Beginner’s Guide to Python 3 Programming

Python Programming Paradigms

Python is hybrid programming language as it allows you to write very procedural code, to use objects in an object oriented manner and to write functional programs

1. **Procedural Programming**: represented as a sequence of instructions that tell the computer what it should do explicitly.
2. **Declarative Programming**: allow developers to describe how a problem should be solved, with the language/environment determining how the solution should be implemented.
3. **Object Oriented Programming**: approaches that represent a system in term of the objects that system. Each object ca hold its own data (also known as state) as well as define behavior that defines what object can do.
4. **Functional programming**: Language decompose a problem into a set of functions. Each function is independent of any external state, operating only on the inputs they received to generate their outputs.

Python as interpreted language

An interpreted language is one that does not require a separate compilation phase to convert the human readable format into something that can be executed by a computer. Instead the plain text version is fed into another program (generally referred to as the interpreter) which then executes the program for you.

`print ()` function

it will print whatever user give, when user give a string it will print string, when it’s given an integer it’s will print integer, also for other data type as float and others

Hello world

```
Print (“Hello World”)
```

Hello world with a variable

```
A = “hello world”
Print(A)
```

```
>> hello world
```

`input ()` function

to use when it prompts user for input. All input will stored as a strina data.

Prompting a value

```
User_name = input (“Enter your name :”)
Print (“hello”, “ “, User_name)
```

Prompting for numerical input

```
High= input (“enter your body high: “)
High = int (High)

Pi= input (“enter pi value : “)
Pi = float (Pi)
```

Python variables

Python as Dynamic Typing. That is the type of the data held by a variable can Dynamically change as the program executes

```
my_variable = 'John'
print(my_variable)
my_variable = 42
print(my_variable)
my_variable = True
print(my_variable)
```

```
>>John
>>42
>>True
```

Python Strings

a string is a series, or sequence, of characters in order. In this definition a character is anything you can type on the keyboard in one keystroke

Representing Strings

```
‘Hello World’
“Hello World”
```

String Concatenation

```
String_1=’Good’
String_2=”day”
String_3= String_2 + String_2
print(string_3)
```

```
>> Good day
```

Accessing Character form a String

```
my_string = 'Hello World'
print(my_string[4]) # characters at position 4
print(my_string[1:5]) # from position 1 to 5
print(my_string[:5]) # from start to position 5
print(my_string[2:]) # from position 2 to the end
```

Repeating Strings

```
print (‘*’ * 10)
print (‘Hi’ * 10)
```

```
>> **********
>> HiHiHiHiHiHiHiHiHiHi
```

Splitting Strings

```
title = 'The Good, The Bad, and the Ugly'
print('Source string:', title)
print('Split using a space')
print(title.split(' '))
print('Split using a comma')
print(title.split(','))
```

```
>>Source string: The Good, The Bad, and the Ugly
>>Split using a space
>>['The', 'Good,', 'The', 'Bad,', 'and', 'the', 'Ugly']
>>Split using a comma
>>['The Good', ' The Bad', ' and the Ugly']
```

Counting Strings

```
my_string = 'Count, the number of spaces'
print("my_string.count(' '):", my_string.count(' '))
```

```
>> my_string.count(' '): 8
```

Replacing Strings

```
welcome_message = 'Hello World!'
print(welcome_message.replace("Hello", "Goodbye"))
```

```
>> Goodbye World!
```

Finding Sub Strings

`String.find(string_to_find)`

The method returns −1 if the string is not present. Otherwise it returns an index indicating the start of the substring

```
print('Edward Alun Rawlings'.find('Alun'))
```

```
>> 5
```

```
print('Edward John Rawlings'.find('Alun'))
```

```
>> -1
```

Other Strings Operations

```
some_string = 'Hello World'
print('Testing a String')
print('-' * 20)
print('some_string', some_string)
print("some_string.startswith('H')",
some_string.startswith('H'))
print("some_string.startswith('h')",
some_string.startswith('h'))
print("some_string.endswith('d')", some_string.endswith('d'))
      print('some_string.istitle()', some_string.istitle())
      print('some_string.isupper()', some_string.isupper())
      print('some_string.islower()', some_string.islower())
      print('some_string.isalpha()', some_string.isalpha())
      print('String conversions')
      print('-' * 20)
      print('some_string.upper()', some_string.upper())
      print('some_string.lower()', some_string.lower())
      print('some_string.title()', some_string.title())
      print('some_string.swapcase()', some_string.swapcase())
      print('String leading, trailing spaces', " xyz ".strip())
```

```
Testing a String
--------------------
some_string Hello World
some_string.startswith('H') True
some_string.startswith('h') False
some_string.endswith('d') True
some_string.istitle() True
some_string.isupper() False
some_string.islower() False
some_string.isalpha() False
String conversions
--------------------
some_string.upper() HELLO WORLD
some_string.lower() hello world
some_string.title() Hello World
some_string.swapcase() hELLO wORLD
String leading, trailing spaces xyz
```

Converting Other Type into Strings

```
msg = 'Hello Lloyd you are ' + str(21)
print(msg)
```

Beginner’s Guide to Python 3 Programming

Number in Python

There are three types number used in python; integer, float and complex number

```
X=1 #integer
X=1.0 #float
X = 2 + 2 j #complex
```

Converting to integer

```
String_value= ‘200’
Float_value=4.3

String_to_int = int(String_value)
Float_to_int =int(Float_value)
```

Converting to float

```
String_value=’203’
Integer_value=23

String_to_int=float(String_value)
Float_to_int=float(Integer_value)
```

Access complex number component

```
X = 2 + 2j
Print(‘Real:‘ , X.real , ‘imagn :’ , X.imag)
```

Arithmetic Operator

Add	1+2 =3
Subtract	5-2 = 3
Multiple	5*2 =10
Divide	10/2 = 5
Integer Division	7//3 = 2
Modulus	13%3 = 1
Exponent	2**3 = 8

Assignment Operator

```
x+=2    -> x=x+2
x-=2    -> x=x-2
x*=2    -> x=x*2
x/=2    -> x=x/2
x//=2   -> x=x//2
x%=2    -> x=x%2
x**=2   -> x=x**2
```

None Value

This is used to represent null values or nothingness

```
winner = None
print('winner:', winner)
print('winner is None:', winner is None)
print('winner is not None:', winner is not None)
```

winner: None
winner is None: True
winner is not None: False

If Statement

This statement is used to control the flow of execution within a program based on some condition

```
if <condition-evaluating-to-boolean>:
   statement
```

Comparison Operator

=	-> equal
!=	-> not equal
<	-> less than
>	-> greater than
<=	-> less than or equal
>=	-> greater than or equal

Logical Operator

and -> Returns True if both left and right are true
or -> Returns two if either the left or the right is truce
not -> Returns true if the value being tested is False

Using If

```
num = int(input('Enter a number: '))
if num < 0:
        print(num, 'is negative')
```

Using else in an If

```
num = int(input('Enter yet another number: '))
if num < 0:
 print('Its negative')
else:
 print('Its not negative')
```

The Use of elif

```
savings = float(input("Enter how much you have in savings: "))
if savings == 0:
 print("Sorry no savings")
elif savings < 500:
 print('Well done')
elif savings < 1000:
 print('Thats a tidy sum')
elif savings < 10000:
 print('Welcome Sir!')
else:
 print('Thank you')
```

If expression

<result1> if <condition-is-met> else <result2>

```
age=int(input("enter your age:"))
Status=None

Status=('teenager' if age>12 and age<20 else 'note teenager')
print(Status)
```

Iteration/Looping

The while loop and the for loop available in Python. These loops are used to control the repeated execution of selected statements.

While Loop

```
while <test-condition-is-true>:
    statement or statements
```

```
count=0
Upper_limit=int(input("Enter the upper limit:"))
print('Starting')
while count<=Upper_limit:
    print(count, ' ', end='')
    count+=1
print()
print('Done')
```

For Loop

```
for i = from 0 to 10
   statement or statements


for <variable-name> in range(...):
      statement statement
```

For in range

```
print('Print out values in a range')

for i in range(0,10):
    print(i, ' ', end='')
print()
print("Done")
```

For in range with increment

```
print('Print out values in a range')

for i in range(0,10):
            print(i, ' ', end='')
        print()
        print("Done")
```

Print out something in for

```
for _ in range(0,10):
    print('*', end='')
print()
```

Break loop

```
print('only print code if all iterations completed')
num= int(input('Enter a number to check for:'))
for i in range(0,6):
    if i==num:
        break
    print(i,' ',end='')
print('Done')
```

Continue loop

```
for i in range (0,10):
    if i % 2==1:
        continue
    print(i, ' ', end='')
print()
print('Done')
```

For loop with else

The else part is executed if and only if all items in the sequence are processed

```
print('Only print code if all iterations completed')

num=int(input('Enter a number to check for:'))
for i in range(0,6):
    if i==num:
        break
    print(i, ' ', end='')
else:
    print()
          print('All iterations succesful')
```

Factorial using for loop

```
number=int(input("Enter a number:"))

faktorial=1
if number<0:
    print("The factorial is not defined")
elif number==0:
    print("0! =", faktorial)
else:
    for i in range(1,number+1):
faktorial=faktorial*i
          print(number,"! = ",faktorial)
```

Prime number check

This program able to check if a number is a prime number or not

```
num=int(input("Enter a number:"))
if num>1:
    for i in range(2,num):
        if(num%i)==0:
            print(num,"is not a prime number")
            break
    else:
        print(num,"is a prime number")

else:
    print(num, "is not a prime number")
```

Print prime numbers in range

```
lower = int(input("Enter lower number:"))
upper = int(input("Enter upper number:"))

print("Prime numbers between", lower, "and", upper, "are:")

for num in range(lower, upper + 1):
   if num > 1:
       for i in range(2, num):
           if (num % i) == 0:
               break
       else:
           print(num)
```

Beginner's Guide to Python 3 Programming

Recursion

Recursion is a programming solution which is a function call itself one or more times in order to solve a particular problem

Recursive Behaviour

- The key here is that an overall problem can be solved by breaking it down into smaller examples of the same problem.
- Functions that solve problems by calling themselves are referred to as recursive functions.
- For a recursive function to the useful it must therefore have a termination condition.

Factorial using Recursion

```
def fact(n):
    if n==1:
        return n
    else:
        return n*fact(n-1)
```

Cek prime number using Recursion

```
def is_prime(number,a=2):
    prime=True
    if a>=number:
        print("",end='')
    elif(number%a)==0:
        prime=False
    else:
        return is_prime(number,a+1)
    return prime
```

Pascal triangle using Recursion

Pascals triangle is a triangle of the binomial coefficients. The values held in the triangle are generated as follows: In row 0 (the topmost row), there is a unique nonzero entry 1. Each entry of each subsequent row is constructed by adding the number above and to the left with the number above and to the right, treating blank entries as 0

```
def pascal_triangle(n,i=0):
    if i>=n:
        print("",end="")
    else :
        #print (i)
        for j in range (n-i-1):
            print(end=" ")
        for j in range (i+1):

          print(fact(i)//(fact(j)*fact(i-j)), end=" ")
        print()
        pascal_triangle(n,i+1)
```

Intro Structured Analysis – Functional Decomposition

Functional Decomposition is one way in which a system can be broken down into its constituent parts

Functional Decomposition Terminology

The key terms used within Functional Decomposition are:

- **Function**. This is a task that is performed by a device, system or process.
- **Decomposition**. This is the process by which higher level functions are broken down into lower level functions where each function represents part of the functionality of the higher level function.
- **Higher Level Function**. This is a function that has one or more sub functions.
- **Sub Function**. This is a function that provides some element of the behaviour of a higher level function.
- **Basic Function**. A basic function is a function that has no smaller sub functions

Function in Python

Python functions are groups of related statements that can be called together, that typically perform a specific task, and which may or may not take a set of parameters or return a value.

Defining Functions

```
def function_name(parameter list):
     """docstring"""
     Statement
     statement(s)
```

Returning value from a Functions

```
def square(n):
      return n * n
```

Returning multiple value from a Functions

```
def swap(a, b):
      return b, a
a = 2
b = 3
x, y = swap(a, b)
print(x, ',', y)
```

Local and Global Variable

"In practice developers usually try to limit the number of global variables in their programs as global variables can be accessed anywhere and can be modified anywhere and this can result in unexpected behaviours (and has been the cause of many, many bugs in all sorts of programs over the years)"

Local Variable

```
def my_function():
   a_variable = 100
   print(a_variable)
```

Global Keyword

```
max = 100
def print_max():
   global max
   max = max + 1
   print(max)
```

nonlocal Keyword

```
   def outer():
      title = 'original title'
      def inner():
           nonlocal title
           title = 'another title'
           print('inner:', title)
        inner()
        print('outer:', title)
```

Functional Programming

Is a programming paradigm, a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data

Functional Programming Points

- **Focussed on the computational side** of computer programming
- Generate results based **purely on the data** provided to them
- Functions **only rely on their inputs** to generate a new output
- **Do not generate on any side effects** and **do not depend on the current state** of the program

- Functional Programming aims to **avoid side effects**
- Functional Programming **avoids concepts such as state**
- Functional Programming **promotes immutable data**
- Functional Programming **promotes declarative programming**

Higher Order Function

A function that takes another function as a parameter is known as a higher order function.

Simple Example

```
def apply(x,function):
    result=function(x)
    return result
def mult(y):
    return y*10.0
print(apply(5,mult))
```

Functions returning functions

```
def make_funtion(s):
    if s=='add':
        def adder(x,y):
            return x+y
        return adder
    elif s=='subs':
        def substarct(x,y):
            return x-y
        return substarct
    else:
        raise ValueError('Unkown request')

f1=make_funtion('add')
f2=make_funtion('subs')
print(f1(3,2))
print(f2(3,2))
```

Beginner's Guide to Python 3 Programming

Curried Function

A curried function in Python is a function where one or more of its parameters have been applied or bound to a value, resulting in the creation of a new function with one fewer parameters than the original.

```
def multiply(a,b):
    return a*b

def multby(func,num):
    return lambda y:func(num,y)

double=multby(multiply,2)
print(double(5))
```

Closures

Closure allows a function to reference a variable available in the scope where the function was originally defined, but not available by default in the

```
def increment(num):
    return num + 1
def reset_function():
    global increment
    addition = 50
    increment = lambda num: num +
  addition

print(increment(5))
reset_function()
print(increment(5))

>>6
>>55
```

Object Orientation

OOP provides an approach to structuring programs/applications so that the data held, and the operations performed on that data, are bundled together into classes and accessed via objects

Class in Python

In Python everything is an object and as such is an example of a type or class of things. For example, integers are an example of the int class, real numbers are.

Define a class

```
class nameOfClass(SuperClass):
        __init__
        attributes
        methods
```

__init__

__init__ method are local variables and will disappear when the method terminates. This is an initializer (also known as a constructor) for the class. It indicates what data must be supplied when an instance of the Person class is created and how that data is stored internally

Attributes

Attributes are instance variable that the class has

Method

A method is the name given to behaviour that is linked directly to the Person class; it is not a free-standing function rather it is part of the definition of the class

```
class Person:
  def __init__(self,name,age):
          self.name=name
          self.age=age
  def __str__(self):
          return self.name+' is '+str(self.age)
  def birthday(self):
           print('Happy birthday you were ',self.age)
          self.age+=1
          print('You are now', self.age)
  def calculate_pay(self,hours_worked):
          rate_of_pay=7.50
          if self.age>=21:
             rate_of_pay+=2.50
```

Class variable

Referred to as class variable or attributes (as opposed to instance variables or attributes).

```
class Person:
    instance_count=0 #class variable
    def __init__(self,name,age):
        Person.instance_count+=1
        self.name=name
        self.age=age

p1=Person('Jason',36)
p2=Person('carol',21)
p3=Person('James',19)
print(Person.instance_count)
```

Class method

Decorated with @classmethod and take a first parameter which represents the class rather than an individual instance. Class method linked with class, rather than an individual object. Class method didn't need an instance, but still need other attributes in the class

```
class Person:
   instance_count=0
   @classmethod
   def increamnet_instance_count(cls):
     cls.instance_count+=1

   def __init__(self,name,age):
      Person.increamnet_instance_count()
      self.name=name
      self.age=age

p1=Person("mario",23)
print(Person.instance_count)
```

Static method

decorated with the @staticmethod, that is same as free standing function, but is defined within class. These method didn't other attributes or instances.

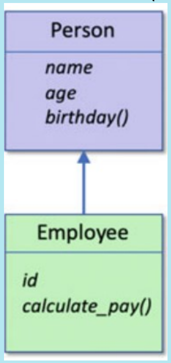
```
class Person:
   def __init__(self,name,age):
      self.name=name
      self.age=age
   @staticmethod
   def BMI(heigh,weight):
      heigh=heigh/100
      return weight/(heigh*heigh)

p1=Person('Mario',23)
print(p1.name,' is ',p1.age,', BMI
           :',Person.BMI(163,52))
p2=Person('Tiara',10)
print(p2.name,' is ', p2.age, ', BMI
           :',Person.BMI(110,29
```

Class Inheritance

Inheritance allows features defined in one class to be inherited and reused in the definition of another class.

Example

That is one class (in this case the Employee class) can inherit features from another class (in this case Person).



```
class Person:
      def __init__(self, name, age):
           self.name = name
           self.age = age
      def birthday(self):
           print('Happy birthday you were', self.age)
           self.age += 1
           print('You are now', self.age)

  class Employee(Person):
      def __init__(self, name, age, id):
           super().__init__(name, age)
           self.id = id
      def calculate_pay(self, hours_worked):
           rate_of_pay = 7.50
           if self.age >= 21:
           rate_of_pay += 2.50
           return hours_worked * rate_of_pay
```

Multiple Inheritances

Referred to as class variable or attributes (as opposed to instance variables or attributes).

```
class A:
   def __str__(self):
      return 'A'
   def print_info(self):
      print('A')
class B:
   def __str__(self):
      return 'B'
class C:
   def __str__(self):
      return 'C'
   def get_data(self):
      return 'CData'
class D:
   def __str__(self):
      return 'D'
   def print_info(self):
      print('D')
class E:
   def __str__(self):
      return 'E'
   def print_info(self):
      print ('E')

class F(C,D,E):
   def __str__(self):
      return super().__str__()+'F'
   def get_data(self):
      return super().get_data()+'FData'
   def print_info(self):
      print('F'+self.get_data())
```