Python Basics

Table of Contents

Python Basics....1
1Python is an indented, object oriented, functional programming language....2
I. indented language....2
II. object oriented language....2
III. functional programming language....2
2Data structures....3
I.List....3
II.Dictionary....5
III.Tuple....6
IV.Set....7
3Exception handling in Python....8
4File operations....9
I. Reading line by line from a file....9
II. Writing line by line to a file....10
III. Reading and writing from/to file using with statement....10
5Simple functions in python....11

1 Python is an indented, object oriented, functional programming language

I. indented language

```
#  => Any new code block like in 'if','else','for' etc should be opened in a new column
#  example: Accept a number check it is even or odd
number = int(input("enter number:"))
if number%2 == 0:
   #new code block start
   print("number is an even number")
   #new code block end
else:
   #new code block start
   print("number is an odd number")
   #new code block end
```

II. object oriented language

```
# => every item in python is an object will have attributes and functions
# attribute and function of float variable
f1 = 23.45
#functions
result1 = f1.is_integer() #returns False f1 is a float value
print(result1) # prints False
result2 = f1.hex() # returns hexadecimal equivalent of 23.45 ie.'0x1.7733333333333p+4'
print(result2) # prints '0x1.7733333333333p+4'
#attributes or properties
result3 = f1.real # returns 23.45
result4 = f1.imag # returns 0.0
print(result3) # prints 23.45
print(result4) # prints 0.0
```

III. functional programming language

```
# => 1. a function can be passed as an argument to another function
def add(a,b):
   result = a+b
   return result
```

```
def function_two(add, n, m):
   result1 = add(n,m)
   return result1
result = function_two(add, 100, 200)
print(result) # prints 300
# => 2. a function can be returned as a result from a function
def get_function():
   def function_one(a,b):
      return a*b
   return function_one
func_1 = get_function()
result = func_1(100,200)
print(result) # prints 20000
```

2 Data structures

I. List

list is a collection of objects

```
>>> list1 = []
>>> dir(list1)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__delslice__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__',
'__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__setslice__',
'__sizeof__', '__str__', '__subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
>>> for i in [90,20,30,20, 35,110,34,65,12,13,9]:
...    list1.append(i)
...
>>> list1
[90, 20, 30, 20, 35, 110, 34, 65, 12, 13, 9]
>>> list1.sort.__doc__

'L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;\ncmp(x, y) -> -1, 0, 1'
>>> list1.sort()
>>> list1
[9, 12, 13, 20, 20, 30, 34, 35, 65, 90, 110]
>>> list1.reverse.__doc__

'L.reverse() -- reverse *IN PLACE*'
>>> list1.reverse()
>>> list1
[110, 90, 65, 35, 34, 30, 20, 20, 13, 12, 9]
>>> list.insert.__doc__
'L.insert(index, object) -- insert object before index'
>>> list1.insert(5,1000)
>>> list1
```

```
[110, 90, 65, 35, 34, 1000, 30, 20, 20, 13, 12, 9]
>>> list1.count(1000)
1
>>> list1
[110, 90, 65, 35, 34, 1000, 30, 20, 20, 13, 12, 9]
>>> list1.count.__doc__

'L.count(value) -> integer -- return number of occurrences of value'
>>> list1.count(20)
2
>>> len.__doc__

'len(object) -> integer\n\nReturn the number of items of a sequence or collection.'
>>> len(list1)
12
>>> list1.index.__doc__

'L.index(value, [start, [stop]]) -> integer -- return first index of value.\nRaises ValueError if the value is not
present.'
>>> list1.index(1000)
5
>>> list1.index(20)
7
>>> list.extend.__doc__
'L.extend(iterable) -- extend list by appending elements from the iterable'
>>> list2=[200,300,400]
>>> list1.extend.__doc__

'L.extend(iterable) -- extend list by appending elements from the iterable'
>>> list1.extend(list2)
>>> list1
[110, 90, 65, 35, 34, 1000, 30, 20, 20, 13, 12, 9, 200, 300, 400]
>>> list1
[110, 90, 65, 35, 34, 1000, 30, 20, 20, 13, 12, 9, 200, 300, 400]
>>> list1.remove.__doc__
'L.remove(value) -- remove first occurrence of value.\nRaises ValueError if the value is not present.'
>>> list1.remove(20)
>>> list1
[110, 90, 65, 35, 34, 1000, 30, 20, 13, 12, 9, 200, 300, 400]
>>> list1.pop.__doc__

'L.pop([index]) -> item -- remove and return item at index (default last).\nRaises IndexError if list is empty or
index is out of range.'
>>> list1.pop()
400
>>> list1

[110, 90, 65, 35, 34, 1000, 30, 20, 13, 12, 9, 200, 300]
>>> list1[-1]
300
>>> list1[len(list1)-1]
 300
>>> list1.__contains__.__doc__
'x.__contains__(y) <==> y in x'
>>> list1.__contains__(300)
```

```
True
>>> list1.__contains__(3000)
False
>>> list1


[110, 90, 65, 35, 34, 1000, 30, 20, 13, 12, 9, 200, 300]
```

II. Dictionary

Dictionary is a collection key : value pairs

```
>>> diction1 = {1:"one",2:"two",3:"three"}
>>> diction1

{1: 'one', 2: 'two', 3: 'three'}
>>> diction={}
>>> keys_list=[]
>>> values_list =[]
>>> for k in range(0,10):
...     keys_list.append(k)
…
>>> for v in range(100,1000,100):
...     values_list.append(v)
...
>>> keys_list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> values_list

[100, 200, 300, 400, 500, 600, 700, 800, 900]
>>> for i in range(0,len(keys_list)-1,1):
...     diction[keys_list[i]] = values_list[i]
...
>>> diction

{0: 100, 1: 200, 2: 300, 3: 400, 4: 500, 5: 600, 6: 700, 7: 800, 8: 900}
>>> diction1.keys()
[1, 2, 3]
>>> diction1.values()
['one', 'two', 'three']

>>> for k,v in diction1.items():
 print ("key : %s => value %s\n"%(str(k),str(v)))

key : 1 => value one
key : 2 => value two
key : 3 => value three

>>> diction1
{1: 'one', 2: 'two', 3: 'three'}
>>> diction1.pop.__doc__
```

```
'D.pop(k[,d]) -> v, remove specified key and return the corresponding value.\nIf key is not found, d is returned if given, otherwise KeyError is raised'
>>> diction1.pop(3)
'three'
>>> diction1
{1: 'one', 2: 'two'}

>>>
>>> diction1.update({2:"two hundred"})
>>> diction1
{1: 'one', 2: 'two hundred'}
>>> diction1.update({1:"one  hundred"})
>>> diction1

{1: 'one  hundred', 2: 'two hundred'}
>>> for i in diction1.items():
...     print (i)
...
(1, 'one  hundred')
(2, 'two hundred')
```

III. Tuple

```
# Tuple is an immutable data type, we cannot modify items in a tuple

>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> t=tuple(l)
>>> t
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> t[0]
0
>>> t[1]=1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> dir(t)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']

>>> t.count(0)
1
>>> t.count(9)
1
>>> t.index(8)
8
>>> t.index(4)
4
>>> t

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

IV. Set

Set is a collection of items with no duplicates

```
>>> l=range(10)
>>> l1=range(5,15)
>>> s=set(l)
>>> s1=set(l1)
>>> s
set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> s1
set([5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
>>> s1.__and__(s)
set([8, 9, 5, 6, 7])
>>> s1.__or__(s)
set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
>>> s1.__xor__(s)
set([0, 1, 2, 3, 4, 10, 11, 12, 13, 14])
>>> s1.__union__(s)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'set' object has no attribute '__union__'
>>> s1.union(s)
set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
>>> s1.intersection(s)
set([8, 9, 5, 6, 7])
>>> s1.issubset(s)
False
>>> s1.issuperset(s)
False
>>> s2
set([8, 9, 5, 6, 7])
>>> s2.issubset(s1)
True
>>> s1.issuperset(s)
False
>>> s1
set([5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
>>> s
set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> s2
set([8, 9, 5, 6, 7])
>>> s1.issuperset(s2)
True

>>>
>>> s2
set([8, 9, 5, 6, 7])
>>> s1.issuperset(s2)
True

>>> s2.pop()
8
>>> s2
set([9, 5, 6, 7])
```

```
>>> s2.pop()
9
>>> s2
set([5, 6, 7])
>>> s2.pop()
5
>>> s2

set([6, 7])
>>> s1
set([5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
>>> s1.remove(10)
>>> s1
set([5, 6, 7, 8, 9, 11, 12, 13, 14])
>>> s2.add(1000)
>>> s2
set([1000, 6, 7])
>>> s2.add(2000)
>>> s2
set([1000, 2000, 6, 7])

>>>
>>> l1.extend(l)
>>>
>>> l1
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> s1=set(l1)
>>> s1

set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
```

3 Exception handling in Python

To handle errors like division by zero, index error in list etc.,

```
def check_assertion(p):
    assert(p != 0),"P is not Zero"

def check_divisible_by_zero(number):
    try:
        n = number/0
    except ZeroDivisionError as e:
        print ("Zero division error occurred")
        return -1
    else:
        return 0
```

```
class MyExcep1(Exception):
     def __init__(self,value):
         self.value = value
     def __str__(self):
         return repr(self.value)

def check_myexception():
    try:
        raise MyExcep1("myException1 occured")
    except MyExcep1 as e:
        print ("MyExcep1 Exception occured")

def check_for_multiple_exceptions():
    try:
         for i in range(0,10):
             print i[i]
    except AssertionError as e:
         print (e)
    except ZeroDivisionError as e:
         print (e)
    except Exception as e:
         print (e)

def main():
   p = 0
   try:
      check_assertion(p)
   except AssertionError as e:
      print ("Assertion Error Exception occured")

   print (check_divisible_by_zero(100))

   check_myexception()

   check_for_multiple_exceptions()

if __name__ == "__main__":
    main()
```

4 File operations

I. Reading line by line from a file

```
file_name = "input.txt"

def read_all_lines_from_file(input_file):
    file_object = open(input_file, "r+")
    while True:
```

```
        line = file_object.readline()
        if line:
           print line
        else:
           break
    file_object.close()

def main():
    read_all_lines_from_file(file_name)

if __name__ == "__main__":
    main()
```

II. Writing line by line to a file

```
file_name = "input.txt"

def write_all_lines_from_file(output_file):
    file_object = open(output_file, "w")
    count = 0
    while count < 10:
         file_object.write("This is line number : %d\n"%int(count))
         count = count + 1
    file_object.close()

def main():
    write_all_lines_from_file(file_name)

if __name__ == "__main__":
     main()
```

III. Reading and writing from/to file using with statement

```
in_file_name="input.txt"
out_file_name="output.txt"


def read_and_write_lines_into_file(in_file_name, out_file_name):
    with open(in_file_name, "r+") as in_f:
        with open(out_file_name, "r+") as out_f:
            while True:
                line = in_f.readline()
                if line:
                    out_f.write("[OUTFILE]: %s"%line)
                else:
                    break
```

```
def main():
    read_and_write_lines_into_file(in_file_name, out_file_name)

if __name__ == "__main__":
   main()
```

5 Simple functions in python

```
def sum_of_n_numbers(list1):
    sum = 0
    for x in list1:
       sum = sum + x
    return sum

def factorial_of_n_numbers(list1):
    fact = 1
    for x in list1:
        fact = fact * x
    return fact

def even_or_odd_check(num):
    if num%2:
        return False
    else:
        return True

def prime_check(num):
    for x in range(2,num-1):
        if x!= num:
            if num%x == 0:
                return False
    return True

def main():
    l1 = [12,13,14,15,16,1,40,29,45]

    print ("Sum of N Numbers")
    print (sum_of_n_numbers(l1))

    print ("Factorial of N Numbers")
    print (factorial_of_n_numbers(l1))

    print ("Odd and Even Numbers check")
    for x in l1:
        print (x)
        print (even_or_odd_check(x))
```

```
    print ("Prime numbers check")
    for x in l1:
        print (x)
        print (prime_check(x))


if __name__ == '__main__':

    main()
```