Syntax of defining function

```
def function-name([arguments]):
        '''doc string'''
        statement-1
        statement-2
```

"def" keyword is used to write a function in python
function-name is identifier/user defined word
arguments variables which receive values/input
a function is block, which must have at least one statement, in order to define function without any operation we can include "pass"

We can define function in two ways

1. Without arguments
2. With arguments

A function without arguments does not receive input/values
A function with arguments receive values/input

Example:

```
# function without arguments
def fun1():
   print("This is function1")
def fun2():
   print("This is function2")

fun1() # invoking function/calling function
fun1()
fun1()
fun2()
```

Output:

```
This is function1
This is function1
This is function1
This is function2
```

The memory for the function is called when function is invoked/called and de-allocated once the execution of function is completed

In python functions are treated as objects

Example:

```
def fun1():
    '''this is function1'''
    print("Hello fun1")


print(fun1.__name__)
print(fun1.__doc__)
print(type(fun1))
```

Output:

```
fun1
this is function1
<class 'function'>
>>>
```

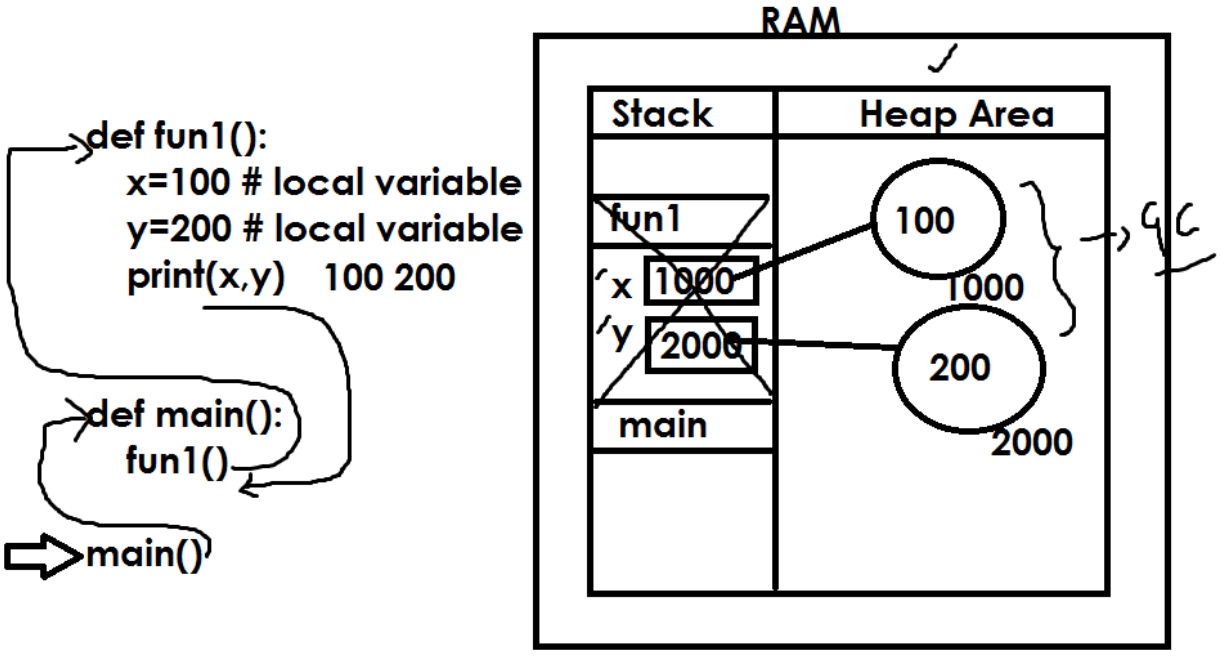
Example:

```
def draw_line():
    for i in range(20):
        print("*",end='')
    print()


def main():
    draw_line()
    print("PYTHON")
    draw_line()
    print("PROGRAMMING")
    draw_line()

main()
```

Output:

```
**********
PYTHON
**********
PROGRAMMING
**********
>>>
```

Local variables
The variables declared inside function are called local variables
Local variables scope is within function, we cannot access these variables outside function
Local variables memory is allocated when function is called and de-allocated after execution of function.



Example:

```
def fun1():
    x=100 # local variable
    y=200 # local variable
    print(x,y)

def main():
    fun1()

main()
```

Output:

```
100 200
>>>
```

Example:

```
def fun1():
    x=100 # local variable
```

```
    y=200 # local variable
    print(x,y)

def fun2():
    print(x,y)

def main():
    fun1()
    fun2()
main()
```

Output:

```
100 200
Traceback (most recent call last):
  File "C:/Users/user/Desktop/python6pm/py115.py", line 12, in <module>
    main()
  File "C:/Users/user/Desktop/python6pm/py115.py", line 11, in main
    fun2()
  File "C:/Users/user/Desktop/python6pm/py115.py", line 7, in fun2
    print(x,y)
NameError: name 'x' is not defined
>>>
```

Global variables

Variables declared outside the function are called global variables
Global variables memory is allocated when program is executed/run.

Example:

```
x=100 # 1
y=200 # 2
def fun1():
    print(x,y) # 100 200
def fun2():
    print(x,y) # 100 200

print(x,y) # 100 200 ==> 3

def main():
    print(x,y) # 100 200 ==> 5
    fun1() # 6
    fun2() # 7
```

```
main() # 4
```

Output:

```
100 200
100 200
100 200
100 200
>>>
```

Example:

```
def fun1():
    print(x,y)

fun1()
x=100
y=200

def main():
    fun1()

main()
```

Output:

```
Traceback (most recent call last):
  File "C:/Users/user/Desktop/python6pm/py117.py", line 4, in <module>
    fun1()
  File "C:/Users/user/Desktop/python6pm/py117.py", line 2, in fun1
    print(x,y)
NameError: name 'x' is not defined
>>>
```

Example:

```
# developing calculator
n1=int(input("enter first number"))
n2=int(input("enter second number"))
def add():
    print(f'sum is {n1+n2}')
def sub():
    print(f'diff is {n1-n2}')
```

```
def multiply():
    print(f'product is {n1*n2}')
def div():
    print(f'result is {n1/n2}')

def main():
    add()
    sub()
    multiply()
    div()
main()
```

Output:

```
enter first number10
enter second number5
sum is 15
diff is 5
product is 50
result is 2.0
>>>
```

Example:

```
x=100 # global variable
def fun1():
    y=200 # local variable
    print(x) # 100
    print(y) # 200

def fun2():
    z=300 # local variable
    x=400 # local variable
    print(x) # 400
    print(z) # 300

def main():
    fun1()
    fun2()
    print(x) # 100

main()
```

Output:

```
100
200
400
300
100
>>>
```

global keyword