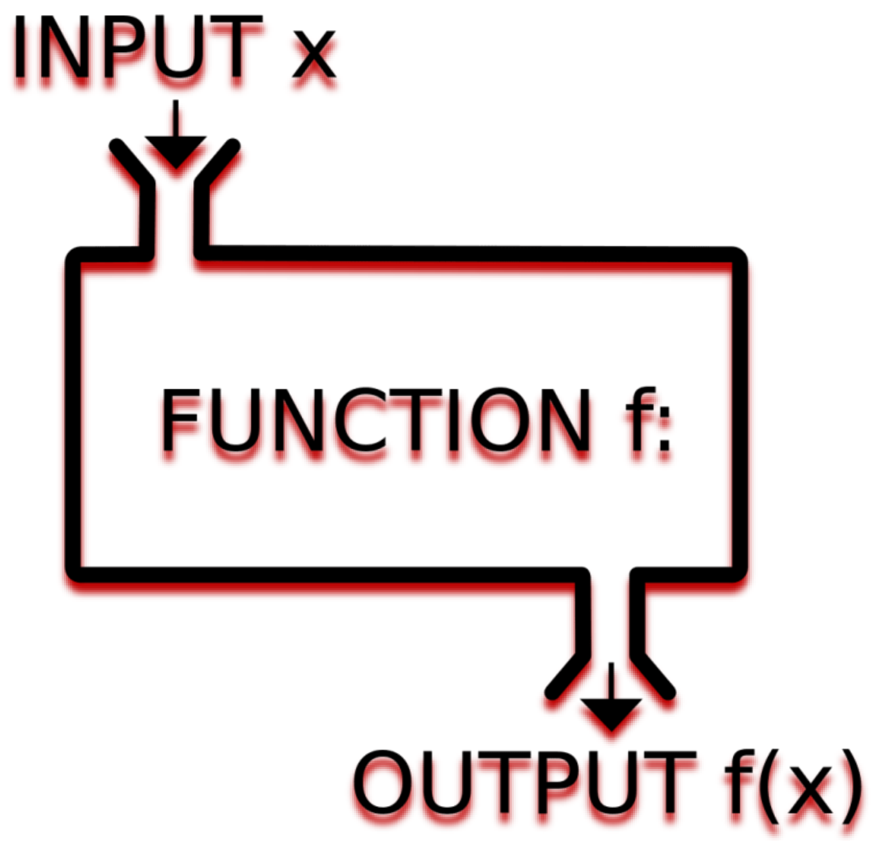
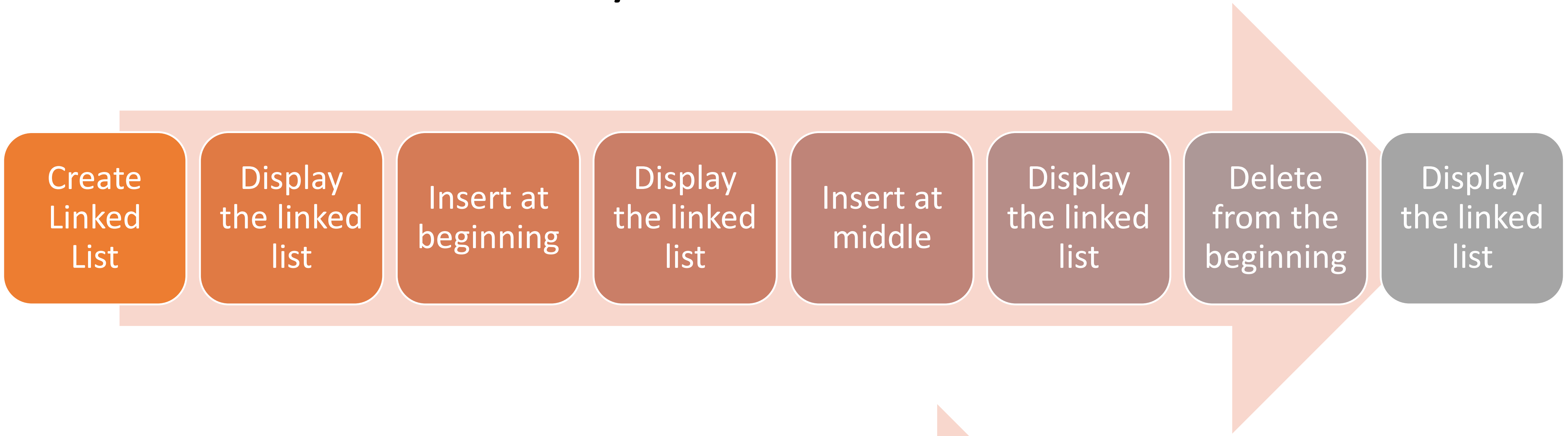


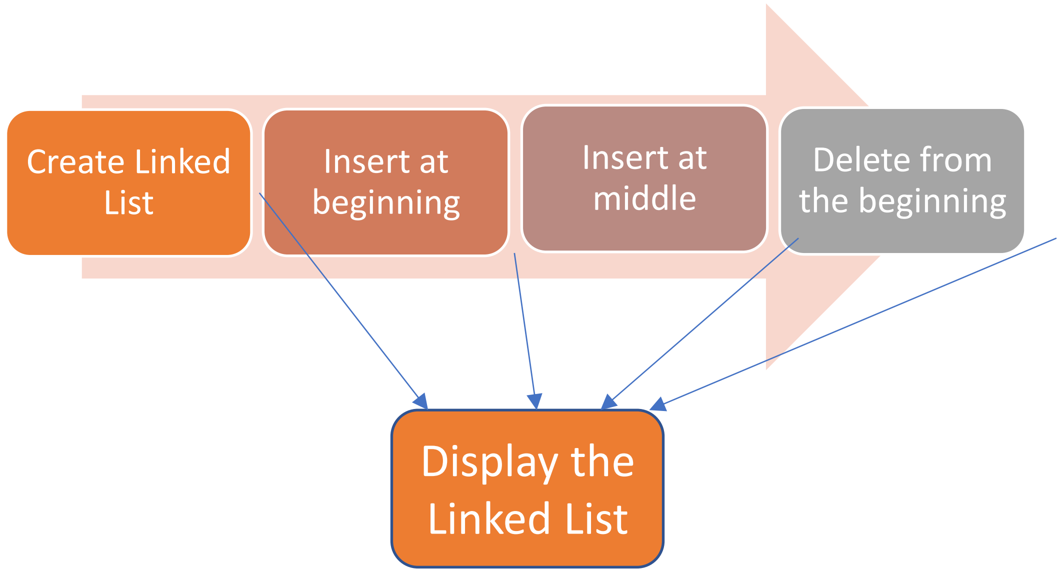
Function in Python

Introduction to Function

- Function is a group of statements that exists within a program for performing a specific task.
- A function accepts input arguments and produce an output by executing valid commands present in the function.
- Function name and file names need not be the same.
- Functions can be categorized into the following three types:
  - ➢Built-in
  - ➢Modules
  - ➢User-defined

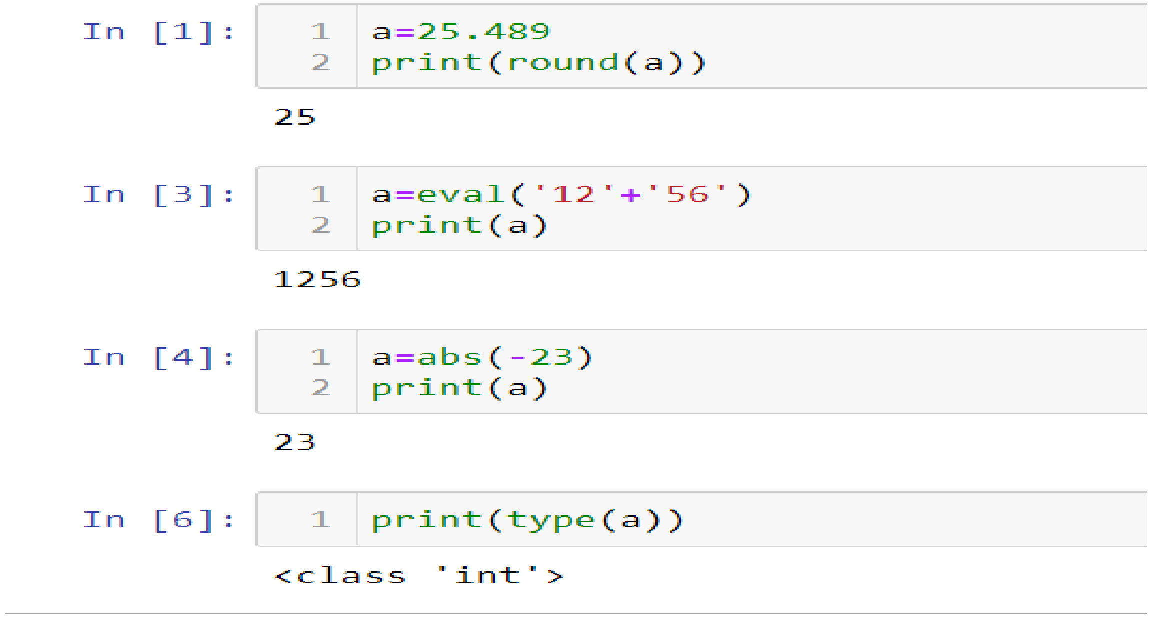
Why use function?





Built-in Functions

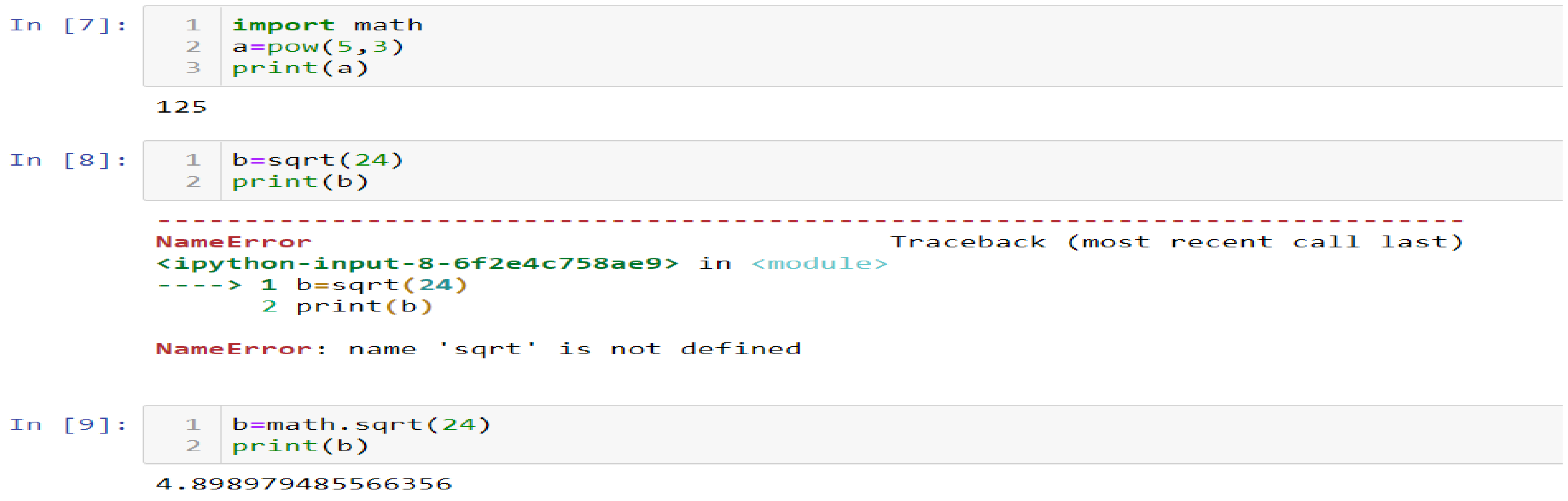
- Pre-defined functions are already available in Python
- It makes programming easier, faster and more powerful.
- Input functions
- Type conversion functions
- Eval function
- Abs function
- Round function
- Len function
- Type function



```
In [1]:  a=25.489
         print(round(a))

25

In [3]:  a=eval('12'+'56')
         print(a)

1256

In [4]:  a=abs(-23)
         print(a)

23

In [6]:  print(type(a))

<class 'int'>
```

Modules

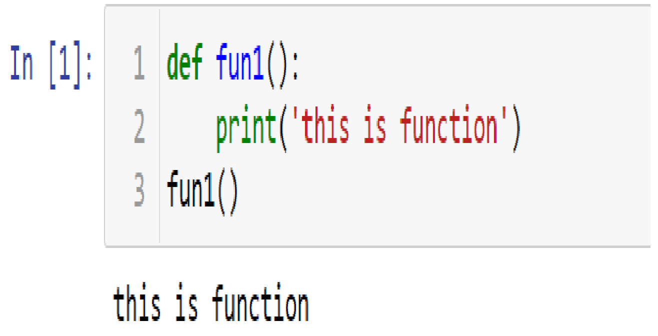
- A module is a file containing functions and variables
- It make easier to reuse the code.
- Commonly used modules are called libraries.



```
In [7]:  import math
         a=pow(5,3)
         print(a)

         125

In [8]:  b=sqrt(24)
         print(b)

         ---------------------------------------------------------------------------
         NameError                                 Traceback (most recent call last)
         <ipython-input-8-6f2e4c758ae9> in <module>
         ----> 1 b=sqrt(24)
               2 print(b)

         NameError: name 'sqrt' is not defined

In [9]:  b=math.sqrt(24)
         print(b)

         4.898979485566356
```

User Defined Function

- A user defined function is created or defined by the def statement followed by function name.
- Function name is followed by a pair of parenthesis and a colon with the statements to be executed indented as a block.
- Syntax: $def\ function - name(parameters)$:
  $Statements$
- An optional return statement to return a value from the function.

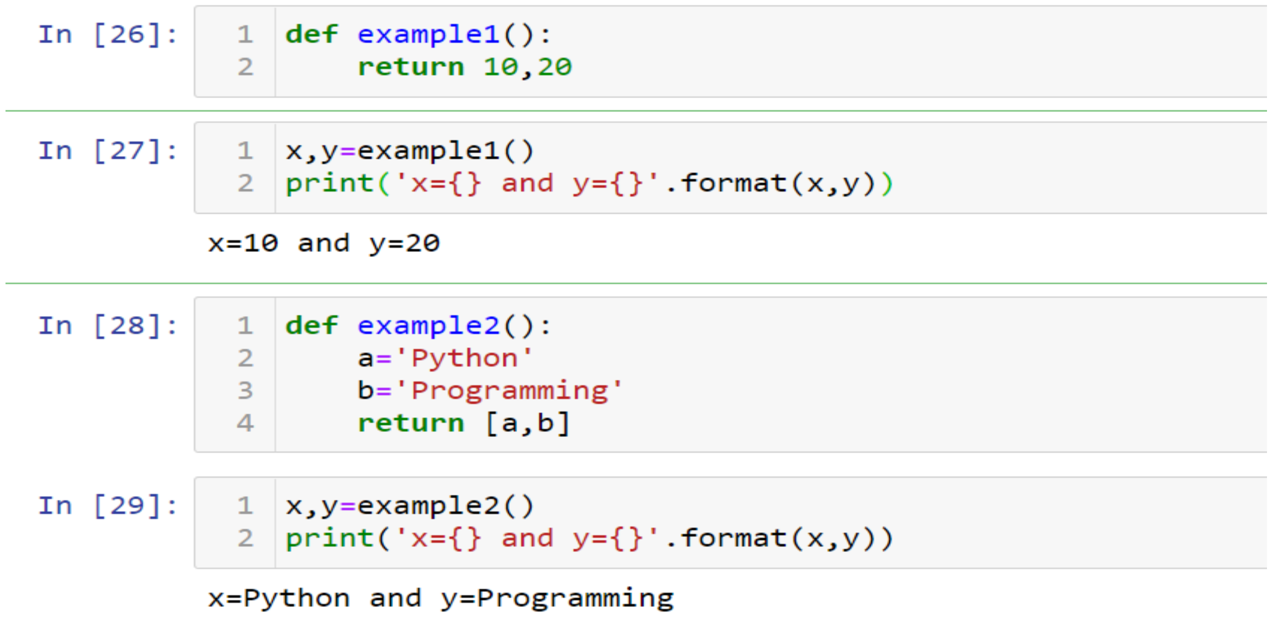


```
In [1]: def fun1():
            print('this is function')
        fun1()

this is function
```

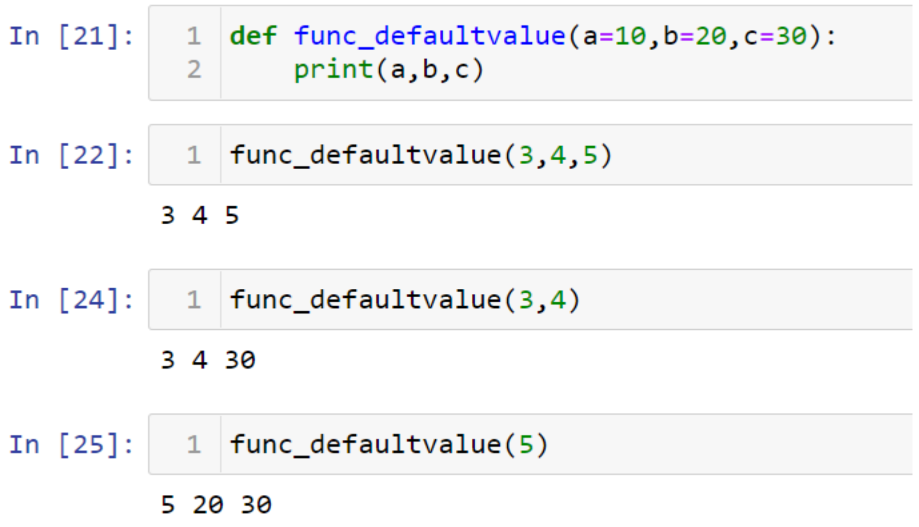
Exercise

- Write a program to check whether a number even or odd.
- Write a program to check whether a number prime or not.
- Write a program to check whether the inputted year is leap year or not.

- Default values can be set for function parameters.
- Function can return multiple values.



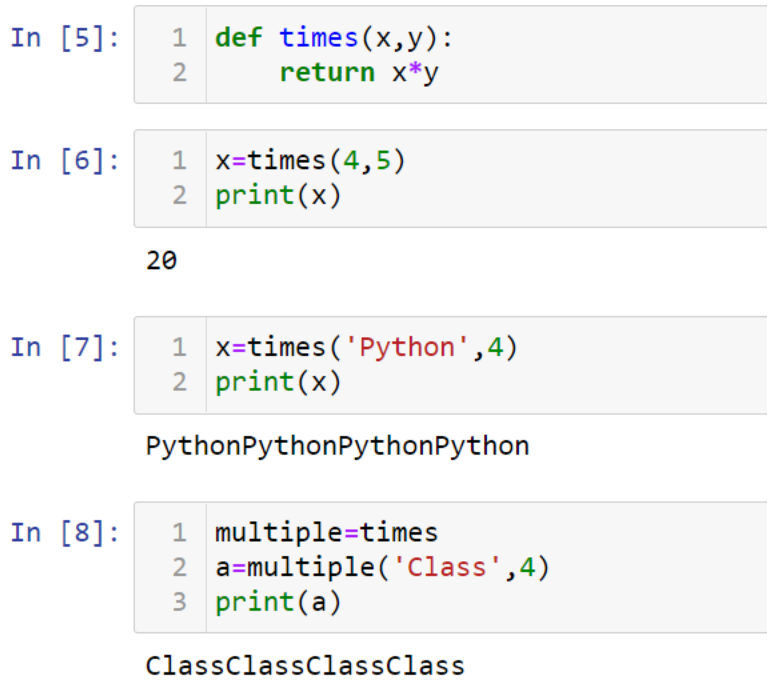
```
In [26]: def example1():
             return 10,20

In [27]: x,y=example1()
         print('x={} and y={}'.format(x,y))

         x=10 and y=20

In [28]: def example2():
             a='Python'
             b='Programming'
             return [a,b]

In [29]: x,y=example2()
         print('x={} and y={}'.format(x,y))

         x=Python and y=Programming
```



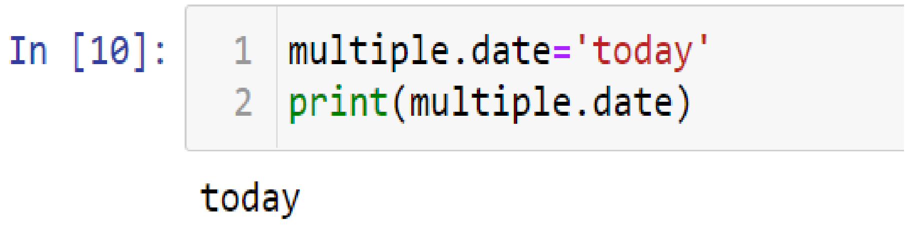
```
In [21]: def func_defaultvalue(a=10,b=20,c=30):
             print(a,b,c)

In [22]: func_defaultvalue(3,4,5)

         3 4 5

In [24]: func_defaultvalue(3,4)

         3 4 30

In [25]: func_defaultvalue(5)

         5 20 30
```

def execute at runtime

- Function can be assigned to different name and can be called through the new name.
- Functions are just objects, they are recorded explicitly in memory at program execution time.
- Functions allow arbitrary attributes to be attached to record information for later use:



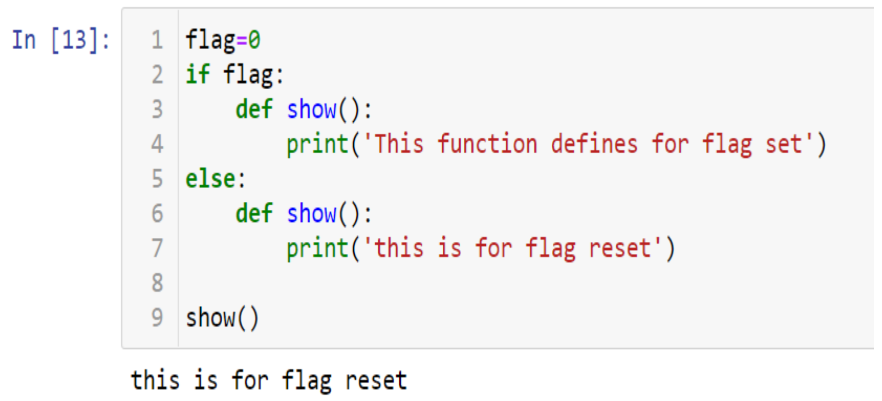
```
In [5]:  def times(x,y):
             return x*y

In [6]:  x=times(4,5)
         print(x)

20

In [7]:  x=times('Python',4)
         print(x)

PythonPythonPythonPython

In [8]:  multiple=times
         a=multiple('Class',4)
         print(a)

ClassClassClassClass
```



```
In [10]:  multiple.date='today'
          print(multiple.date)

today
```

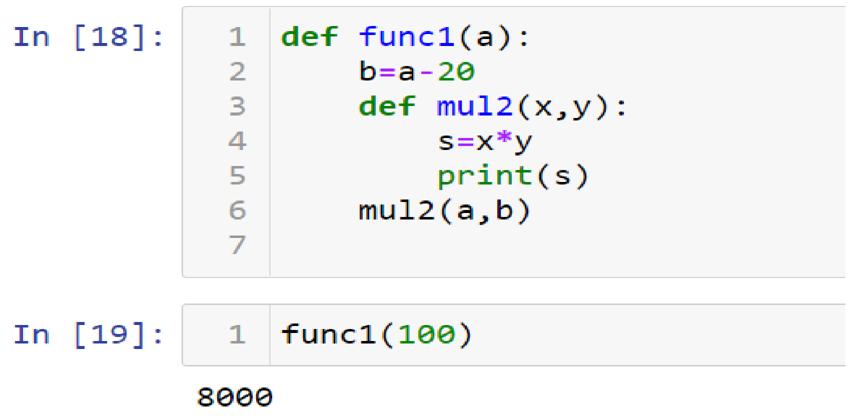
def execute at runtime

- The Python def is a true executable statement: when it runs, it creates a new function object and assigns it to a name.
- a def can appear anywhere a statement can—even nested in other statements.
- it's also completely legal to nest a function def inside an if statement to select between alternative definitions



In [13]:
```
flag=0
if flag:
    def show():
        print('This function defines for flag set')
else:
    def show():
        print('this is for flag reset')

show()
```

```
this is for flag reset
```



In [18]:
```
def func1(a):
    b=a-20
    def mul2(x,y):
        s=x*y
        print(s)
    mul2(a,b)

```

In [19]:
```
func1(100)
```

```
8000
```

Execution modes in Python

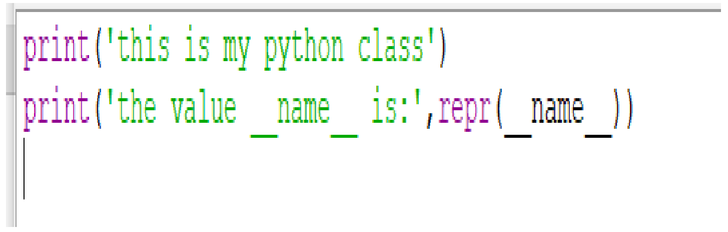
- Python code can be executed in two ways:

1. Can execute the Python file as a script using the command line.

$$python\ filename.py$$

2. Can import the code from one Python file into another file or into the interactive interpreter

$$import\ filename$$

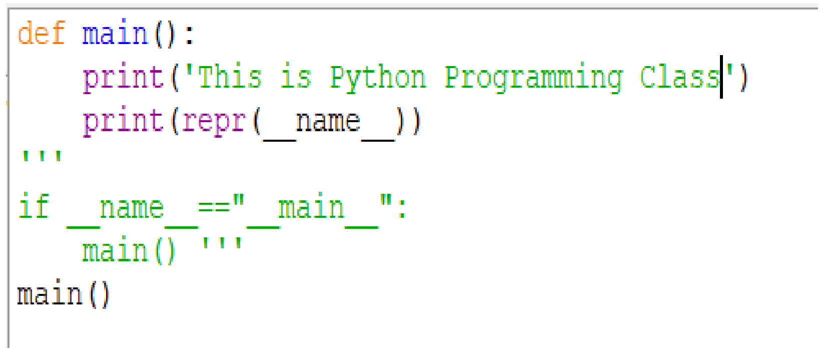


```
print('this is my python class')
print('the value __name__ is:',repr(__name__))
```

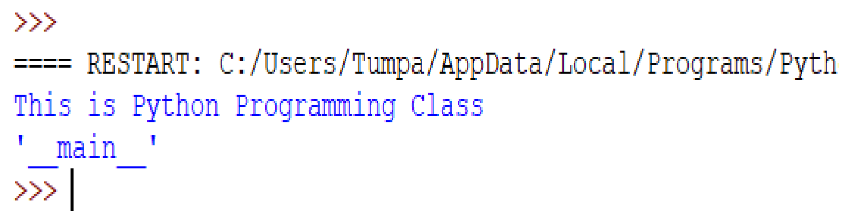
Executes the above code using two methods.

main function

- The Python interpreter executes scripts starting at the top of the file, and there is no specific function that Python automatically executes.
- having a defined starting point for the execution of a program is useful for understanding how a program works.



```
def main():
    print('This is Python Programming Class')
    print(repr(__name__))
'''
if __name__=="__main__":
    main() '''
main()
```



```
>>>
==== RESTART: C:/Users/Tumpa/AppData/Local/Programs/Pyth
This is Python Programming Class
'__main__'
>>>
```

Best Practice for Writing Code

- Put most code into a function or class.
- Use __name__ to control execution of your code.
- Create a function called main() to contain the code you want to run.
- Call other functions from main().

Python Scope

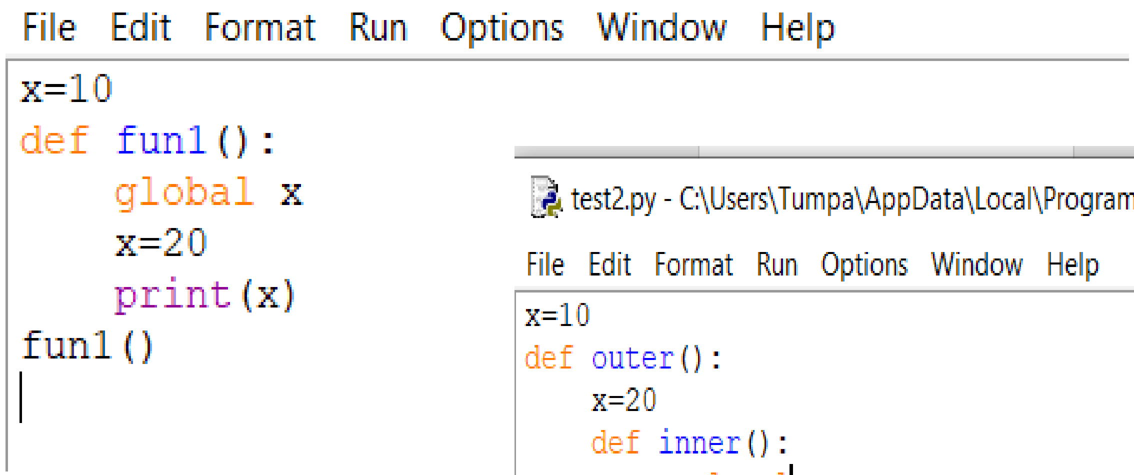
- Where a name is assigned in source code determines the namespace it will live in, and hence its scope of visibility.
- all names assigned inside a function are associated with that function’s namespace.
- Names defined inside a def can only be seen by the code within that def. You cannot even refer to such names from outside the function.
- Names defined inside a def do not clash with variables outside the def, even if the same names are used elsewhere.
- If a variable is assigned outside all defs, it is global to the entire file.

Scope Rules

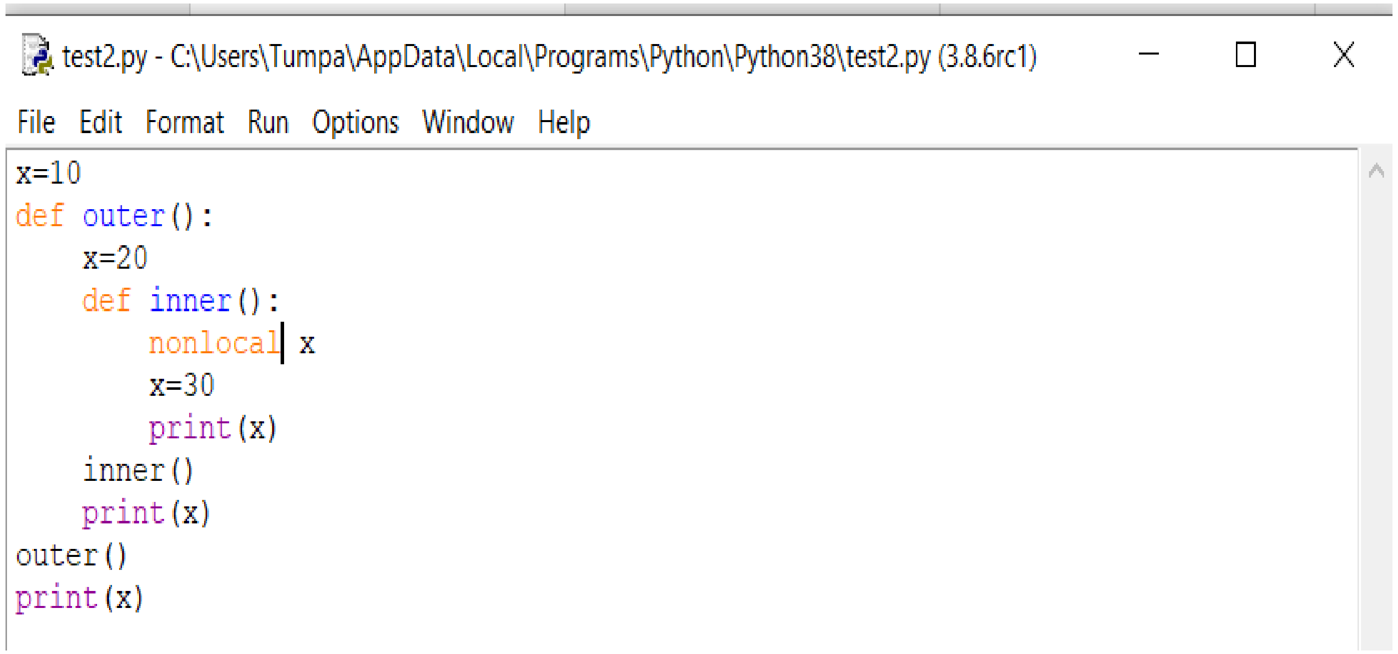
- The enclosing module is a global scope.
- Each call to a function creates a new local scope.
- Assigned names are local unless declared global or nonlocal.
- All other names are enclosing function locals, globals, or built-ins.

Name Resolution: The LEGB Rule

- Name references search at most four scopes: local, then enclosing functions (if any), then global, then built-in.
- Name assignments create or change local names by default.
- global and nonlocal declarations map assigned names to enclosing module and function scopes.



```
x=10
def fun1():
    global x
    x=20
    print(x)
fun1()
```


test2.py - C:\Users\Tumpa\AppData\Local\Programs\Python\Python38\test2.py (3.8.6rc1)
File Edit Format Run Options Window Help

```
x=10
def outer():
    x=20
    def inner():
        nonlocal x
        x=30
        print(x)
    inner()
    print(x)
outer()
print(x)
```