Python programming for beginners

Stefan Zhauryd
Instructor

Module 7

Object-Oriented Programming

In this module, you will learn about:

- **Basic concepts of object-oriented programming (OOP)**
- **The differences between the procedural and object approaches (benefits and profits)**
- **Classes, objects, properties, and methods;**
- **Designing reusable classes and creating objects;**
- Inheritance and polymorphism;
- Exceptions as objects.

Methods in detail

```
st = str('fsdfsdf')
st.method()
```

```
class Classy:
    def method(self):
        print("method")


obj = Classy()
obj.method()
```

```
method
>>>
```

```
class Classy:
    def method(self, par):
        print("method:", par)


obj = Classy()
obj.method(1)
obj.method(2)
obj.method(3)
```

```
method: 1
method: 2
method: 3
>>>
```

Methods in detail: continued

```
class Classy:
    varia = 2
    def method(self):
        print(self.varia, self.var)


obj = Classy()
obj.var = 3
obj.method()
```

```
2 3
>>>
```

```
class Classy:
    def other(self):
        print("other")

    def method(self):
        print("method")
        self.other()


obj = Classy()
obj.method()
```

```
method
other
>>>
```

Methods in detail: continued
st = str(1234)

```
class Classy:
    def __init__(self, value):
        self.var = value


obj_1 = Classy("object")

print(obj_1.var)
```

```
object
>>>
```

If you name a method like this: **__init__**, it won't be a regular method - it will be a constructor.

If a class has a constructor, **it is invoked automatically and implicitly when the object of the class is instantiated.**

The constructor:

- **is obliged to have the self parameter** (it's set automatically, as usual);
- may (but doesn't need to) **have more parameters than just self**; if this happens, the way in which the class name is used to create the object must reflect the __init__ definition;
- **can be used to set up the object**, i.e., properly initialize its internal state, create instance variables, instantiate any other objects if their existence is needed, etc.
- **cannot return a value**
- **cannot be invoked** directly either from the object or from inside the class

Methods in detail: continued

__count
_ClassType__cour

```
class Classy:
    def __init__(self, value = None):
        self.var = value


obj_1 = Classy("object")
obj_2 = Classy()

print(obj_1.var)
print(obj_2.var)
```

```
object
None
>>>
```

```
class Classy:
    def visible(self):
        print("visible")

    def __hidden(self):
        print("hidden")


obj = Classy()
obj.visible()

try:
    obj.__hidden()
except:
    print("failed")

obj._Classy__hidden()
```

```
visible
failed
hidden
>>>
```

The inner life of classes and objects __dict__

```
class Classy:
    varia = 1
    def __init__(self):
        self.var = 2
    def method(self):
        pass
    def __hidden(self):
        pass

obj = Classy()

print(obj.__dict__)
print(Classy.__dict__)
```

```
{'var': 2}
{'__module__': '__main__', 'varia': 1, '__init__': <function Classy.__init__ at 0x000002A3DCCE33A0>, 'method': <function Classy.method at 0x000002A3DCCE3430>, '_Classy__hidden': <function Classy.__hidden at 0x000002A3DCCE34C0>, '__dict__': <attribute '__dict__' of 'Classy' objects>, '__weakref__': <attribute '__weakref__' of 'Classy' objects>, '__doc__': None}
>>>
```

The inner life of classes and objects: continued __**name**__

built-in property worth mentioning is __**name**__, which is a string.

The property contains the name of the class. It's nothing exciting, just a string.

```
class Classy:
    pass


print(Classy.__name__)
obj = Classy()
print(type(obj).__name__)

#Error AttributeError: 'Classy' object has no attribute '__name__'
#print(obj.__name__)
```

```
Classy
Classy
>>>
```

The inner life of classes and objects: continued __**module**__

__**module**__ is a string, too - it stores the name of the module which contains the definition of the class.

```
class Classy:
    pass


print(Classy.__module__)
obj = Classy()
print(obj.__module__)
```

```
__main__
__main__
>>>
```

The inner life of classes and objects: continued __bases__

__**bases**__ is a tuple. The tuple contains classes (not class names) which are direct superclasses for the class.

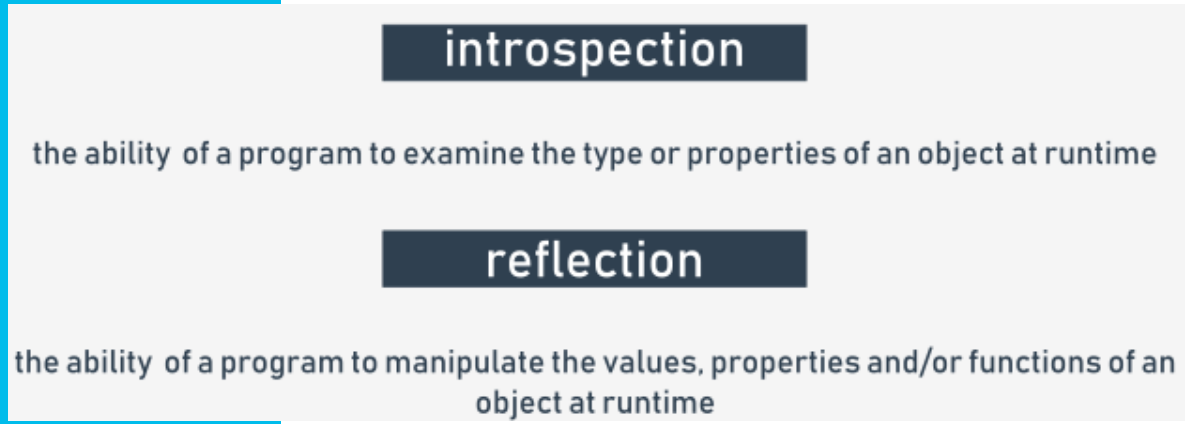
```
class SuperOne:
    pass

class SuperTwo:
    pass

class Sub(SuperOne, SuperTwo):
    pass

def printBases(cls):
    print('( ', end='')

    for x in cls.__bases__:
        print(x.__name__, end=' ')
    print(')')

printBases(SuperOne)
printBases(SuperTwo)
printBases(Sub)
```

```
( object )
( object )
( SuperOne SuperTwo )
>>>
```

Reflection and introspection

introspection, which is the ability of a program to examine the type or properties of an object at runtime;

reflection, which goes a step further, and is the ability of a program to manipulate the values, properties and/or functions of an object at runtime.



Investigating classes

~~type(num) == str~~

isinstance(num, str)

~~res = obj.a~~

res = getattr(obj, 'a')

```
{'a': 1, 'b': 2, 'i': 3, 'ireal': 3.5, 'integer': 4, 'z': 5}
{'a': 1, 'b': 2, 'i': 4, 'ireal': 3.5, 'integer': 5, 'z': 5}
>>>
```

the **getattr()** function takes two arguments: an object, and its property name **(as a string)**, and returns the current attribute's value;

the **setattr()** function takes three arguments: an object, the property name (as a string), and the property's new value.

```
class MyClass:
    pass

obj = MyClass()
obj.a = 1
obj.b = 2
obj.i = 3
obj.ireal = 3.5
obj.integer = 4
obj.z = 5

def incIntsI(obj):
    for name in obj.__dict__.keys():
        if name.startswith('i'):
            val = getattr(obj, name)
            if isinstance(val, int):
                setattr(obj, name, val + 1)

print(obj.__dict__)
incIntsI(obj)
print(obj.__dict__)
```

Key takeaways

```
class Sample:
    def __init__(self):
        self.name = Sample.__name__
    def myself(self):
        print("My name is " + self.name +\
              " living in a " + Sample.__module__)


obj = Sample()
obj.myself()
```

```
My name is Sample living in a __main__
>>>
```

1. A **method** is a function embedded inside a class. The first (or only) parameter of each method is usually named self, which is designed to identify the object for which the method is invoked in order to access the object's properties or invoke its methods.

2. If a class contains a **constructor** (a method named __init__) it cannot return any value and cannot be invoked directly.

3. All classes (but not objects) contain a property named **__name__,** which stores the name of the class. Additionally, a property named **__module__** stores the name of the module in which the class has been declared, while the property named **__bases__** is a tuple containing a class's superclasses.

# Inheritance - why and how?
st='fsdf'
print(st)

```
class Star:
    def __init__(self, name, galaxy):
        self.name = name
        self.galaxy = galaxy


sun = Star("Sun", "Milky Way")
print(sun)
```

```
<__main__.Star object at 0x000001888001E970>
>>>
```

Inheritance - why and how?
__str__

The default __str__() method returns the previous string - ugly and not very informative. You can change it just by defining your own method of the name.

```
class Star:
    def __init__(self, name, galaxy):
        self.name = name
        self.galaxy = galaxy

    def __str__(self):
        return self.name + ' in ' + self.galaxy


sun = Star("Sun", "Milky Way")
print(sun)
```

```
Sun in Milky Way
>>>
```

Home work 10_1 Timer

We need a class able to count seconds. Easy? Not as much as you may think as we're going to have some specific expectations.

Read them carefully as the class you're about write will be used to launch rockets carrying international missions to Mars. It's a great responsibility. We're counting on you!

Your class will be called **Timer.** Its c**onstructor accepts three arguments representing hours** (a value from range [0..23] - we will be using the military time), **minutes** (from range [0..59]) and **seconds** (from range [0..59]).

Zero is the default value for all of the above parameters. **There is no need to perform any validation checks.**

Home work 10_1 Timer

The class itself should provide the following facilities:

- objects of the class should be "printable", i.e. they should be able to implicitly convert themselves into strings of the following form: "hh:mm:ss", with leading zeros added when any of the values is less than 10;
- the class should be equipped with parameterless methods called next_second() and previous_second(), incrementing the time stored inside objects by +1/-1 second respectively.

Use the following hints:

- all object's properties should be private;
- consider writing a separate function (not method!) to format the time string.

Complete the template I've provided. Run your code and check whether the output looks the same as ours.

```
def two_digits(val):
    s = str(val)
    if len(s) == 1:
        s = '0' + s
    return s
```

Home work 10_1 Timer

def __str__(self):
    return t_dig(hh)+':'+t_dig(mm)+':'+t_dig(ss)

```
23:59:59
00:00:00
23:59:59
```



```
1  class Timer:
2      def __init__( ??? ):
3          #
4          # Write code here
5          #
6
7      def __str__(self):
8          #
9          # Write code here
10         #
11
12     def next_second(self):
13         #
14         # Write code here
15         #
16
17     def prev_second(self):
18         #
19         # Write code here
20         #
21
22
23 timer = Timer(23, 59, 59)
24 print(timer)
25 timer.next_second()
26 print(timer)
27 timer.prev_second()
28 print(timer)
29
```

ЗАДАНИЯ

1) Прорешать всю классную работу
2) Выполнить все домашние задания

Почитать:

1) Byte of Python - стр.108-120

Крайний срок сдачи 17/10 в 21:00 (можно раньше, но не позже)

https://docs.python-guide.org/writing/structure/

ЗАДАНИЯ

Название файлов, которые вы отправляете мне в telegram:

Vasia_Pupkin_class_work_L10_P1.py

Без классной работы домашнее не принимается на проверку.

Vasia_Pupkin_L10_P1.py

Формат сообщения которое вы присылаете мне

(после полного выполнения домашнего задания, только один раз) в Telegram:

Добрый день/вечер. Я Вася Пупкин, и это мои домашние задания к лекции 10 часть 1 про ООП.

И отправляете файлы

Крайний срок сдачи 17/10 в 21:00 (можно раньше, но не позже)

https://docs.github.com/articles/using-pull-requests

Create your possibilities. Bye bye.