Week 5- Practical work. Creating classes and using them

Lesson Plan

1. Creating classes:

Python was designed as an object-oriented programming language. This means that it is built with the following principles in mind:

– all data in it are represented by objects;

– program can compose how kit interactingobjects that send messages to each other;

– each an object It has own part memory and maybeconsist of other objects;

– every object has a type;

– all objects one type may accept alone and thosesamemessages (and perform the same actions);

– each program object is created on the basis of some class of objects.

Python provides the following ways to work with classes:

– you can define your own classes;

– inherit from built-in and own classes (one or more);

– derivatives classes may redefine any methodsbase classes.

Therefore, building a Python application must start with designing and creating classes. Classes can be located either at the beginning of the program code, or imported from other module files (also at the beginning of the code).

The class statement is used to create classes. This is a compound statement that consists of a header line and a body. The title consists of the keyword class, the name of the class, and possibly titles.

superclasses(the concept of a superclass will be discussed later) in parentheses. There may be no superclasses, in which case the parentheses are not required.

class bodyconsists of a block of various instructions. The body must be indented (as are any nested constructs in Python).

Schematically, the class can be represented as follows:

```
classClassname: variable =
value
…
defmethod name(self, ...):self.variable
= value
…
…
```

This diagram is not complete. For example, superclasses (in parentheses) may be listed after the class name in the header, and methods may be more complex.

2. class attributes

Attributesclass are the names of variables outside the functions and the names of the functions. These attributes are inherited by all objects created from this class. Attributes provide the properties and behavior of an object.

Class attributes are of two types:

– data attributes;

– attribute-methods.

Data attributesusually written at the top. The memory for attributes is allocated the moment they are first assigned, either outside or inside the method.

attribute-methodor simply a method is a function inside a class that performs a specific job, which, most often, involves accessing the attributes of the created object.

MethodsClasses are small programs designed to work with objects. Methods can create new properties (data) of an object, change existing ones, and perform other actions on objects. Methods are defined by the def keyword:

defmethod name(self,[parameter list])

Attributes are accessed according to the scheme objectname.attributename. Example.

```
classSimple: u'Simple
class' var =87
deff(self):return"hello world"
```

Here Simple.var and Simple.f are custom attributes.

The first argument to every class method is always the current instance of the class. It is common to call this argument self (the equivalent of the word this in C++). The self argument refers to the instance of the class on which the method is being invoked, establishing a relationship with a particular object.

In the init method (discussed later), self refers to the newly created object. When a method is called, self is not specified, Python adds it automatically.

Objects can have attributes that are created in the method body if the method is called on a specific object.

An example of creating a class:

```
#!/usr/bin/env python2.7 #
-*- coding: utf-8 -*-
classFirst:                    #Definition class
color=red                      #Defining an attribute and setting its value

defout(self):                  #Definition class method
print(self.color +"!") #Method body
```

3. Creating objects:

Objects are created like this:

objectname = classname()

Parentheses are required here! After such an instruction, an object appears in the program, which can be accessed by the name of the variable associated with it. When an object is created, it receives the attributes of its class, i.e. objects have the characteristics defined in their classes.

There is no limit to the number of objects that can be created based on a particular class. Objects of the same class have a similar set of attributes, but the attribute values can be different. In other words, objects of the same class are similar but individually distinguishable.

The method call for a specific object in the main block of the program looks like this:

objectname.methodname(...)

Below is an example of creating classes, class methods, objects, class method calls:

```
#!/usr/bin/env python2.7 #
-*- coding: utf-8 -*-
classFirst:                 #First class definition
color=red                   #Determining and setting the attribute value

defout(self):               #Defining the out method. The self parameter indicates #if
the method belongs to the #defined class (First)
print(self.color +"!")


classSecond:                #Second class definition
color=red                   #Determining and setting the attribute value
form ="circle"              #Determining and setting the attribute value

# Define the changecolor method. The self parameter indicates # whether
the method belongs to the defined class (Second)
defchangecolor(self,
newcolor):self.color = newcolor
```

```
# Define the changeform method. The self parameter indicates # whether
the method belongs to the defined class (Second)
defchangeform(self,
newform):self.form = newform


obj1 = second()#Creating an object obj1 of class Second ()obj2 =
second()#Creating an obj2 object of class Second
()print(obj1.color, obj1.form)# display "red circle"print(obj2.color,
obj2.form)# display "red circle"

obj1.changecolor("green")# change the color of the first
objectobj2.changecolor(blue)# change the color of the second
objectobj2.changeform("oval")# changing the shape of the second
object

print(obj1.color, obj1.form)# display "green circle"
print(obj2.color, obj2.form)# display "blue oval"
```

class objectand an instance of a class are two different objects. The first is generated at the class declaration stage, the second is generated when the class name is called. There can be one class object - it is used as a definition of objects belonging to it, there can be as many instances of the class as you want - these are entities created on the basis of the class.

Often an object (not to be confused with a class object) and an instance of the class are used as synonyms.

4. Class constructor

The class constructor allows you to set certain parameters of an object when it is created. Thus, it becomes possible to create objects with predefined attributes. A class constructor is a method: init(self)

For example, let's create a class Rectangle (Rectangle) with predefined attributes: color, length and width:

```
classRectangle:
```

```
def_____init___(self,color="green",
width=100,height=100):self.color = color
self.width = widthself.height =
height

defsquare(self):
returnself.width *self.height


rect1 =
Rectangle()print(rect1.
color)print(rect1.squar
e())
rect1 =
Rectangle("yellow",23,34)print(rect1.
color)print(rect1.square())
```

5. Practical task

1. Write the two scripts above. See how they work. In the second program, add one more property and one method that allows you to change it. Create a third object and change all of its properties.

2. Create a Name class that takes a first and last name as arguments when constructing.

The class must support attributes:
first_name returning the name
last_name returning the last name
full_name returning the first and last name
initials, which returns the initials

The class must convert the given first and last names into a form where the first and last names start with an uppercase letter and all other letters are lowercase (because the calling code can pass strings like "JOHN", 'jOHN', 'sMiTh', etc. .)
Call examples:
a1 = Name('john', 'SMITH')
a1.first_name→'John'
a1.last_name→'Smith'
a1.full_name→'John Smith'

3. Create a Calculator class that supports:

addition of two numbers
calculating the difference between two numbers
multiplication of two numbers
dividing one number by another
Examples of calls and returns from functions:
calculator = Calculator()

calculator.add(10, 5)→fifteen

calculator.subtract(10, 5)→5