Basic Programming in Python

5. Session: Object Oriented Programming

Nohayr Muhammad Abdelmoneim

Summer Term 2023

May 22nd, 2023

Overview

- Procedural Programming
- Object Oriented Programming
- How to define a Class
- How to create an Object
- Printing an Object

So far..

- Set of sequential instructions
- Set of variables
- Conditions/loops
- Functions/Methods

Employee example

```
EmployeeName="John"
EmployeeSalary=2000
EmployeeTax=2 #tax deduction will be 20%
EmployeeChildren=2

def calSalary():
    return (1-Employee_tax/10)*Employee_salary+Employee_children*100
```

```
NetSalary=calSalary()
```

```
print(NetSalary)
```

```
1800.0
```

What if there are more employees?

```
EmployeeName1="John"
EmployeeSalary1=2000
EmployeeTax1=2 #tax deduction will be 20%
EmployeeChildren1=2

EmployeeName2="Peter"
EmployeeSalary2=3000
EmployeeTax2=1 #tax deduction will be 10%
EmployeeChildren2=0

EmployeeName3="Sarah"
EmployeeSalary3=4000
EmployeeTax3=3 #tax deduction will be 30%
EmployeeChildren3=1

def calSalary(salary, tax, children):
    return (1-tax/10)*salary+children*100
```

```
NetJohn=calSalary(EmployeeSalary1, EmployeeTax1, EmployeeChildren1)
print("Net salary for John is: ", NetJohn)
NetPeter=calSalary(EmployeeSalary2, EmployeeTax2, EmployeeChildren2)
print("Net salary for Peter is: ", NetPeter)
NetSarah=calSalary(EmployeeSalary3, EmployeeTax3, EmployeeChildren3)
print("Net salary for Sarah is: ", NetSarah)
```

```
Net salary for John is:  1800.0
Net salary for Peter is:  2700.0
Net salary for Sarah is:  2900.0
```

Object-Oriented Programming (OOP)

- Is a programming paradigm that mainly depends on “Objects”
- An object represents a certain thing or aspect.
- An object contains both data (variables) and the methods needed to act on these variables.
- An object is self contained, it should contain all the data and methods needed to represent that object.

Defining a class

- A class in Python is defined by the word class followed by the class name
- A class can be considered as a template or a blueprint that represents a general thing like a car, a student, an employee, etc.
- A class is usually a combination of attributes (variables) and methods (functions)
- A class instance or an object is a variable created with the type of the given class.
- One class can have many instances/objects, each of them can have different values for their attributes (variables).

Defining a class

```
class MyFirstClass:
    s="Hello world of classes!"
```

```
MyObject=MyFirstClass()
print(MyObject.s)
```

```
Hello world of classes!
```

```
AnotherObject=MyFirstClass()
print(AnotherObject.s)
```

```
Hello world of classes!
```

__init__()

- This is a special method called a Dunder method
- This method is executed automatically whenever an instance of a class is defined
- Usually, __init__() should include the code that needs to be executed first thing when an instance/object is created.
- An example would be assigning values to class variables (attributes)
- P.S. A Dunder method is called this way because it begins and ends with double underscore “__”
- A dunder method allows instances to interact with built-in functions/operations in Python.

Example

- Class definition:

```
class Fruits:
    def __init__(self, color, size):
        self.color=color
        self.size=size
```

- Instance creation:

```
melon=Fruits("red","big")
orange=Fruits("orange","medium")
print(melon.color)
```

```
red
```

- melon and orange are objects of the class Fruits

Important notes

- “self” is not a reserved word, it’s just a convention.
- “self” is not passed as a parameter when creating an object of the class.
- Attributes names doesn’t need to match parameters names in __init__() method

```
class Fruits:
    def __init__(myObject, c, s):
        myObject.color=c #color is the class attribute, c is the given value at creation
        myObject.size=s  #size is the class attribute, s is the given value at creation
    def printFruit(Another_name):
        print("The color is ",Another_name.color,"The size is ",Another_name.size)
```

```
melon=Fruits("red","big")
orange=Fruits("orange","medium")
print(melon.color)
melon.printFruit()
orange.printFruit()
```

```
red
The color is  red The size is  big
The color is  orange The size is  medium
```

Employees example revisited

```
class Employee:
    def __init__(self, name, salary, tax, children):
        self.name=name
        self.salary=salary
        self.tax=tax
        self.children=children
    def calSalary(self):
        return (1-self.tax/10)*self.salary+self.children*100
    def printInfo(self):
        print("Employee name: ",self.name,"\n Employee salary:", self.salary \
              ,"\n Employee tax level:", self.tax, "\n Number of children: ", self.children \
             ," \n Employee net salary: ", self.calSalary())
```

```
e1=Employee("John", 2000, 2,2)
e2=Employee("Peter", 3000,1,0)
e3=Employee("Sarah", 4000, 3,1)
e1.printInfo()
e2.printInfo()
e3.printInfo()
```

```
Employee name:  John
 Employee salary: 2000
 Employee tax level: 2
 Number of children:  2
 Employee net salary:  1800.0
Employee name:  Peter
 Employee salary: 3000
 Employee tax level: 1
 Number of children:  0
 Employee net salary:  2700.0
Employee name:  Sarah
 Employee salary: 4000
 Employee tax level: 3
 Number of children:  1
 Employee net salary:  2900.0
```

Printing

- To print the contents of a class we can either define a method for printing (Like in previous example)
- Or we can modify the print function of Python to serve our class.
- This is done using another Dunder method which is __str__

```
e=Employee("John", 2000, 2,2)
print("Using our print method:")
e.printInfo()
print("\n Using Python's print method:")
print(e)
```

```
Using our print method:
Employee name:  John
 Employee salary: 2000
 Employee tax level: 2
 Number of children:  2
 Employee net salary:  1800.0

 Using Python's print method:
<__main__.Employee object at 0x7fde310c3c70>
```

__str__

- __str__ () is a Dunder method that controls how the object is represented when printed. It should return a string.
- You can define your own __str__ method for each class.

```
class Employee:
    def __init__(self, name, salary, tax, children):
        self.name=name
        self.salary=salary
        self.tax=tax
        self.children=children
    def calSalary(self):
        return (1-self.tax/10)*self.salary+self.children*100
    def __str__(self):
        return f"""
               Employee name: {self.name}
               Employee salary: {self.salary}
               Employee tax level: {self.tax}
               Number of children: {self.children}
               Employee net salary:  {self.calSalary()}
               """
```

```
e=Employee("John", 2000, 2,2)
print(e)
```

```
Employee name: John
Employee salary: 2000
Employee tax level: 2
Number of children: 2
Employee net salary:  1800.0
```

Other dunder methods

- You can redefine many Python operations for your own class using dunder methods
- __add__ would redefine the addition property for your class

```
e1=Employee("John", 2000, 2,2)
e2=Employee("Peter", 3000,1,0)
print(e1+e2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[137], line 3
      1 e1=Employee("John", 2000, 2,2)
      2 e2=Employee("Peter", 3000,1,0)
----> 3 print(e1+e2)

TypeError: unsupported operand type(s) for +: 'Employee' and 'Employee'
```

Other dunder methods

```
class Employee:
    def __init__(self, name, salary, tax, children):
        self.name=name
        self.salary=salary
        self.tax=tax
        self.children=children
    def calSalary(self):
        return (1-self.tax/10)*self.salary+self.children*100
    def __str__(self):
        return f"""
               Employee name: {self.name}
               Employee salary: {self.salary}
               Employee tax level: {self.tax}
               Number of children: {self.children}
               Employee net salary:  {self.calSalary()}
               """
    def __add__(ObjectBeforePlus,ObjectAfterPlus):
        return str(ObjectBeforePlus)+str(ObjectAfterPlus)
```

```
e1=Employee("John", 2000, 2,2)
e2=Employee("Peter", 3000,1,0)
print(e1+e2)
```

```
Employee name: John
Employee salary: 2000
Employee tax level: 2
Number of children: 2
Employee net salary:  1800.0

Employee name: Peter
Employee salary: 3000
Employee tax level: 1
Number of children: 0
Employee net salary:  2700.0
```

Famous Errors

- Forgetting to add object reference in methods

```
class MyFirstClass:
    def helloWorld():
        print("Hello world of classes!")
```

```
MyClass=MyFirstClass()
```

```
MyClass.helloWorld()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[79], line 1
----> 1 MyClass.helloWorld()

TypeError: MyFirstClass.helloWorld() takes 0 positional arguments but 1 was given
```

Famous Errors

- Forgetting to add object reference to attributes

```
class Fruits:
    def __init__(self, c, s):
        self.color=c
        self.size=s
    def printFruit(Another_name):
        print("The color is ",color,"The size is ",size)
```

```
melon=Fruits("red","big")
orange=Fruits("orange","medium")
print(melon.color)
melon.printFruit()
orange.printFruit()
```

```
red

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[143], line 4
      2 orange=Fruits("orange","medium")
      3 print(melon.color)
----> 4 melon.printFruit()
      5 orange.printFruit()

Cell In[142], line 6, in Fruits.printFruit(Another_name)
      5 def printFruit(Another_name):
----> 6     print("The color is ",color,"The size is ",size)

NameError: name 'color' is not defined
```

QUESTIONS?