*Ethiopian Institute of architecture, building construction and city development, AAU_ EiABC*

*Programming application short notes: Python Programming Language (part 4)*

Students should revise their study of python part one , part two and part three of the course and proceed to this (python part four) part of the course. Students should also take a look at the appendix section of all the lecture notes.

# Contents

## *List of figures*

## *List of tables*

## *List of boxes*

# Programming application short notes: Python Programming language (part 4)

## Python part 4

### 1. Object oriented programming (OOP) in python

So far, you have covered fundamentals of programming in python (variables, functions and modules). In part one and two of your lecture note you got informed about the fact that python supports: object oriented programming, functional programming and procedure oriented programming.

Extracted from '1.3.2. Discuss about programming styles' of part 1 and 2 of your lecture notes on python. You can also find a more elaborated information of the same topic from part one of your general course on programing and application under '3. Six Different Programing styles'

**(1) Imperative programming**: is just lists of instructions without blocks and loops.
**(2) Structural Programming**: based on imperative programing uses: **if blocks, loops**
**(3) Procedural Programming** has **if blocks, loops, functions**
**(4) Object oriented programming**: uses: **if blocks, loops, functions, objects**
**(5) Declarative programming** (we tell the program what we want instead of how to do a task)
**(6) Functional programming** (nothing happens outside of the scope of the functions used. Since it is declarative, we do not use loops. To replace loops we use recursions.)

*Box 1: Programming styles*

Aside from referring to functions (built in functions) such as integers, floats, strings, etc. as objects now we can discuss about simulating real life objects (we refer to real life phenomenon such as tables, trees, etc. as objects while simulating them in virtual environments, programming) such as: a person, a laptop, a house, etc. every object has (1) Attributes and (2) behavior. Properties (parameters) of an object for example for a table: height, width, time of manufacturing, etc. are attributes of the object. While behaviors (actions) of a person as an object can be speaking, walking, etc. As an object a person knows (attribute) something and based on what the person knows does (action or behavior) something (Navin Reddy). To Store data we define variables (attribute) and to define behaviors we use methods (behavior). Functions in object oriented programming are called methods.

| Properties of an Object | |
|---|---|
| Attributes | Variables |
| Behaviors | Methods (functions) |

*Figure 1: Properties of objects*

From what you have been studying so far the only thing that you will be adding is the fact that you will be working with concepts and that you will be simulating them based on their real world properties as objects. Here concepts you can work with are: Object, class, encapsulation, abstraction and polymorphism. The relationship between class and object is that an object is an instance of class. For example in the class planets you can have instances of mars, earth, etc. objects.

## 1.1. Class and object in python

Without class you cannot work with objects. When we define a function we use the syntax 'def fun_name' with a colon, when we define a class we use the syntax 'class Name:' Class as a key word not capitalized and name of the class capitalized and with a colon at the end. For class we then type the attribute and the behavior of the abject on the next lines of the code.

Recap on data types: run the following codes and see what prints out:

| | |
|---|---|
| `a=10`<br>`b=10.0`<br>`c='all is well'`<br>`print(type(a))`<br>`print(type(b))`<br>`print(type(c))` | `class Tree:`<br>`    def config(self):`<br>`        print("Tree, Root, Stem, Branch, Leaf") # (you can also use the pass statement for this code)`<br><br>`Tree1 = Tree() #object creation`<br>`print(type(Tree1))` |
| `<class 'int'>`<br>`<class 'float'>`<br>`<class 'str'>` | `<class '__main__.Tree'>` |
| The print indicate to what the variables belong to: 'a' belongs to the class integer. | 'Tree1' belongs to the class 'Tree' and the module 'main'. __main__ because the class is constructed here and is not a built in function. |

*Box 2: Recap on data types*

| Compare the following two codes | |
|---|---|
| Uses the module math and log function | Uses the class tree and the config(self) function |
| `import math`<br>`a = 100`<br>`b = 10`<br>`c = math.log(a,b)`<br>`print(c)` | `class Tree:`<br>`    def config(self):`<br>`        print("Tree, Root, Stem, Branch, Leaf")`<br>`tree1 = Tree()`<br>`tree1.config()` |
| `2.0` | `Tree, Root, Stem, Branch, Leaf` |
| Practice your 'control + press' and 'control + space' short keys. Recap also on 'help()' services(utility) of python IDLE and PyCharm. | Replace 'tree1.config()' with 'Tree.config(tree1)' see what prints out and discuss. |

## 1.2. __INIT__ method in python

| Discuss about special variables and methods. | |
|---|---|
| Special variables<br>__name__ | special method<br>__init__ |
| | |

*Table 1: object oriented programing (code) structure basic 1*

Example of a basic structure of an object oriented programing (code) Further explanation of the structure will be discussed on further chapters and subchapters.

```
class Tree:
    def __init__(self,attribute,behavior):  # initialization
        self.a = attribute
        self.b = behavior

    def config(self):
        print("The tree is: ", self.a, self.b)

tree1 = Tree("Cactus", "with few leafs")    # object creation or constructor
tree2 = Tree("Eucalyptus", "with dense leafs")

tree1.config()
tree2.config()
```

```
The tree is:  Cactus with few leafs
The tree is:  Eucalyptus with dense leafs
```

*Table 2: object oriented programing (code) structure basic 2*

Another example of basic structure of an object oriented programing (code) Further explanation of the structure will be discussed on further chapters and subchapters.

```
class conMaterial:
    def __init__(self,name,size):
        self.name = name
        self.size =size


material1 = conMaterial('column', 6)
material2= conMaterial('beam', 3)

print(material1.name, material1.size)
```

```
column 6
```

## 1.3. Constructor, self and comparing objects in python

Discuss about heap memory, id()/ address of an object or variable and their sizes. The size of an object depends up on the number of variables and their sizes. The decision/ assigning of the size of the variables or objects is done by your constructor. Every time you run a function the object takes different places/ id in your heap memory.

*Box 3: Heap memory and id*

### 1.3.1. Constructor

*Table 3: Constructor*

Discuss about constructors by making use of the following code. This part is a revision of the persiouse exercises and intended to discuss further about the constructor.

```
class Person:
    def __init__(self):
        self.name = 'Kebede'
        self.age = 30

p1   = Person()    # constructor
p2 = Person()     # constructor (The constructor calls the __init__ function by
default when you run your code.)

p1.name = 'Selamawit' #You can change values of objects
p1.age= 35

print(p1.name)
print(p1.age)
print(id(p1))
print(id(p2))
print(id(p1.name))
print(id(p1.age))
```

```
Selamawit
35
1058695602128
1058695602032
1058695320752
1058688953712
```

### 1.3.2. Self and comparing objects

#### *1.3.2.1. Self and update*

*Table 4: Self and update*

Discuss about self by making use of the following code via the update function.

```
class Person:
    def __init__(self):
        self.name = 'Kebede'
        self.age = 30
    def update(self):    #we can update values
        self.age = 15

p1   = Person()
p2 = Person()
```

```
p1.name = 'Selamawit'
p1.age= 35

p1.update()    # the last output of the running program will be according to this
update on the indicated instance. Here the self-function works by referring to the
indicated instance as well. In this case, the age the instance that will be updated
is age of p1. In case where we have several instances we will indicate them
respectively and apply the update. P2.update(), p3.update(), p4.update(),
p3.update(), etc.

print(p1.name)
print(p1.age)
```

```
Selamawit
15
```

### 1.3.2.2. Self and comparing objects

*Table 5: Self and comparing objects 1*

See the line of codes below and discuss about defining a comparison function and discuss about the purpose of self.

```
class Person:
    def __init__(self):
        self.name = 'Kebede'
        self.age = 30
    def compare(self,any): # we are defining/creating compare here as it is not a
built in function.
        if self.age == any.age: return True
        else: return False
p1  = Person()
p2 = Person()
p1.age=25
p2.age=25
if p1.compare(p2): # we are calling the defined created and defined compare function
to execute comparison.  /creating
    print("They are of the same age.")
else: print("They are of different age.")
```

```
They are of the same age.
```

*Table 6: Self and comparing objects 2, list*

Self ,comparison

```
class Person:
    def __init__(self):
        self.name = 'Kebede'
        self.age = 30
    def compare(self,any):
        if self.age == any.age: return True
        else: return False
p1  = Person()
p2 = Person()
p1.age=60 # we are changing value
p2.age=45 # we are changing value
if p1.compare(p2):
```

Self, list , comparison

```
class Person:
    def __init__(self):
        self.name = 'name'
        self.age = 30

    def compare2(self,z):
        if self.age>=z:return True
        else: return False

y = [p1, p2, p3, p4] = Person(),
Person(), Person(), Person()  #
Constructor. We are creating objects
```

```python
    print("They are of the same age.")
else: # we are adding an else function
    print("They are of different age!")
```

```python
vis list.
x = [p1.age, p2.age, p3.age, p4.age] = 80, 50, 30, 100  # we are changing value via list
z=sum(x)/len(x)

if p1.compare2(z):
    print("Older than most.")

else: # we are adding an else function
    print("Younger than most!")
print(max(x)) # We are inquiring max value of the list
print(min(x))
print(z)
```

```
They are of different age!
```

```
Older than most.
100
30
65.0
```

## 1.4. Types of variables in python in oop (Attributes).

There are two types of variables in oop: (1) instance variable (2) class (static) variable. See table below.

> Name-space is an area where you create and store objects/ variables. There are two kinds of name spaces: (1) class name-space where you store class variables (2) object (instance) name space where you store instance variables (Placeholder1).

*Box 4: Name space*

| Variables in oop | | |
|---|---|---|
| Class variables | Static variables | |
| Instance variables | | |

*Figure 2: Types of variables in oop*

*Table 7: Instance and Class/ Static Variables*

| Instance variables are declared in the init function/ inside the constructor. | Class or static variables are declared outside the init function/ outside the constructer. |
|---|---|

```python
class Table:
    def __init__(self):
        self.price=300 # instance variable
        self.material='timber' # instance variable

table1 = Table()
table2 = Table()

table1.price=500  # we can change the value of instance variable

print(table1.price, table1.material)
print(table2.price, table2.material)
```

```python
class Table:
    name = 'TABLE' #class variable
    def __init__(self):
        self.price=300 # instance variable
        self.material='timber' # instance variable

table1 = Table()
table2 = Table()

Table.name= "standup table" # we can change the value of class variable that will affect all objects of classes

table1.price=500  # we can change the value of instance variable
```

| | |
|---|---|
| | `print(table1.price, table1.material, table1.name)` |
| 500 timber<br>300 timber | 500 timber TABLE<br>300 timber TABLE |

## 1.5. Types of methods in python oop (behavior)

There are three types of methods in oop: (1) Instance (have two types: Accessor method and Mutator method), (2) class and (3) static.

Unlike variables in oop class and static are different. In instance methods: If we are fetching variables we use Accessor methods while we use Mutators to modify variables.

| Methods (functions) in oop | | |
|---|---|---|
| Instance method | Accessors/ getters | Passes self and works with instance variables |
| | Mutators/ setters | |
| Class methods | | Passes class and works with class variables |
| Static | | It neither passes class or instance variables nor is it concerned with their variables. |

*Figure 3: types of methods (functions) in oop*

### 1.5.1. Instance methods

*Table 8: Instance Method*

Instance methods (Accessors/ getters and Mutators/ setters), instance methods work with instance variables.

```
class Vegetable:


    market = 'markato' # this is class variable


def __init__(self,p1,p2,p3): #this is initialization function (init function)
        self.p1=p1
        self.p2=p2
        self.p3=p3

    def avg(self):  # this is instance method, and it passes self and works with instance variables.
        return(self.p1 + self.p2 + self.p3)/3 # here self.p1 , self.p2 and self.p3 are instance variables


    def get_p1(self): #instance method, type Accessor, getter, it fetches a variable or a value
        return self.p1
    def set_p1(self,value): #instance method, type Mutator, setter, it sets or changes a variable or a value
        self.p1=value

v1=Vegetable(10,15,30)
v2=Vegetable(50,100,74)
v3=Vegetable(60,63,90)
```

```
print(v2.avg())#prints average price of vegetable 2 or v2, this calls the instance
method
```

```
74.66666666666667
```

### 1.5.2. Class methods

Class method , class methods work with class variables

```
class Vegetable:
    market = 'Merkato' # this is class variable. It can be called by class methods.
    def __init__(self,p1,p2,p3): #this is initialization function (init function)
        self.p1=p1
        self.p2=p2
        self.p3=p3

    def avg(self):   # this is instance method, and it passes self.
        return(self.p1 + self.p2 + self.p3)/3

    @classmethod #a dicorator
    def getmarket(cls):  # this is class method, and it passes class. It works with
class variable.
        return cls.market

v1=Vegetable(10,15,30)
v2=Vegetable(50,100,74)
v3=Vegetable(60,63,90)

print(v2.avg())
print(Vegetable.getmarket())
```

```
74.66666666666667
Merkato
```

### 1.5.3. Static methods

We work with instance method when we are concerned with instance variables. We work with class methods when we are concerned with class variables. When we are not concerned with either instance or class variables, we can work with static method.

```
class Vegetable:
    market = 'Merkato' # this is class variable. It can be called (not named) by class methods.
    def __init__(self,p1,p2,p3): #this is initialization function (init function)
        self.p1=p1
        self.p2=p2
        self.p3=p3

    def avg(self):   # this is instance method, and it passes self.
        return(self.p1 + self.p2 + self.p3)/3

    @classmethod #a decorator for class method
    def getmarket(cls):  # this is class method, and it passes class.
        return cls.market
    @staticmethod #a decorator for static method
    def description(): # this is static method, and it is not concerned with either
```

```
instance or class variables..
        print("This is information about vegetable price at market Merkato.")

v1=Vegetable(10,15,30)
v2=Vegetable(50,100,74)
v3=Vegetable(60,63,90)

print(v2.avg()) # this calls the instance method
print(Vegetable.getmarket()) # this calls the class method

Vegetable.description() # this calls the static method
```

```
74.66666666666667
Merkato
This is information about vegetable price at market Merkato.
```

## 1.6. Inner class in python oop

Here we will study how to create inner class or class with in a class.

See the following codes and discuss about the defined function show.

```
class conMaterial:
    def __init__(self,name,size):
        self.name = name
        self.size =size


material1 = conMaterial('column', 6)
material2= conMaterial('beam', 3)

print(material1.name, material1.size)
```

```
column 6
```

```
class conMaterial:
    def __init__(self,name,size):
        self.name = name
        self.size =size
    def show(self):
        print(self.name, self.size)


material1 = conMaterial('column', 6)
material2= conMaterial('beam', 3)

material1.show()
```

```
column 6
```

To create inner class with in an outer class without creating an external file, we can define additional class under init function.

You can create object of inner class inside the outer class or you can create object of inner class outside the outer class provided you use outer class to call it.

```
class conMaterial: # outer class
    def __init__(self,name, size):
        self.name=name
        self.size=size
        self.ing =self.Ingredient() # create the object of the inner class in the
outer class.

    class Ingredient: # create a separet inner class
        def __init__(self):
            self.fine='sand'
            self.coarse='gravel'
            self.metal='steel'
```

```
material1=conMaterial('column', 6)
material2=conMaterial('beam', 10)

ing1 = material1.ing
ing2 = material2.ing

# print(material1.name, material1.size)
#print(material1.ing.fine, material1.ing.coarse , material1.ing.metal)
#print(material2.ing.fine, material2.ing.coarse, material2.ing.metal)

print(ing1.metal, ing1.coarse, ing1.fine)
```

```
steel gravel sand
```

Define the show function as follows as shown below.

```
class coMaterial: # outer class
    def __init__(self,name, size):
        self.name=name
        self.size=size
        self.ing =self.Ingredient() # create the object of the inner class in the
outer class.

    def show(self):
        print(self.name,self.size)
        self.ing.show()

    class Ingredient: # create a separet inner class
        def __init__(self):
            self.fine='sand'
            self.coarse='gravel'
            self.metal='steel'

        def show(self):
            print(self.fine, self.coarse, self.metal)


material1=coMaterial('column', 6)
material2=coMaterial('beam', 10)

material1.show()
```

```
column 6
sand gravel steel
```

## References

(Under development)

## Appendix

(Under development)