print(next(z),next(z),next(z))
```

```
0 1 2
```

```
[58]: def foo():
          print("begin")
          for i in range(3):
              print("before yield", i)
              yield i
              print("after yield", i)
          print("end")
```

```
[59]: f=foo()
```

```
[60]: next(f)
```

```
begin
before yield 0
```

```
[60]: 0
```

```
[61]: next(f)
```

```
after yield 0
before yield 1
```

```
[61]: 1
```

```
[62]: next(f)
```

```
after yield 1
before yield 2
```

```
[62]: 2
```

```
[63]: next(f)
```

```
after yield 2
end


      ␣
↪---------------------------------------------------------------------------

       StopIteration                             Traceback (most recent call last)

       <ipython-input-63-aff1dd02a623> in <module>
    ----> 1 next(f)


       StopIteration:
```

Zastosowanie

```
[64]: def geometryczny(a1, q, n):# ciąg geometryczny

          ciag = [a1]

          for _ in range(n-1):          # n-1 bo pierwszy już jest
              ciag.append(ciag[-1]*q)   # następny = poprzedni * iloraz

          return ciag
```

```
[65]: ciag = geometryczny(1, 3, 10)

      print(ciag)
```

```
[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683]
```

[66]:
```
def gen_geometryczny(a1, q, n):

    for _ in range(n):
        yield a1 # zwróć obecną wartość a1
        a1 *= q  # i czekaj na kolejną iterację
```

[67]:
```
ciag2 = gen_geometryczny(1, 3, 10)
```

[68]:
```
print(next(ciag2),next(ciag2),next(ciag2),next(ciag2),next(ciag2))
```

```
1 3 9 27 81
```

inny przykład

[69]:
```
class geoCiag:
    """Ciąg geometryczny"""

    def __init__(self, a1, q, n=1):
        """Inicjuje ciąg
        a1 -- pierwszy wyraz ciągu
        q -- iloraz
        n -- początkowa liczba wyrazów
        """

        self.__a1 = a1
        self.__q = q
        self.__wyrazy = [a1]

        if n > 1:
            self.generate(n - 1)

    def generate(self, n):
        """Generuje kolejne wyrazy ciągu"""

        for _ in range(n):
            self.__wyrazy.append(self.__wyrazy[-1] * self.__q)


    def __str__(self):
        s = "Ciąg arytmetyczny ({a1}, {q}):".format(a1=self.__a1, q=self.__q)

        for wyraz in self:  # skąd wie, jak po sobie iterować?
            s += " " + str(wyraz)
        return s

    def __iter__(self):  # iterator ciągu
        """Umożliwia iterację po ciągu"""
        for a in self.__wyrazy:
            yield a

    def __len__(self):  # wywoływana przez len()
        """Zwraca ilość wyrazów ciągu"""

        return len(self.__wyrazy)
```

[70]:
```
ciag_klasa = geoCiag(1, 3, 5)

print(ciag_klasa)
```

```
Ciąg arytmetyczny (1, 3): 1 3 9 27 81
```

[71]:
```
len(ciag_klasa)
```

[71]:
```
5
```

[72]:
```
ciag_klasa.generate(3)

print(ciag_klasa)
```

```
Ciąg arytmetyczny (1, 3): 1 3 9 27 81 243 729 2187
```

[73]:
```
len(ciag_klasa)
```

[73]:
```
8
```

[74]:
```
sum(ciag_klasa) #mozemy tak, bo mamy __iter__
```

[74]:
```
3280
```

[ ]:
```

```