GKTCS INNOVATIONS

Functions

What is Function in Python

- A function is a block of organized, reusable code that is used to perform a single, related action.

- Functions provide better modularity for your application and a high degree of code reusing.

- Python provides you many inbuilt functions like print(), but it also gives freedom to create your own functions.

How to define and call a function in Python

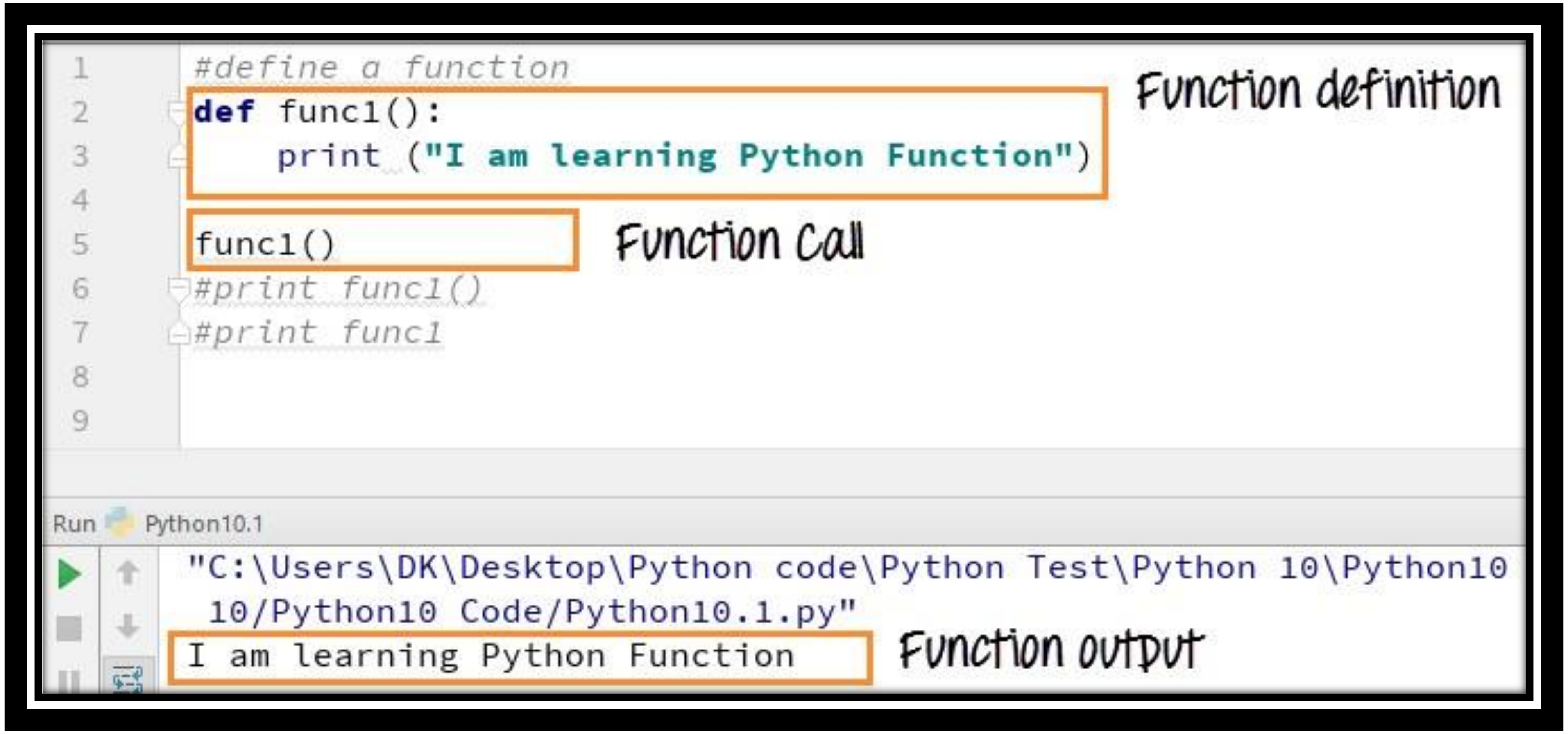
GKTCS INNOVATIONS

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses ( ).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon : and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Example:

Let us define a function by using the command " def func1():" and call the function. The output of the function will be **"I am learning Python function".**



```
#define a function
def func1():
    print ("I am learning Python Function")

func1()
#print func1()
#print func1
```

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\Python10
 10/Python10 Code/Python10.1.py"
I am learning Python Function
```

The function **print func1()** calls our def func1(): and print the command " **I am learning Python function None.**"

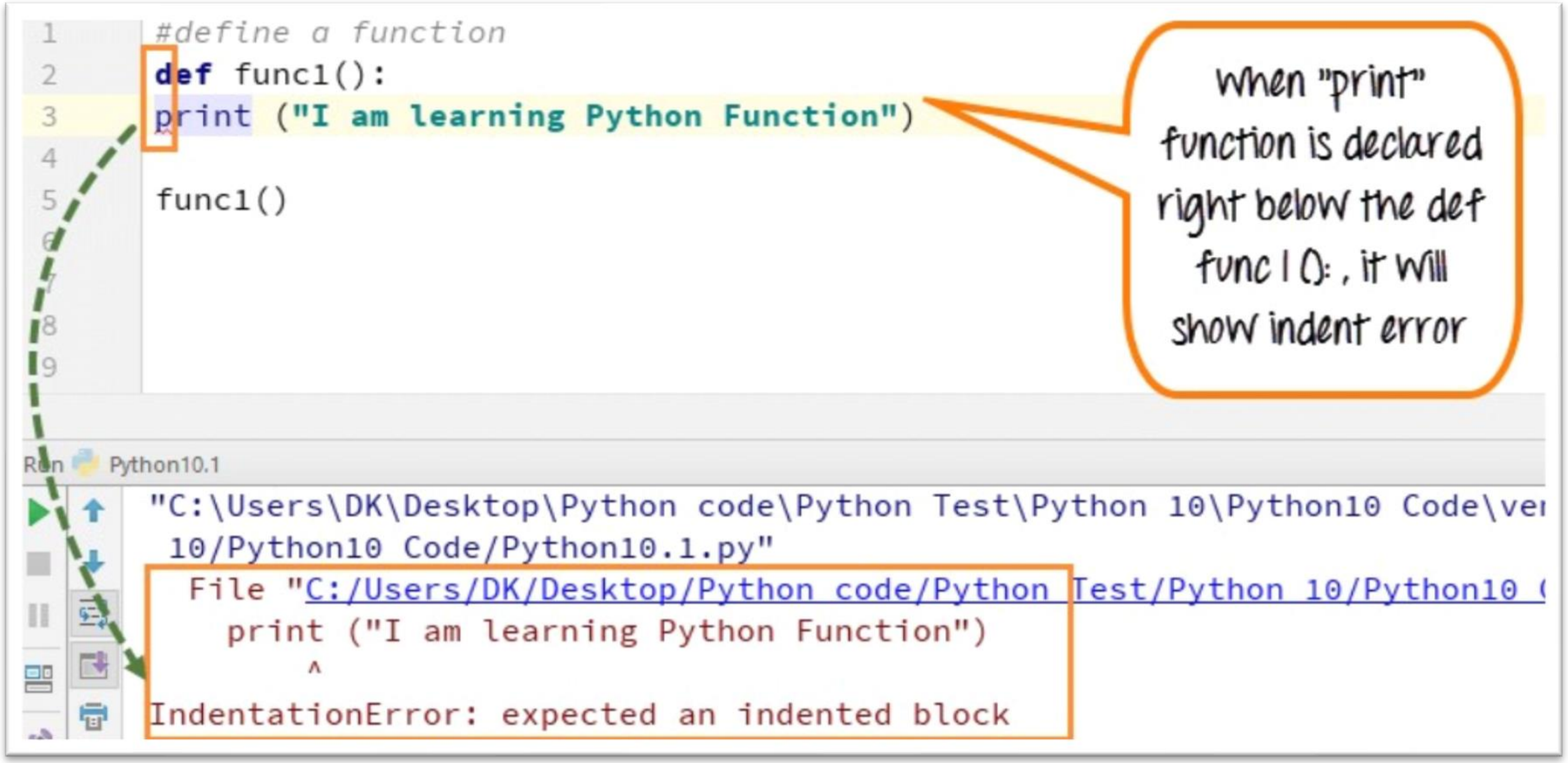
There are set of rules in Python to define a function.

- ❑ Any args or input parameters should be placed within these parentheses
- ❑ The function first statement can be an optional statement- docstring or the documentation string of the function
- ❑ The code within every function starts with a colon (:) and should be indented (space)
- ❑ The statement return (expression) exits a function, optionally passing back a value to the caller. A return statement with no args is the same as return None.

Significance of Indentation (Space) in Python

GKTCS INNOVATIONS

Before we get familiarize with Python functions, it is important that we understand the indentation rule to declare Python functions and these rules are applicable to other elements of Python as well like declaring conditions, loops or variable.

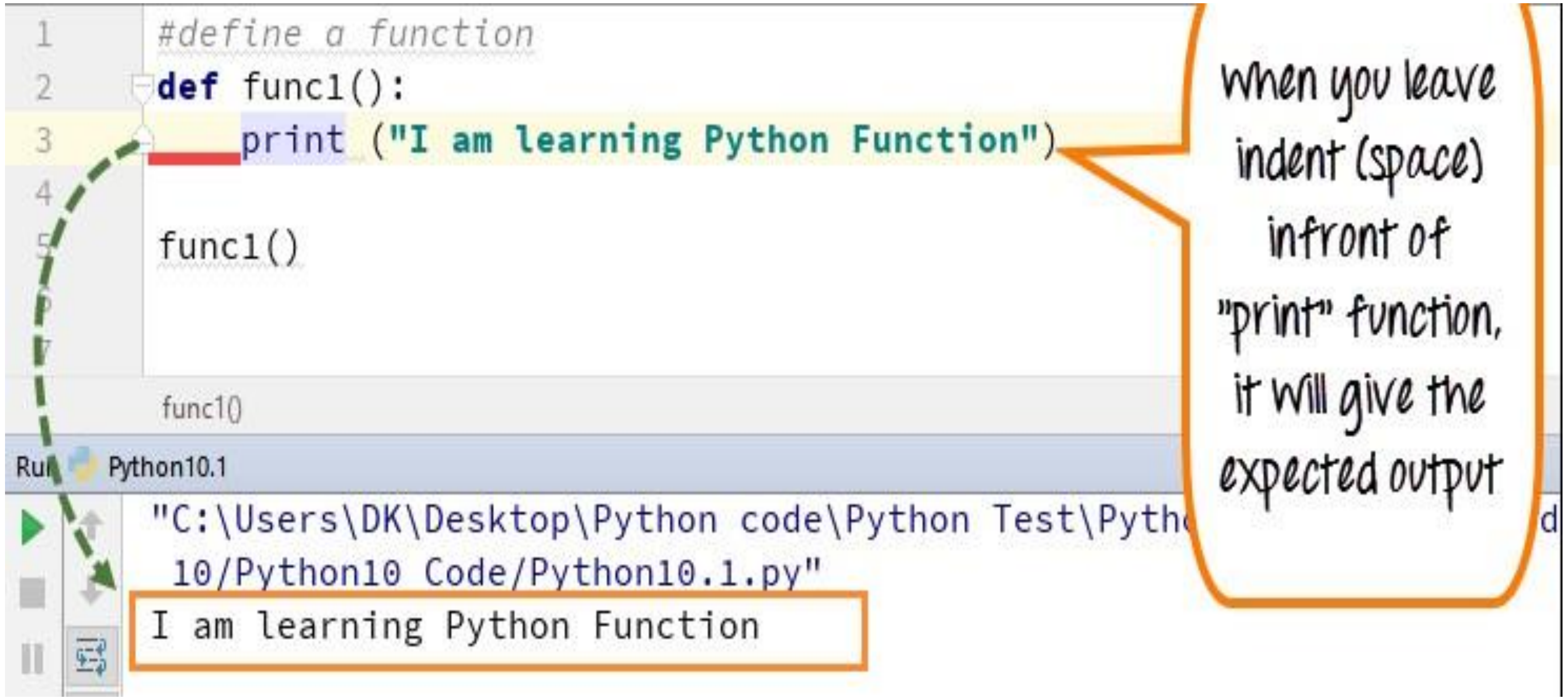


```
#define a function
def func1():
print ("I am learning Python Function")

func1()
```

When "print" function is declared right below the def func1(): , it will show indent error
Run Python10.1

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\Python10 Code\ve
 10/Python10 Code/Python10.1.py"
  File "C:/Users/DK/Desktop/Python code/Python Test/Python 10/Python10 (
    print ("I am learning Python Function")
        ^
IndentationError: expected an indented block
```

Now, when you add the indent (space) in front of "print" function, it should print as expected.



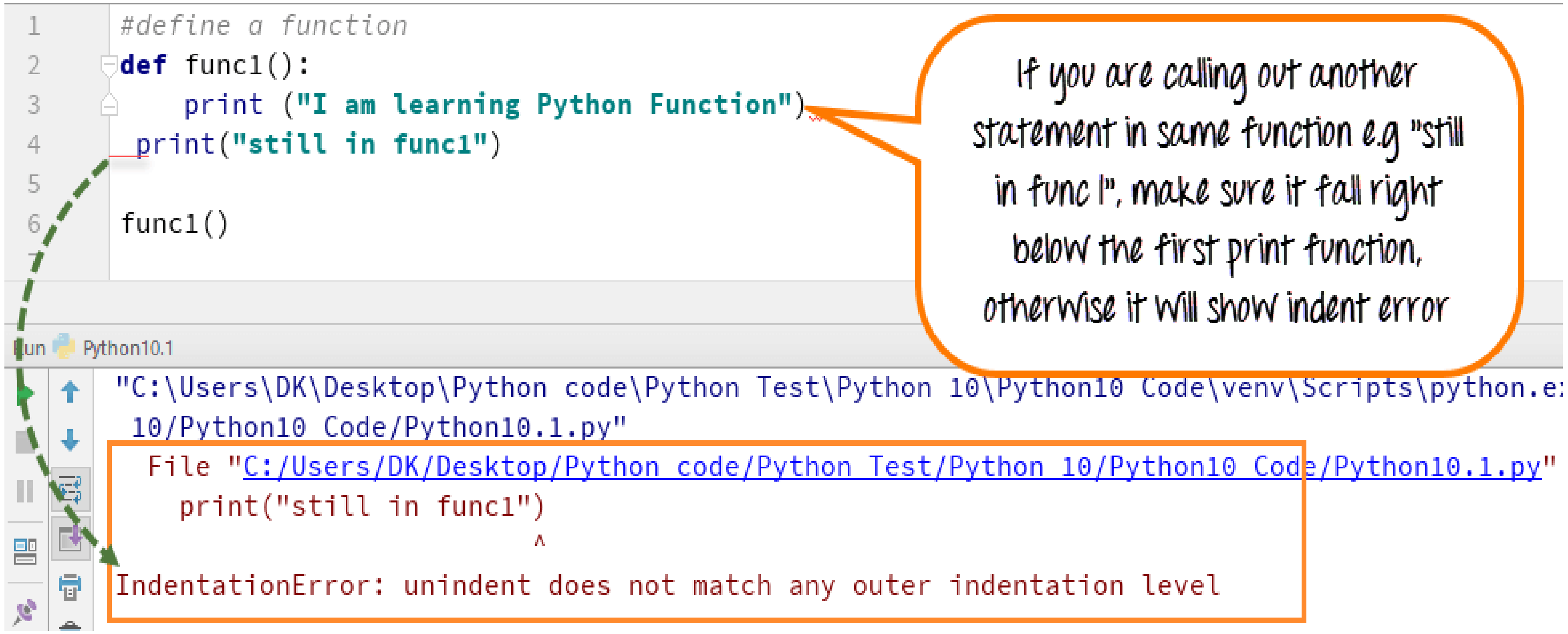
```
#define a function
def func1():
    print ("I am learning Python Function")

func1()
```

func1()
Run
Python10.1
When you leave indent (space) infront of "print" function, it will give the expected output

```
"C:\Users\DK\Desktop\Python code\Python Test\Pyth
 10/Python10 Code/Python10.1.py"
I am learning Python Function
```

At least, one indent is enough to make your code work successfully.
But as a best practice it is advisable to leave about 3-4 indent to call your function.

- It is also necessary that while declaring indentation, you have to **maintain the same indent for the rest of your code**.
- For example, in below screen shot when we call another statement "still in func1" and when it is not declared right below the first print statement it will show an indentation error **"unindent does not match any other indentation level."**

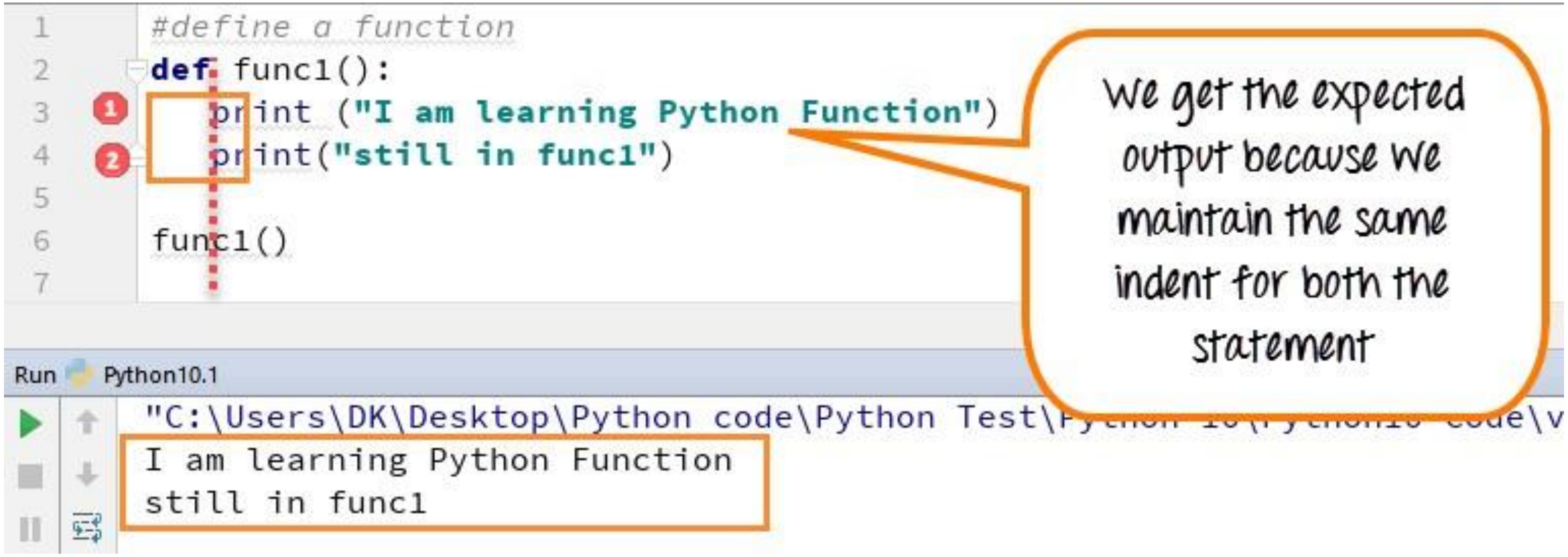


```
#define a function
def func1():
    print ("I am learning Python Function")
  print("still in func1")

func1()
```

If you are calling out another statement in same function e.g "still in func 1", make sure it fall right below the first print function, otherwise it will show indent error
Run Python10.1

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\Python10 Code\venv\Scripts\python.e
 10/Python10 Code/Python10.1.py"
  File "C:/Users/DK/Desktop/Python code/Python Test/Python 10/Python10 Code/Python10.1.py"
    print("still in func1")
                          ^
IndentationError: unindent does not match any outer indentation level
```

Now, when we apply same indentation for both the statements and align them in the same line, it gives the expected output.



```
#define a function
def func1():
    print ("I am learning Python Function")
    print("still in func1")

func1()
```

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\Python10 Code\v
I am learning Python Function
still in func1
```

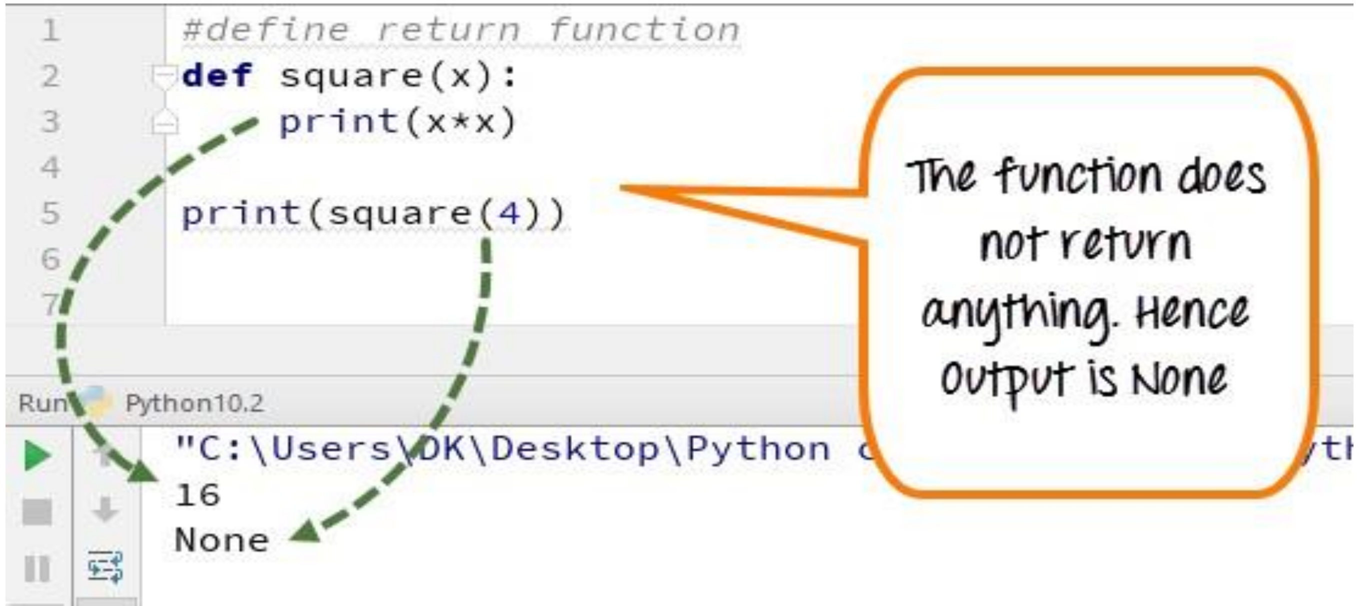
1
2
Run
Python10.1
We get the expected output because we maintain the same indent for both the statement

How Function Return Value?

Return command in Python specifies what value to give back to the caller of the function.

Step1)

Here - we see when function is not "return". For example, we want the square of 4, and it should give answer "16" when the code is executed. Which it gives when we simply use "print x*x" code, but when you call function "print square" it gives "None" as an output. This is because when you call the function, recursion does not happen and fall off the end of the function. Python returns "None" for failing off the end of the function.

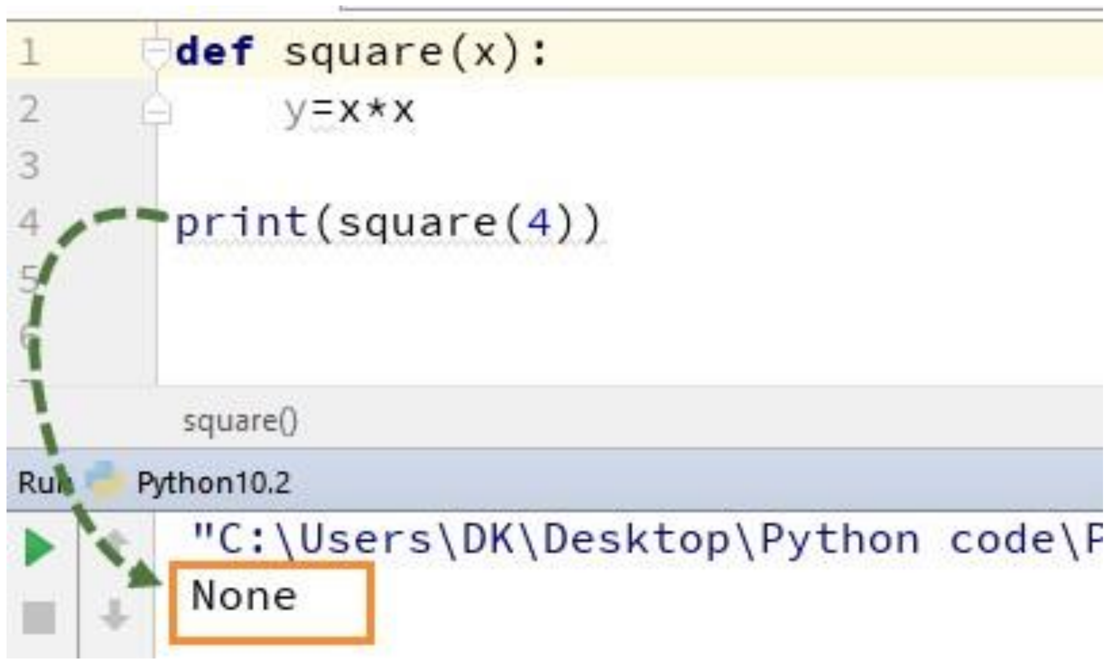


```
#define return function
def square(x):
    print(x*x)

print(square(4))
```

```
"C:\Users\DK\Desktop\Python
16
None
```

Step 2)

To make this clearer we replace the print command with assignment command. Let's check the output.



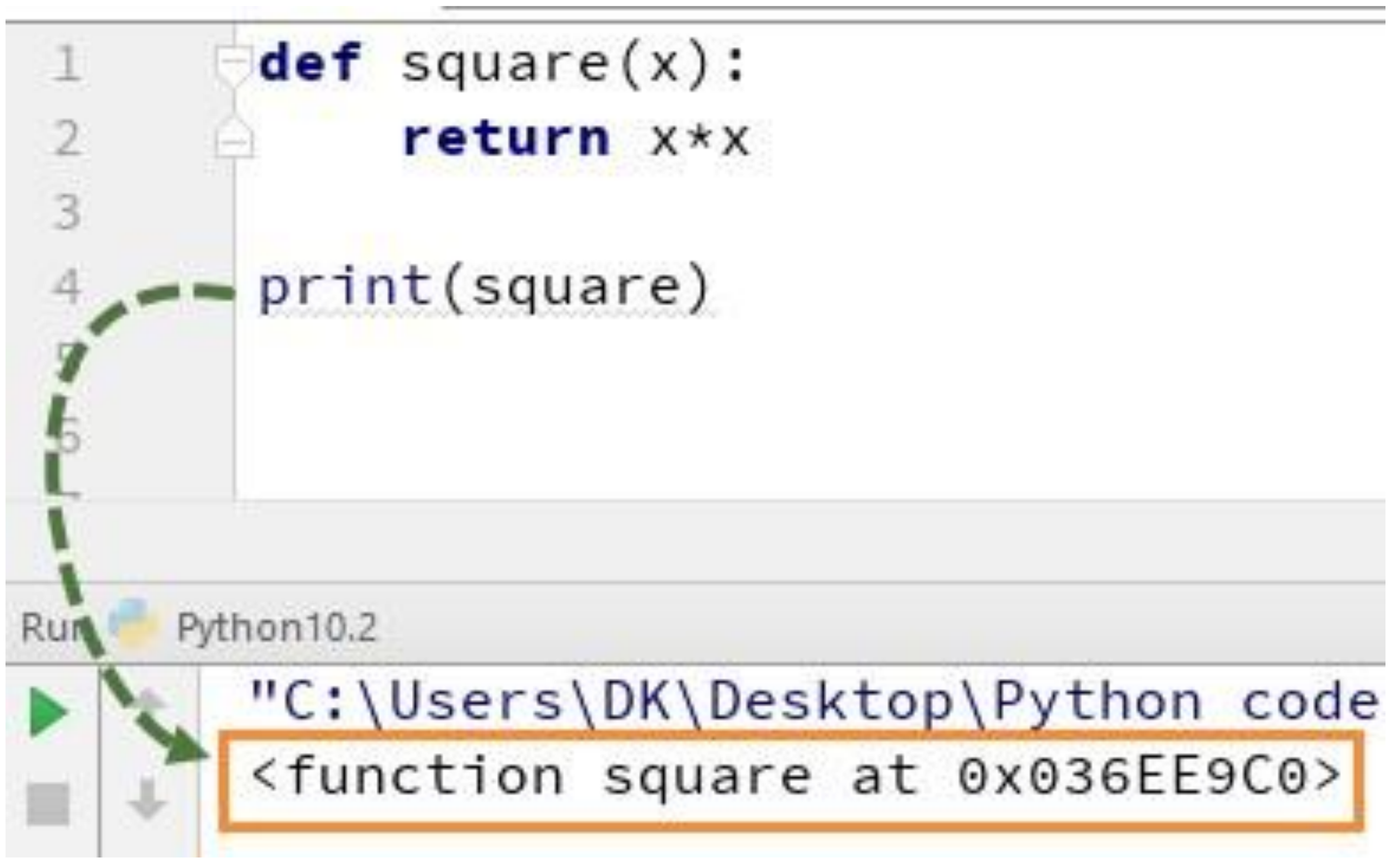
```
def square(x):
    y=x*x

print(square(4))
```

```
"C:\Users\DK\Desktop\Python code\P
None
```

When you run the command "print square (4)" it actually returns the value of the object since we don't have any specific function to run over here it returns "None".

Step 4)

Functions in Python are themselves an object, and an object has some value. We will here see how Python treats an object. When you run the command "print square" it returns the value of the object. Since we have not passed any argument, we don't have any specific function to run over here it returns a default value (0x021B2D30) which is the location of the object.



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

print(square)
```

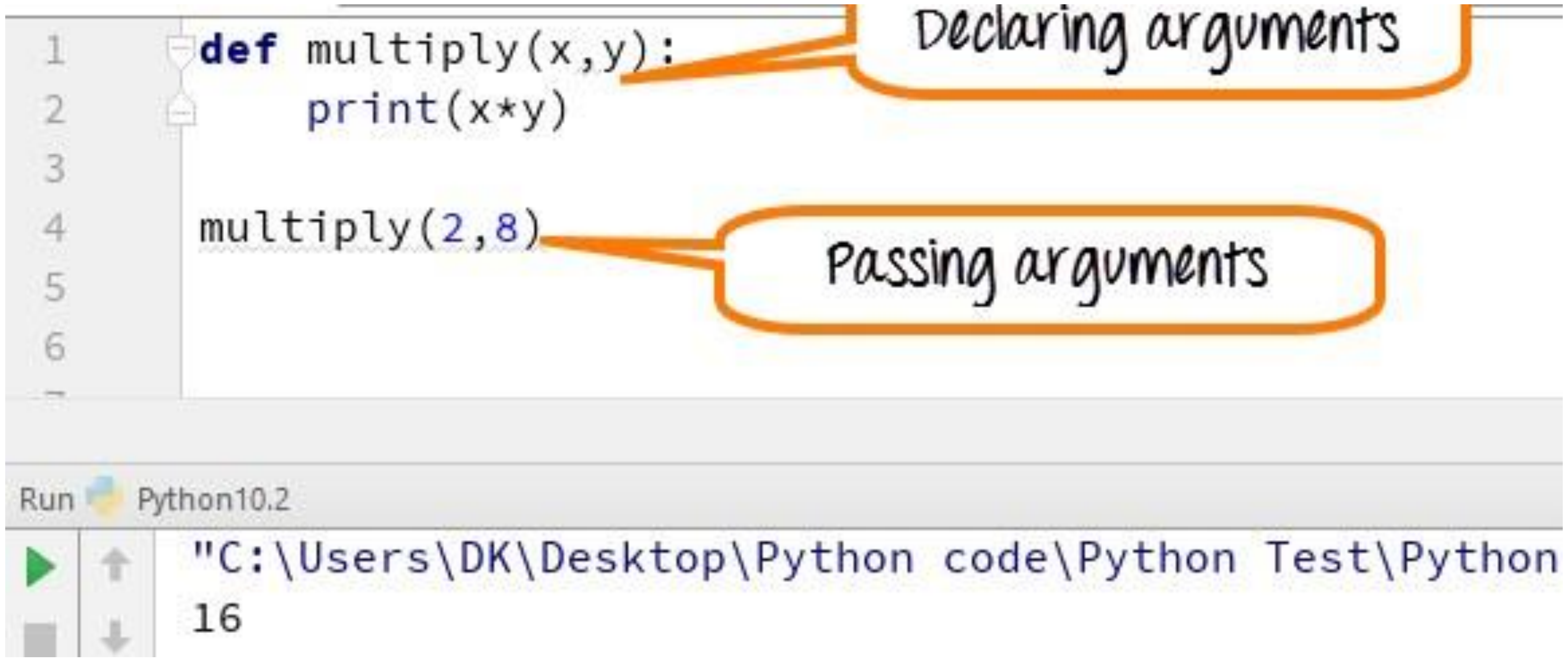
```
"C:\Users\DK\Desktop\Python code
<function square at 0x036EE9C0>
```

Arguments in Functions

The argument is a value that is passed to the function when it's called.
In other words on the calling side, it is an argument and on the function side it is a parameter.
Let see how Python Args works –

Step 1)

Arguments are declared in the function definition. While calling the function, you can pass the values for that args as shown below



```
def multiply(x,y):
    print(x*y)

multiply(2,8)
```

```
"C:\Users\DK\Desktop\Python code\Python Test\Python
16
```

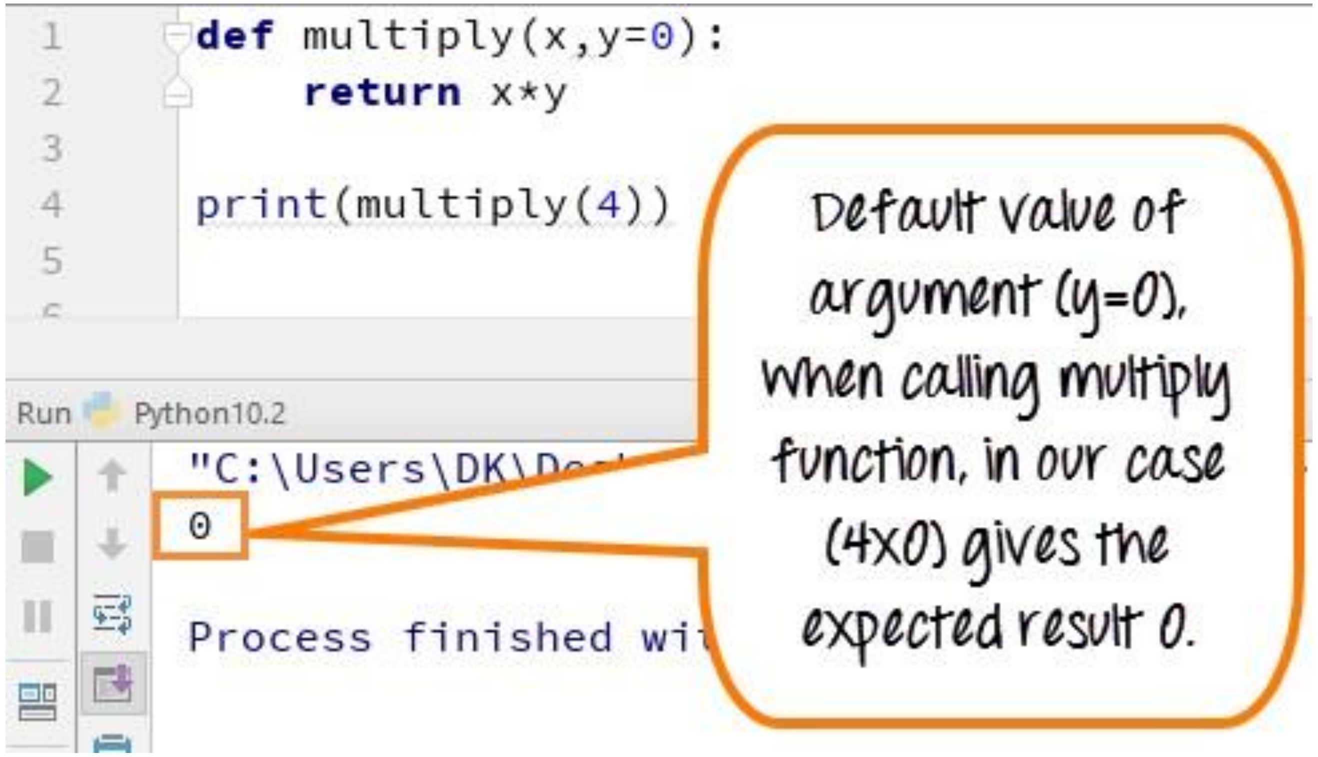
Step 2)

To declare a default value of an argument, assign it a value at function definition.

def multiply(x,y=0)

Example:

x has no default values. Default values of y=0. When we supply only one argument while calling multiply function, Python assigns the supplied value to x while keeping the value of y=0. Hence the multiply of x*y=0



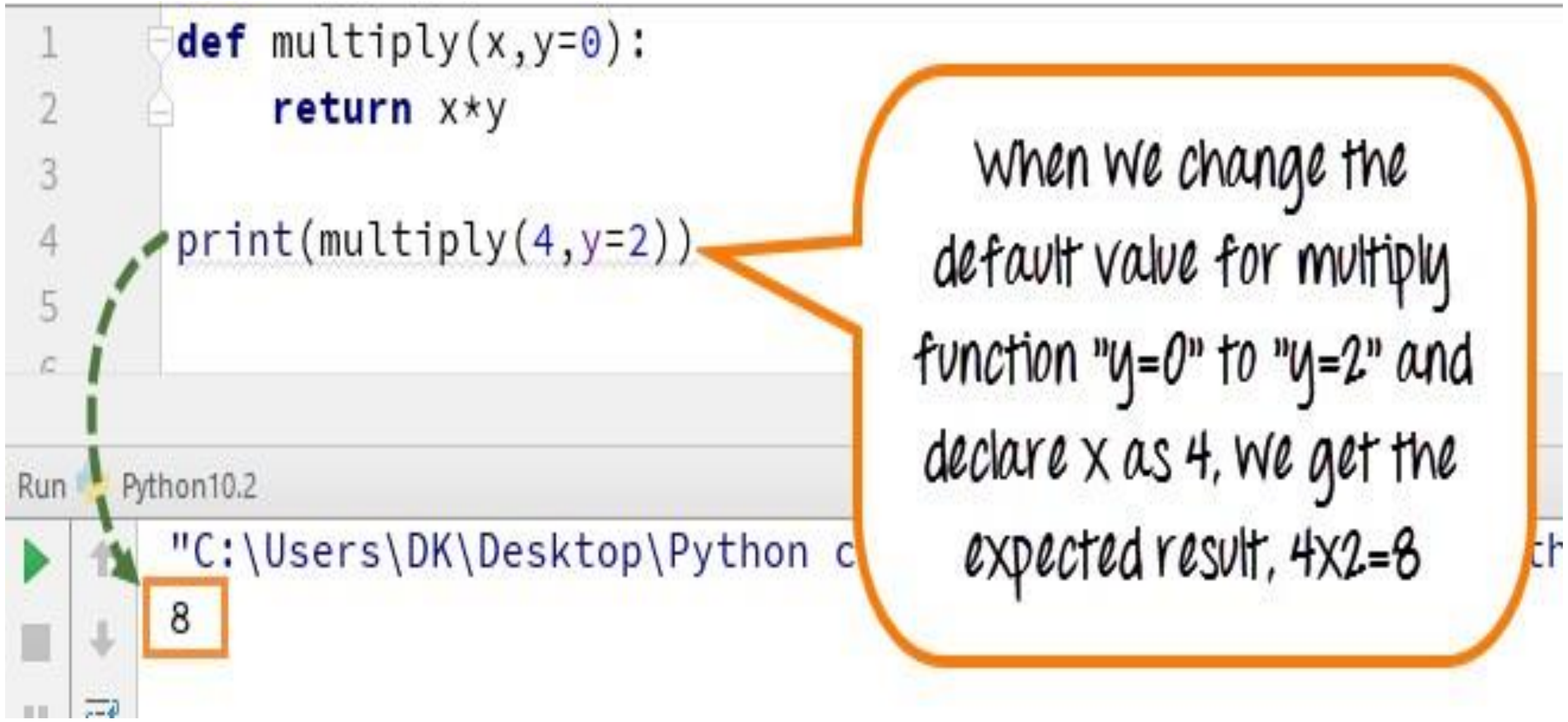
```
def multiply(x,y=0):
    return x*y

print(multiply(4))
```

```
"C:\Users\DK\De
0

Process finished wi
```

Run
Python10.2
Default value of argument (y=0). When calling multiply function, in our case (4x0) gives the expected result 0.

Step 3)

This time we will change the value to y=2 instead of the default value y=0, and it will return the output as (4x2)=8.

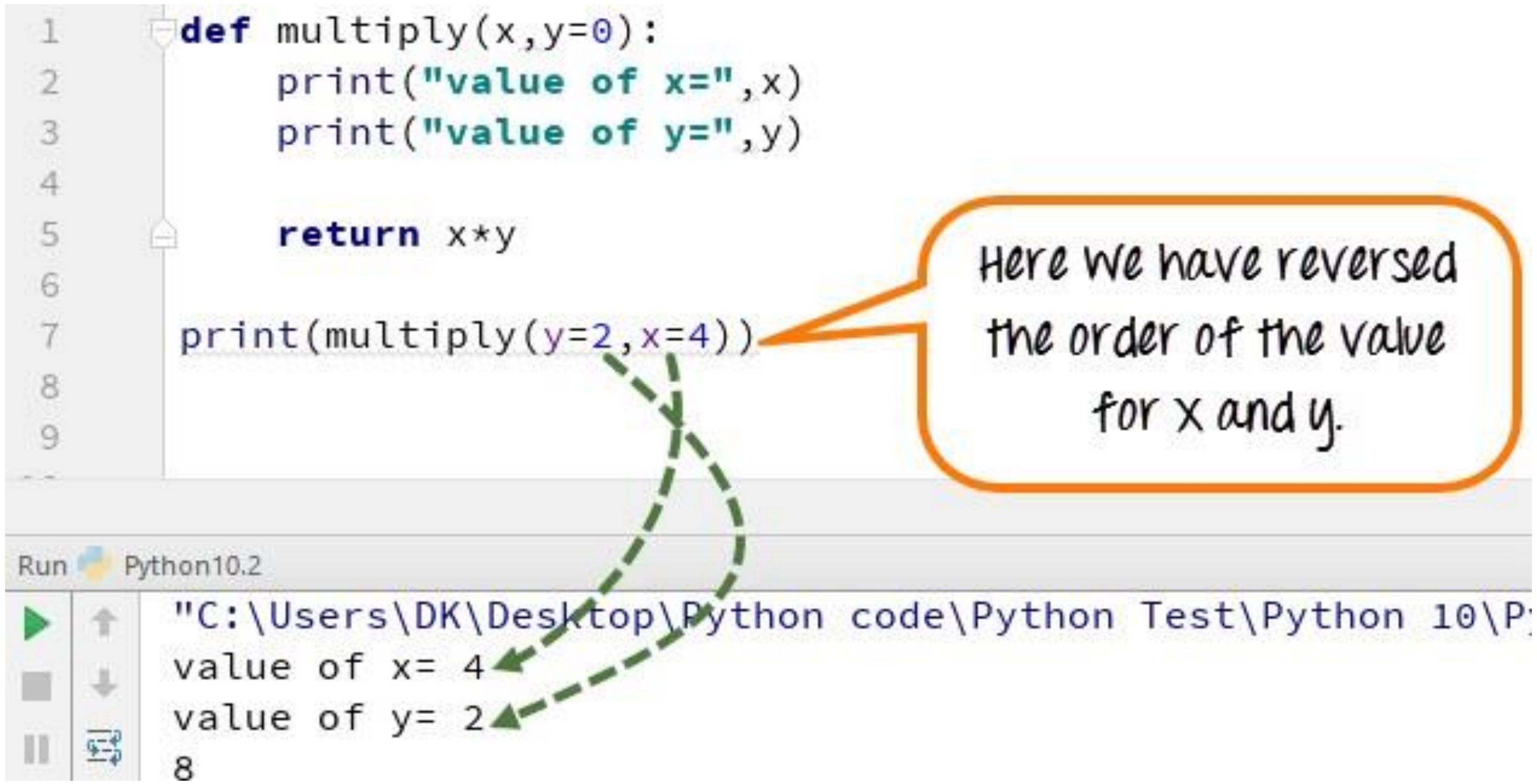


```
def multiply(x,y=0):
    return x*y

print(multiply(4,y=2))
```

```
Run   Python10.2
"C:\Users\DK\Desktop\Python c
8
```

When we change the default value for multiply function "y=0" to "y=2" and declare x as 4, we get the expected result, 4X2=8

Step 4)
You can also change the order in which the arguments can be passed in Python. Here we have reversed the order of the value x and y to x=4 and y=2.



```
def multiply(x,y=0):
    print("value of x=",x)
    print("value of y=",y)

    return x*y

print(multiply(y=2,x=4))
```

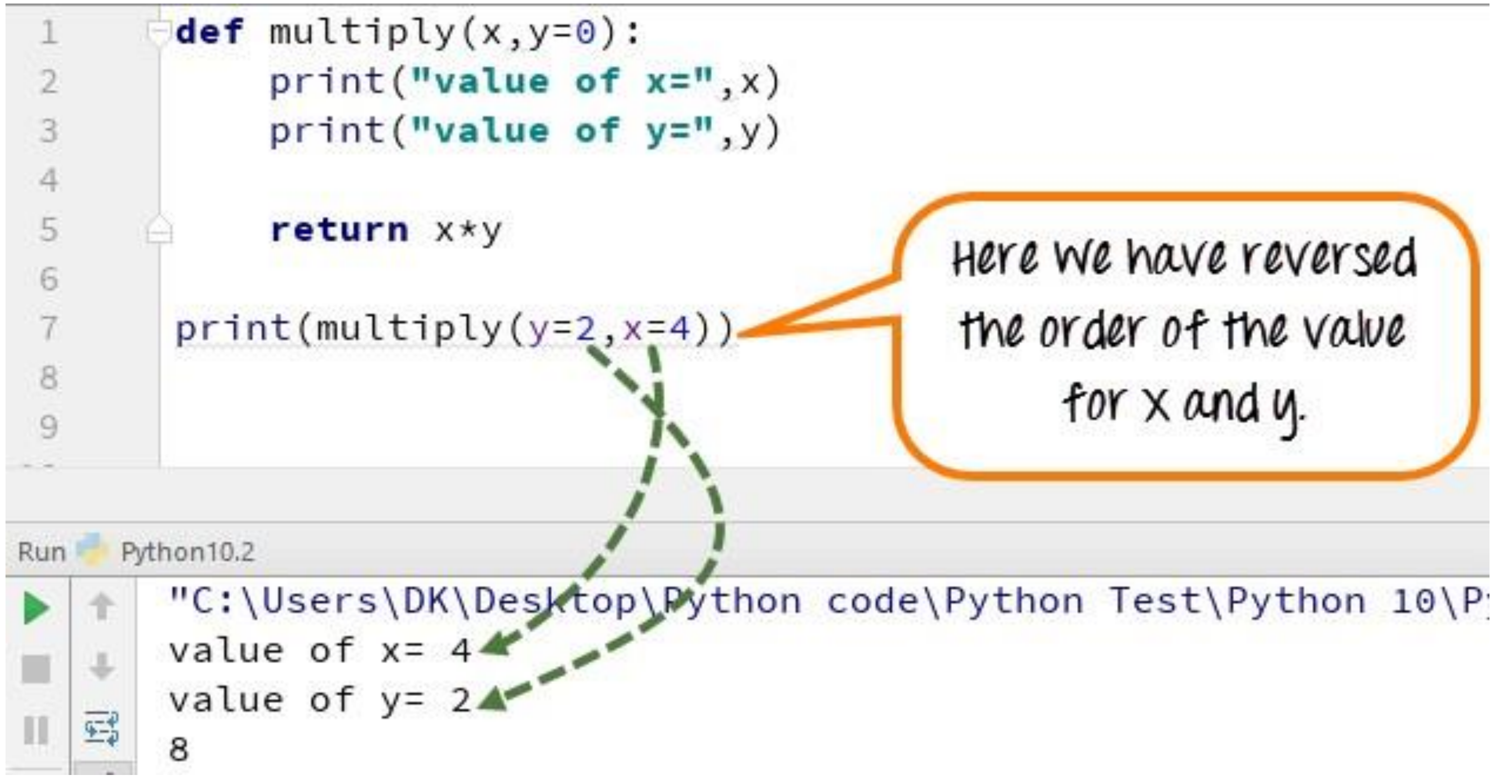
```
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\P
value of x= 4
value of y= 2
8
```

Step 5)

Multiple Arguments can also be passed as an array. Here in the example we call the multiple args (1,2,3,4,5) by calling the (*args) function.

Example:

We declared multiple args as number (1,2,3,4,5) when we call the (*args) function; it prints out the output as (1,2,3,4,5)



```
def multiply(x,y=0):
    print("value of x=",x)
    print("value of y=",y)

    return x*y

print(multiply(y=2,x=4))
```

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\P
value of x= 4
value of y= 2
8
```

Summary

- Function defined by the **def** statement
- The code block within every function starts with a colon (:) and should be indented (space)
- Any arguments or input parameters should be placed within these parentheses, etc.
- At least one indent should be left before the code after declaring function
- Same indent style should be maintained throughout the code within def function
- For best practices three or four indents are considered best before the statement
- You can use the "return" command to return values to the function call.
- Python will print a random value like (0x021B2D30) when the argument is not supplied to the calling function. Example "print function."
- On the calling side, it is an argument and on the function side it is a parameter
- Default value in argument - When we supply only one argument while calling multiply function or any other function, Python assigns the other argument by default
- Python enables you to reverse the order of the argument as well