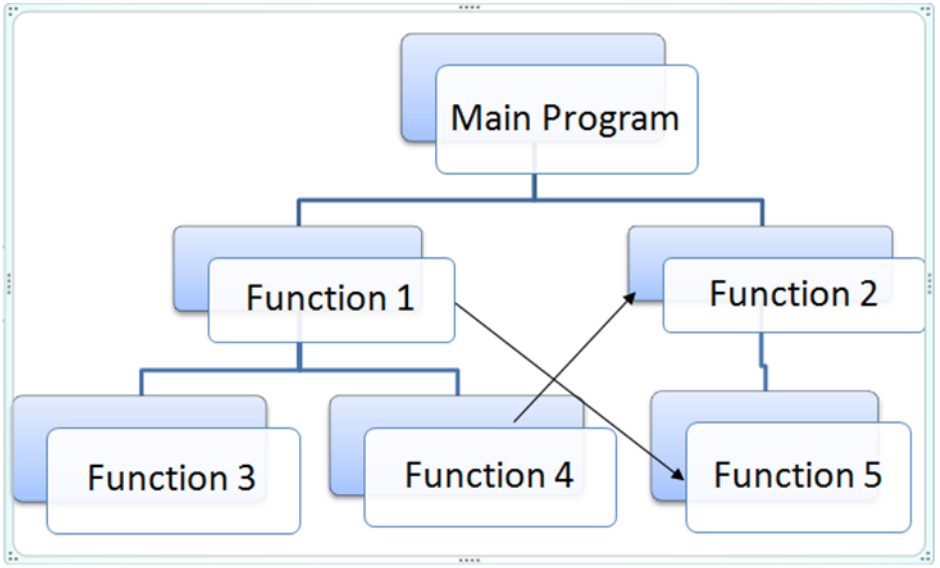
Procedural Oriented Programming
POP
&
Object Oriented Programming
OOP

Procedural Oriented Programming POP



- It means “a set of procedures” which is a “set of subroutines” or a “set of functions “., also known as methods, or functions that contain a series of computational steps to be carried out.

Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating a Function

- In Python a function is defined using the def keyword:

Example

```
def my_function():
        print("Hello from a function")
```

- Function can return data as a result.
- Calling a Function

To call a function, use the function name followed by parenthesis:

```
my_function()
```

Arguments

- Information can be passed into functions as arguments.
- Example

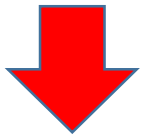
```
def my_function(name):
        print(“Hello “+name)

my_function(“Jack”)
my_function(“Tomas”)
my_function(“bill”)
```

Parameters or Arguments?

- A parameter is the variable listed inside the parentheses in the function definition.

```
def my_function(name):
```

- An argument is the value that is sent to the function when it is called.



```
my_function(“Jack”)
```

Number of Arguments

Example

This function expects 2 parameters, and gets 2 arguments:

```
def my_function(fname, lname):
        print(fname +”  “+ lname)

my_function("Jack","Edison")
```

Note: if you try to call the function with 1 or 3 arguments, you will get an error

Arbitrary Arguments, *args

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

```
def my_function(*kids):
        print("The youngest child is "+ kids[2])


my_function(“Jack"," Tomas "," Bill ")
```

Keyword Arguments

- You can also send arguments with the *key = value* syntax.
- This way the order of the arguments does not matter.

Example

```
def my_function(child3, child2, child1):
        print("The youngest child is "+ child3)

my_function(child1 =“Jack", child2 =" Tomas ", child3 =" Bill")
```

Arbitrary Keyword Arguments, **kwargs

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.
- Example

```
def my_function(**kids):
        print("His last name is "+ kids["lname"])

my_function(fname =“Jack”,lname =“Edison”)
```

Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function without argument, it uses the default value:
- Example

```
def my_function(country =“Iraq"):
        print("I am from "+country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

POP example

```
1  def adding():
2      return x + y,x+1
3
4  def subb():
5      return x-y
6
7  def multi():
8      return x*y
9
10 def divi():
11     return x/y
12
13 x = int(input())
14 y = int(input())
15 result1,x = adding()
16 print("Addition operation = ",result1)
17 result2= subb()
18 print("Subtraction operation = ",result2)
19 result3= multi()
20 print("Multiplication operation = ",result3)
21 result4=divi()
22 print("Division operation = ",result4)
```

Object Oriented Programming OOP

- Class
- Object
- Instance

Create a Class

- To create a class, use the keyword class :

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
        x = 5
```

Create Object

- Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
```

- Accessing Attributes

Object.attribute_name

```
print(p1.x)
```

The __init__() Function

- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:
- Example
- Create a class named Person, use the __init__() function to assign values for name and age:

The __init__() Function

```
class Person:
    def __init__(self, x, y):
        self.name = x
        self.age = y
p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

Object Methods

- Objects can also contain methods. Methods in objects are functions that belong to the object.
- Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def myfunc(self):
        print("Hello my name is " + self.name)
p1 = Person("John", 36)
p1.myfunc()
```

Modify Object Properties

- You can modify properties on objects like this:

Example:

Set the age of p1 to 40:

```
p1.age = 40
```

Type of attribute

1. Public

All members in a Python class are public by default. Any member can be accessed from outside the class environment.

```
class JustCounter:
    secretCount=0


    def count(self):
        self.secretCount+=1
        print(self.secretCount)


counter=JustCounter()
counter.count()
counter.count()
print(counter.secretCount)
```

Type of attribute

2. Protected

Python's convention to make an instance variable protected is to add a prefix _ (single underscore) to it. This effectively prevents it to be accessed, unless it is from within a sub-class.

```
class JustCounter:
    _secretCount=0

    def count(self):
        self._secretCount+=1
        print(self._secretCount)

counter=JustCounter()
counter.count()
counter.count()
print(counter._secretCount)
```

Type of attribute

3. Private

Double underscore __ prefixed to a variable makes it private. It gives a strong suggestion not to touch it from outside the class. Any attempt to do so will result in an AttributeError

```
class JustCounter:
    __secretCount=0

    def count(self):
        self.__secretCount+=1
        print(self.__secretCount)

counter=JustCounter()
counter.count()
counter.count()
print(counter.__secretCount) // Attribute error
```