Python Fundamentals day 9

Today's Agenda

- Return keyword
- Main function
- Taking user inputs

Return keyword

In the other programming languages like Java, C we have seen that

A function can take numerous parameters but return returns only single value. But in python not only does a function accept multiple values as input but also returns multiple values.

```python
def fun():
    a=10
    b=20
    c=30
    return a,b,c

print(fun())
```

Output:

```
In [1]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
(10, 20, 30)
```

By looking at the output format we can see that the values are stored in tuple. Which means, when a function returns multiple values it is stored inside tuple.

So now we know how to print. Let us see **how to store those values**

```
def fun():
    a=10
    b=20
    c=30
    return a,b,c

res=fun()
print(res)
print(type(res))
```

Output:

```
In [2]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
(10, 20, 30)
<class 'tuple'>
```

Now let us see **how to store in multiple variables**

```
def fun():
    a=10
    b=20
    c=30
    return a,b,c

res1,res2,res3=fun()
print(res1)
print(res2)
print(res3)
```

Output:

```
In [3]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
10
20
30
```

In the above output, the tuple is now unpacked and 10 is given to res1, 20 to res2 and 30 to res3.

Note: Number of variables should be equal to number of values returned by the tuple.

Main function – main( )

Let us create mymodule.py which consists of functions power_of and get_remainder and then import the module in test.py

mymodule.py

```python
def power_of(a,b):
    """this function calculates the result
    of a raise to the power of b"""
    c=a**b
    print(c)

def get_quotient(numerator,denominator):
    """This function calculates the quotient of
    numerator divided by denominator"""
    quotient=numerator/denominator
    print(quotient)

power_of(2,5)
get_quotient(100,2)
```

test.py

```python
import mymodule

def get_reminder(num,den):
    '''This function calculates the remainder of num/den'''
    rem = num%den
    print(rem)

get_reminder(100,6)
```

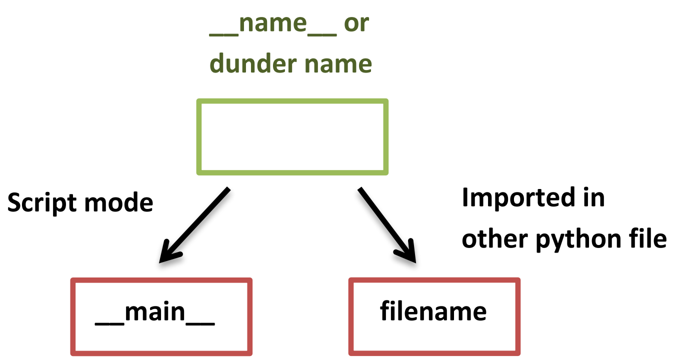
Output:

```
In [5]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
32
50.0
4
```

We can see in the above output, call to the functions from mymodule got automatically executed which is not what we wanted.

Whenever we import a module, we just want to import the functions in the module and don't want the function calls to get executed. So let us see how to do that

- Every python file has a variable name called as dunder name. The content inside this variable is based on whether the python file is executed in script mode or imported in other file.
- If it is executed in script mode, dunder name contains __main__
- If it is imported in other file, dunder name will be same as the file name.



- Now we know that, the function call commands must execute only when the python file is executed in script mode. So let us make the required changes in mymodule

```
def power_of(a,b):
    """this function calculates the result
    of a raise to the power of b"""
    c=a**b
    print(c)

def get_quotient(numerator,denominator):
    """This function calculates the quotient of
    numerator divided by denominator"""
    quotient=numerator/denominator
    print(quotient)

if __name__ == '__main__':
    power_of(2,5)
    get_quotient(100,2)
```

Now if we execute test.py we expect function calls to not get executed unless explicitly called for. Let us see if we succeed this time

```
In [6]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
4
```

Certainly we got as expected. Now let us call functions in mymodule explicitly in test.py and see the output

```
import mymodule

def get_reminder(num,den):
    '''This function calculates the remainder of num/den'''
    rem = num%den
    print(rem)

get_reminder(100,6)
mymodule.power_of(2,5)
mymodule.get_reminder(100,2)
```

Output:

```
In [8]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
4
32
50.0
```

With this information let us create a main()

```
def power_of(a,b):
    """this function calculates the result
    of a raise to the power of b"""
    c=a**b
    print(c)

def get_quotient(numerator,denominator):
    """This function calculates the quotient of
    numerator divided by denominator"""
    quotient=numerator/denominator
    print(quotient)

def main():
    power_of(2,5)
    get_quotient(100,2)

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

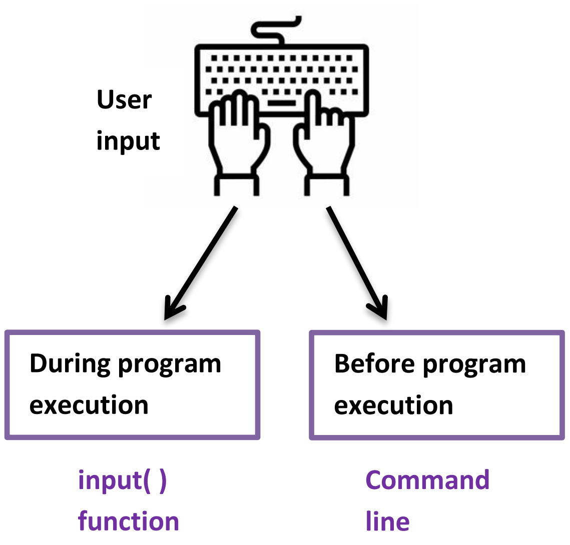
Output:

```
In [9]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
32
50.0
```

Now just like Java, C, C++ here in python also we have main() which is where the execution will start.

Taking user inputs

All clients expect the input to be given by user. So it is very important that we know how to take inputs from user.



- **During program execution – input()**

```
print("Enter the numerator")
num = input()
print("Enter the denominator")
den = input()

res = num/den
print(res)
```

Output:

```
In [10]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
Enter the numerator

10
Enter the denominator

2

  File "C:/Users/rooman/OneDrive/Desktop/
python/test.py", line 6, in <module>
    res = num/den

TypeError: unsupported operand type(s) for /:
'str' and 'str'
```

Note: Irrespective of what we enter as input, it is considered as string. As division cannot be performed on strings let us type cast it to integer.

```
print("Enter the numerator")
num = int(input())
print("Enter the denominator")
den = int(input())

res = num/den
print(res)
```

Output:

```
In [11]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
Enter the numerator

10
Enter the denominator

2
5.0
```

We can further more reduce the size of code and make it efficient as following

```
num = int(input("Enter the numerator\n"))
den = int(input("Enter the denominator\n"))

res = num/den
print(res)
```

Output:

```
In [12]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter the numerator
10

Enter the denominator
2
5.0
```

Before going to the next way of accepting inputs let us see another example

```
exp = input("Enter an expression\n")
res = eval(exp)
print(res)
```

eval() is a function which evaluates an expression.

Output:

```
In [13]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter an expression
3*5-1
14
```