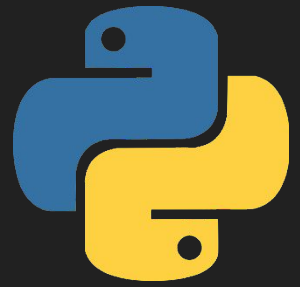


Section X: Объектно-ориентированное программирование (ООП)

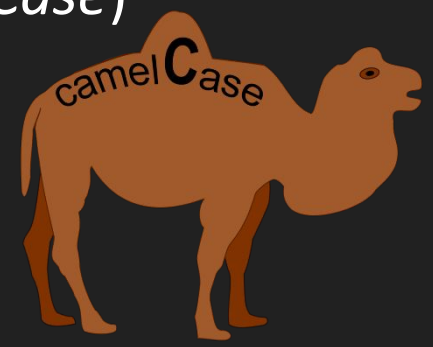
Section X: ООП. Введение

Предыстория, кастомные типы + самый простой класс в мире

+

DRY – Don’t Repeat Yourself

Header (в стиле *CamelCase*)




```
1   class Point:
2       """Represents a point in 2-D space."""
3       pass
```

Пишем докстринги, бережем себя!

```
5   print(Point)
```

```
<class '__main__.Point'>
```

Section X: ООП. Введение

Экземпляры класса. Атрибуты

```
5    blank = Point()
6    print(blank)
```

```
<__main__.Point object at 0x0000016DBBFB93C8>
```

```
6    print(type(blank))
```

```
<class '__main__.Point'>
```

```
7    blank.x = 0
8    blank.y = 1
9    print(blank.x, blank.y)
```

```
0 1
```

Section X: ООП. Введение

Атрибуты и методы

```
1   class Point:
2       """Represents a point in 2-D space."""
3       desc = "Point in 2-D space"
4
5   blank = Point()
6
7   print(blank.desc)
```

```
Point in 2-D space
```

Section X: ООП. Введение

Атрибуты и методы

```
1   class Point:
2       """Represents a point in 2-D space."""
3       def print_coordinates(self):
4           print('x = {}, y = {}'.format(self.x, self.y))

6   blank = Point()
7
8   blank.x = 0
9   blank.y = 1
10  blank.print_coordinates()
```

```
x = 0, y = 1
```

`self` – доступ к атрибутам и методам класса

Section X: ООП. Введение

Инициализация. Конструктор класса – магический метод

```
1   class Point:
2       """Represents a point in 2-D space."""
3       def __init__(self, x, y):
4           self.x = x
5           self.y = y

7   point = Point(x=1, y=0)
8
9   print(point.x, point.y)
```

```
1 0
```

Кто звал `__init__`??

И где вообще значение аргумента `self`??

Section X: ООП. Введение

Атрибуты и методы

```python
class Email:
    def __init__(self, text):
        self.text = text
        self.issent = False

    def send(self):
        print(self.text)
        self.issent = True

letter = Email('Hello there!')
print(letter.issent)
letter.send()
print(letter.issent)
```

```
False
Hello there!
True
```

Section X: ООП. Введение

Экземпляры (instances) класса

```
1   class Point:
2       """Represents a point in 2-D space."""
3       def __init__(self, x, y):
4           self.x = x
5           self.y = y
6
7   class Rectangle:
8       """Represents a rectangle in 2-D space."""
9       def __init__(self, x, y, width, height):
10          self.corner = Point(x, y)
11          self.width = width
12          self.height = height
13
14  rect = Rectangle(x=0, y=0, width=10, height=5)
15  print(rect.corner)
```

```
<__main__.Point object at 0x0000029AF7E79470>
```

```
15  print(rect.corner.x, rect.corner.y)
```

```
0 0
```

Section X: ООП. Введение

Еще немного магии…

```
1   class Point:
2       """Represents a point in 2-D space."""
3       def __init__(self, x, y):
4           self.x = x
5           self.y = y
6
7       def __str__(self):
8           return 'x = {}, y = {}'.format(self.x, self.y)
9
10  class Rectangle:
11      """Represents a rectangle in 2-D space."""
12      def __init__(self, x, y, width, height):
13          self.corner = Point(x, y)
14          self.width = width
15          self.height = height
16
17  rect = Rectangle(x=0, y=0, width=10, height=5)
18  print(rect.corner)
```

```
x = 0, y = 0
```

Section X: ООП. Введение

Экземпляры (instances) класса. Модуль `copy`

```
17    print(rect1 == rect2)
18    print(rect1 is rect2)
```

```
14    rect1 = Rectangle(x=0, y=0, width=10, height=5)
15    rect2 = rect1
```

```
True
True
```

```
14    rect1 = Rectangle(x=0, y=0, width=10, height=5)
15    rect2 = Rectangle(x=0, y=0, width=10, height=5)
```

```
False
False
```

```
14    import copy
15    rect1 = Rectangle(x=0, y=0, width=10, height=5)
16    rect2 = copy.copy(rect1)
```

```
False
False
```

Section X: ООП. Введение

Копирование объектов. Shallow copy

```
14    import copy
15    rect1 = Rectangle(x=0, y=0, width=10, height=5)
16    rect2 = copy.copy(rect1)
```

```
21    rect2.width = 15
22    print(rect1.width)
```

```
10
```

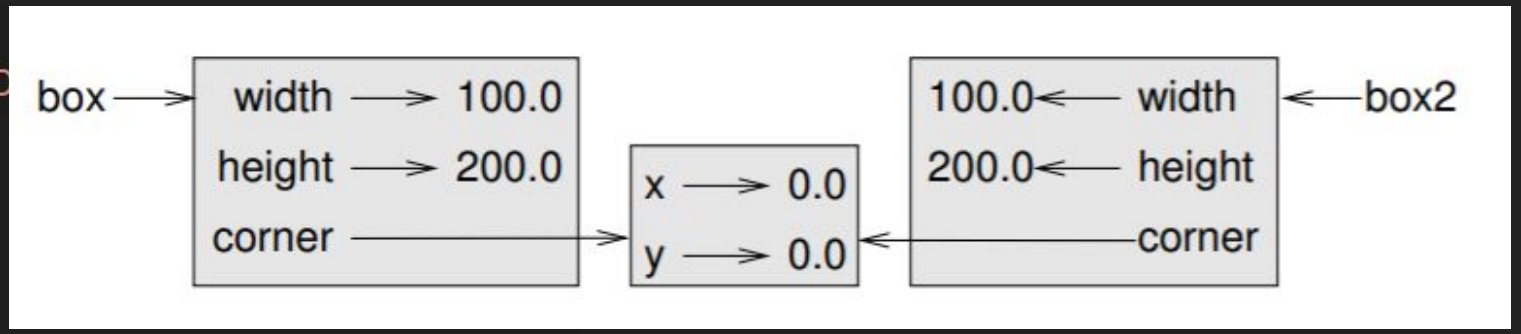
```
24    rect2.corner.x = 3
25    print(rect1.corner)
```

```
x = 3, y = 0
```

Section X: ООП. Введение

Копирование объектов. Shallow copy

```
1   class Point:
2       """Represents a point in 2-D
3       def __init__(self, x, y):
4           self.x = x
5           self.y = y
6
7       def __str__(self):
8           return 'x = {}, y = {}'.format(self.x, self.y)
9
10  class Rectangle:
11      """Represents a rectangle in 2-D space."""
12      def __init__(self, x, y, width, height):
13          self.corner = Point(x, y)
14          self.width = width
15          self.height = height
```

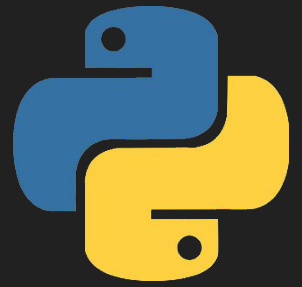


Section X: ООП. Введение

Копирование объектов. Deep copy

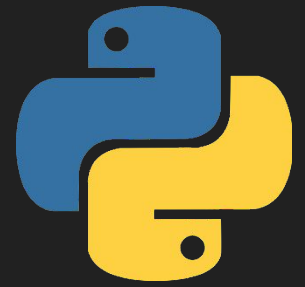
```
17    import copy
18    rect1 = Rectangle(x=0, y=0, width=10, height=5)
19    rect2 = copy.deepcopy(rect1)
20
21    rect2.width = 15
22    print(rect1.width)
23    print(rect2.width)
24
25    rect2.corner.x = 3
26    print(rect1.corner)
27    print(rect2.corner)
```

```
10
15
x = 0, y = 0
x = 3, y = 0
```



Section XI: Принципы ООП

...



Section XI: Принципы ООП: инкапсуляция, наследование, полиморфизм

Section XI: ООП. Принципы

Перегрузка операторов

```
1   class Number:
2       def __init__(self, number):
3           self.number = number
4
5   n1 = Number(1)
6   n2 = Number(1)
7
8   print(n1 == n2)
```

```
False
```

```
10   print(id(n1))
11   print(id(n2))
```

```
1400024831144
1400024831088
```

Section XI: ООП. Принципы

Перегрузка операторов

```python
class Number:
    def __init__(self, number):
        self.number = number

    def __eq__(self, other):
        """Overrides the default implementation"""
        if isinstance(other, Number):
            return self.number == other.number
        return False

n1 = Number(1)
n2 = Number(1)

print(n1 == n2)
```

```
True
```

Section XI: ООП. Принципы

Перегрузка операторов

```
1   class Complex:
2       def __init__(self, a, b):
3           self.a = a
4           self.b = b
5
6        # adding two objects
7       def __add__(self, other):
8           return self.a + other.a, self.b + other.b
9
10      def __str__(self):
11          return self.a, self.b
12
13  Ob1 = Complex(1, 2)
14  Ob2 = Complex(2, 3)
15  Ob3 = Ob1 + Ob2
16  print(Ob3)
```

```
(3, 5)
```

Section XI: ООП. Принципы

Магические методы для перегрузки операторов

OPERATOR	MAGIC METHOD
+	__add__(self, other)
–	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
**	__pow__(self, other)

OPERATOR	MAGIC
<	__lt__(self, other)
>	__gt__(self, other)
<=	__le__(self, other)
>=	__ge__(self, other)
==	__eq__(self, other)
	__ne__(self, other)

OPERATOR	MAGIC METHOD
-=	__isub__(self, other)
+=	__iadd__(self, other)
*=	__imul__(self, other)
/=	__idiv__(self, other)
//=	__ifloordiv__(self, other)
%=	__imod__(self, other)
**=	__ipow__(self, other)

OPERATOR	MAGIC METHOD
–	__neg__(self, other)
+	__pos__(self, other)
~	__invert__(self, other)

https://www.geeksforgeeks.org/operator-overloading-in-python/

Section XI: ООП. Принципы

Типовое распределение (Type-based dispatch)

```python
class Time:
    def __init__(self, hours, minutes, seconds):
        # saves all parameters in the object

    def time_to_int():
        # converts Time object into

    def int_to_time():
        # converts integer into Time
```

```python
    def __add__(self, other):
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)

    def add_time(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)

    def increment(self, seconds):
        seconds += self.time_to_int()
        return int_to_time(seconds)
```

Section XI: ООП. Принципы

Полиморфизм (polymorphism)

```python
class India():
    def capital(self):
        print("New Delhi is the capital of India.")

    def language(self):
        print("Hindi the primary language of India.")

class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
```

```
New Delhi is the capital of India.
Hindi the primary language of India.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
```

Section XI: ООП. Принципы

Наследование (inheritance)

```python
# Base or Super class. Note object in bracket.
class Person(object):
    def __init__(self, name):
        self.name = name

    def getName(self):
        return self.name

    def isEmployee(self):
        return False

# Inherited or Sub class (Note Person in bracket)
class Employee(Person):

    # Here we return true
    def isEmployee(self):
        return True
```

```python
emp = Person("Geek1")  # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

```
Geek1 False
Geek2 True
```

Section XI: ООП. Принципы

Наследование (inheritance)

```python
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

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

```python
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):
        # add properties here
        pass

x = Student("Mike", "Olsen")
x.printname()
```

```
AttributeError: 'Student' object has no attribute 'firstname'
```

Section XI: ООП. Принципы

Наследование (inheritance). Инициализация родителя. Метод

```python
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)

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

```python
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)

x = Student("Mike", "Olsen")
x.printname()
```

Section XI: ООП. Принципы

Наследование (inheritance). Инициализация родителя. Метод

```python
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)

x = Student("Mike", "Olsen", 2020)
x.welcome()
```

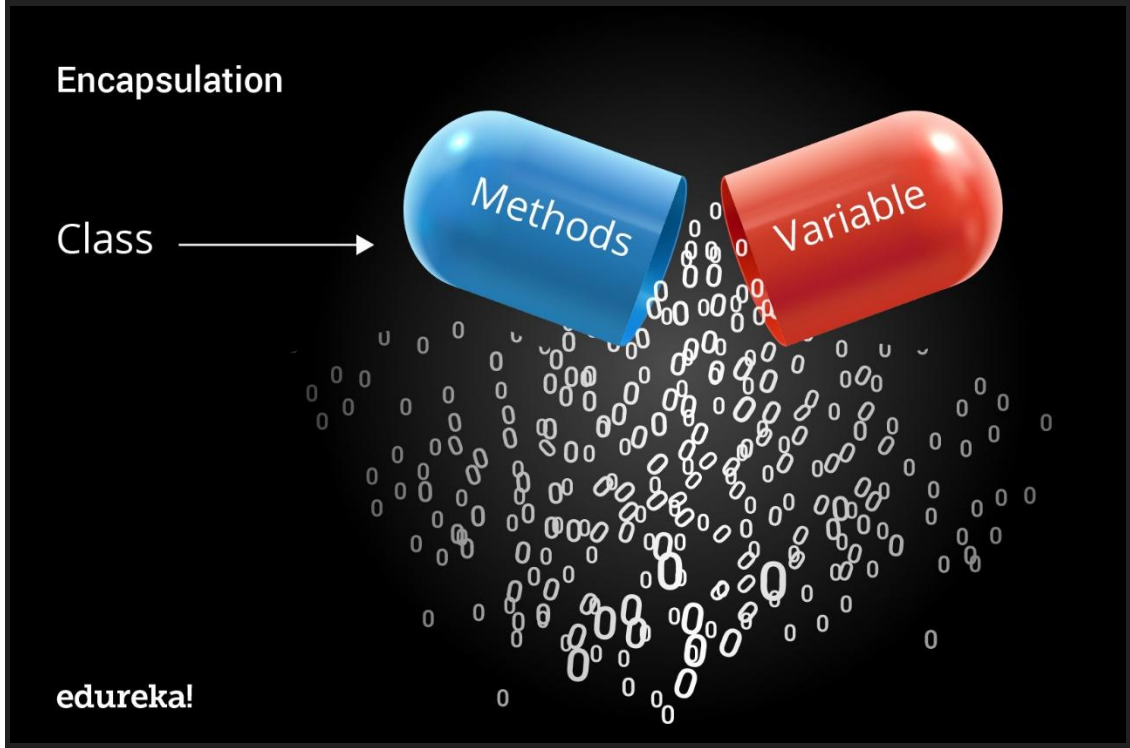
```
Welcome Mike Olsen to the class of 2020
```

Section XI: ООП. Принципы

Инкапсуляция (encapsulation)

```java
// Java program to demonstrate encapsulation
public class Encapsulate
{
    // private variables declared
    // these can only be accessed by
    // public methods of class
    private String geekName;
    private int geekRoll;
    private int geekAge;

    // get method for age to access
    // private variable geekAge
    public int getAge()
    {
      return geekAge;
    }
```



Section XI: ООП. Принципы

Инкапсуляция (encapsulation). Protected members

```
1   # Creating a base class
2   class Base:
3       def __init__(self):
4           # Protected member
5           self._a = 2
6
7   # Creating a derived class
8   class Derived(Base):
9       def __init__(self):
10          # Calling constructor of Base class
11          Base.__init__(self)
12          print("Calling protected member of base class: ")
13          print(self._a)
```

```
15   obj1 = Base()
16   print(obj1.a)
```

```
AttributeError: 'Base' object has no attribute 'a'
```

```
15   obj1 = Base()
16   print(obj1._a)
```

```
2
```

```
18   obj2 = Derived()
```

```
Calling protected member of base class:
2
```

Section XI: ООП. Принципы

Инкапсуляция (encapsulation). Private members

```
1   # Creating a Base class
2   class Base:
3       def __init__(self):
4           self.a = "GeeksforGeeks"
5           self.__c = "GeeksforGeeks"
6
7   # Creating a derived class
8   class Derived(Base):
9       def __init__(self):
10          # Calling constructor of Base class
11          Base.__init__(self)
12          print("Calling private member of base class: ")
13          print(self.__a)
```

```
15  obj1 = Base()
16  print(obj1.a)
```

```
GeeksforGeeks
```

```
18  print(obj1.__c)
```

```
AttributeError: 'Base' object has no attribute '__c'
```

```
15  obj2 = Derived()
```

```
AttributeError: 'Derived' object has no attribute '_Derived__a'
```

Section XI: ООП. Принципы

Разоблачение приватности. Python name mangling

```python
# Creating a Base class
class Base:
    def __init__(self):
        self.a = "GeeksforGeeks"
        self.__c = "GeeksforGeeks"

# Creating a derived class
class Derived(Base):
    def __init__(self):
        # Calling constructor of Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__a)

obj1 = Base()

print(obj1._Base__c)
```

```
GeeksforGeeks
```

https://www.geeksforgeeks.org/private-variables-python/

Section XI: ООП. Принципы

Еще один пример

```python
class Computer:
    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

```
Selling Price: 900
Selling Price: 900
Selling Price: 1000
```

Q&A