# Functions

- 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 gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*
- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

```
def my_function():
  print("Hello from a function")


my_function()
```

There are mainly two types of functions.

- **User-define functions** - The user-defined functions are those define by the **user** to perform the specific task.
- **Built-in functions** - The built-in functions are those functions that are **pre-defined** in Python.

## Syntax of Function

```
def function_name(function_parameters):
        function_body # Set of Python statements
    return # optional return statement
```

Above shown is a function definition that consists of the following components.

- Keyword def that marks the start of the function header.
- A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of the function header.
- One or more valid python statements that make up the function body. An optional return statement to return a value from the function.

**Calling the function:**

```
# when function doesn't return anything
function_name(parameters)
```

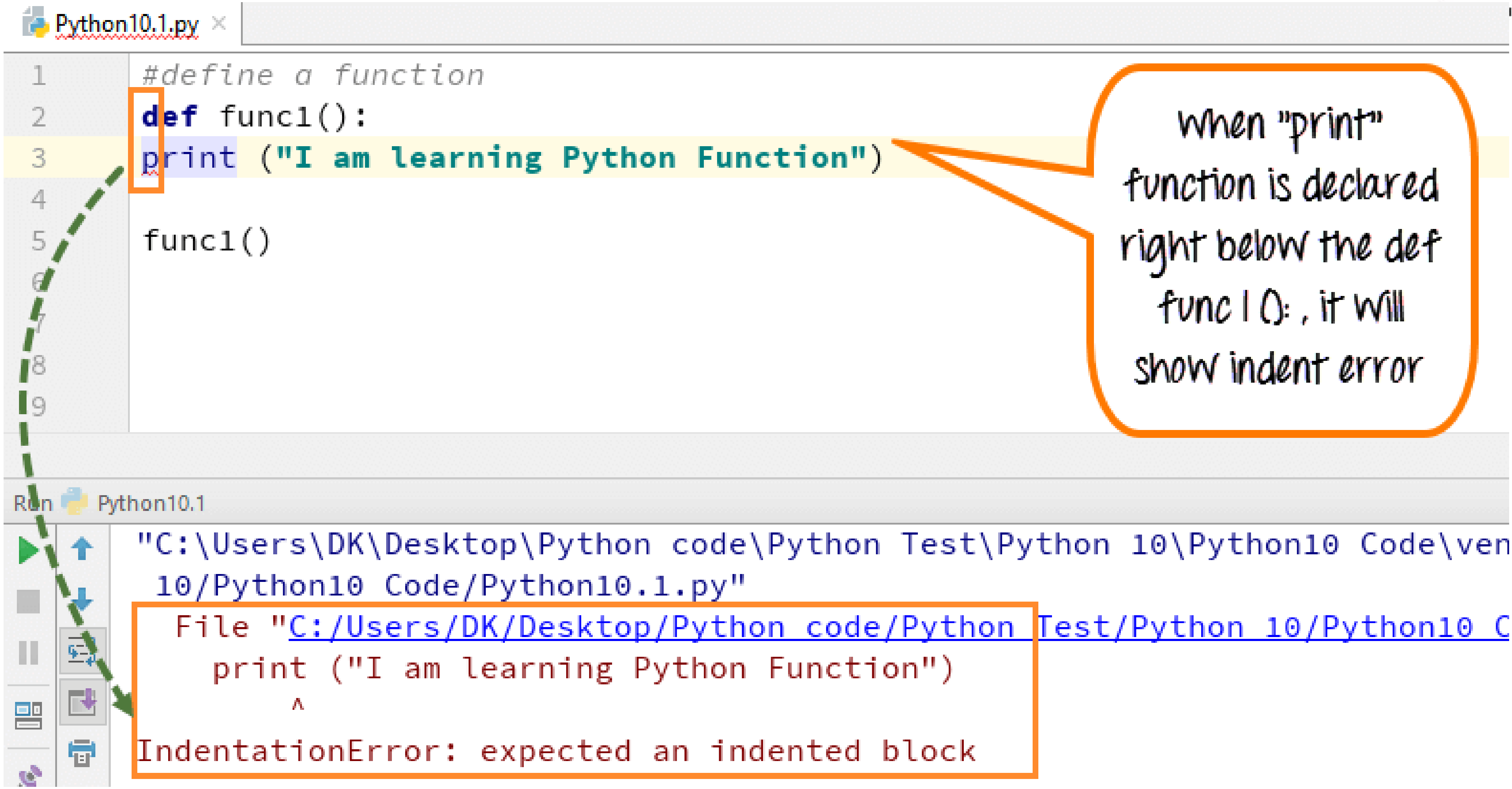
OR

```
# when function returns something
# variable is to store the returned value
variable = function_name(parameters)
```

# Significance of Indentation (Space)

Python follows a particular style of indentation to define the code, since Python functions don't have any explicit begin or end like curly braces to indicate the start and stop for the function, they have to rely on this indentation.

Here we take a simple example with "print" command. When we write "print" function right below the def func 1 (): It will show an "**indentation error: expected an indented block**".



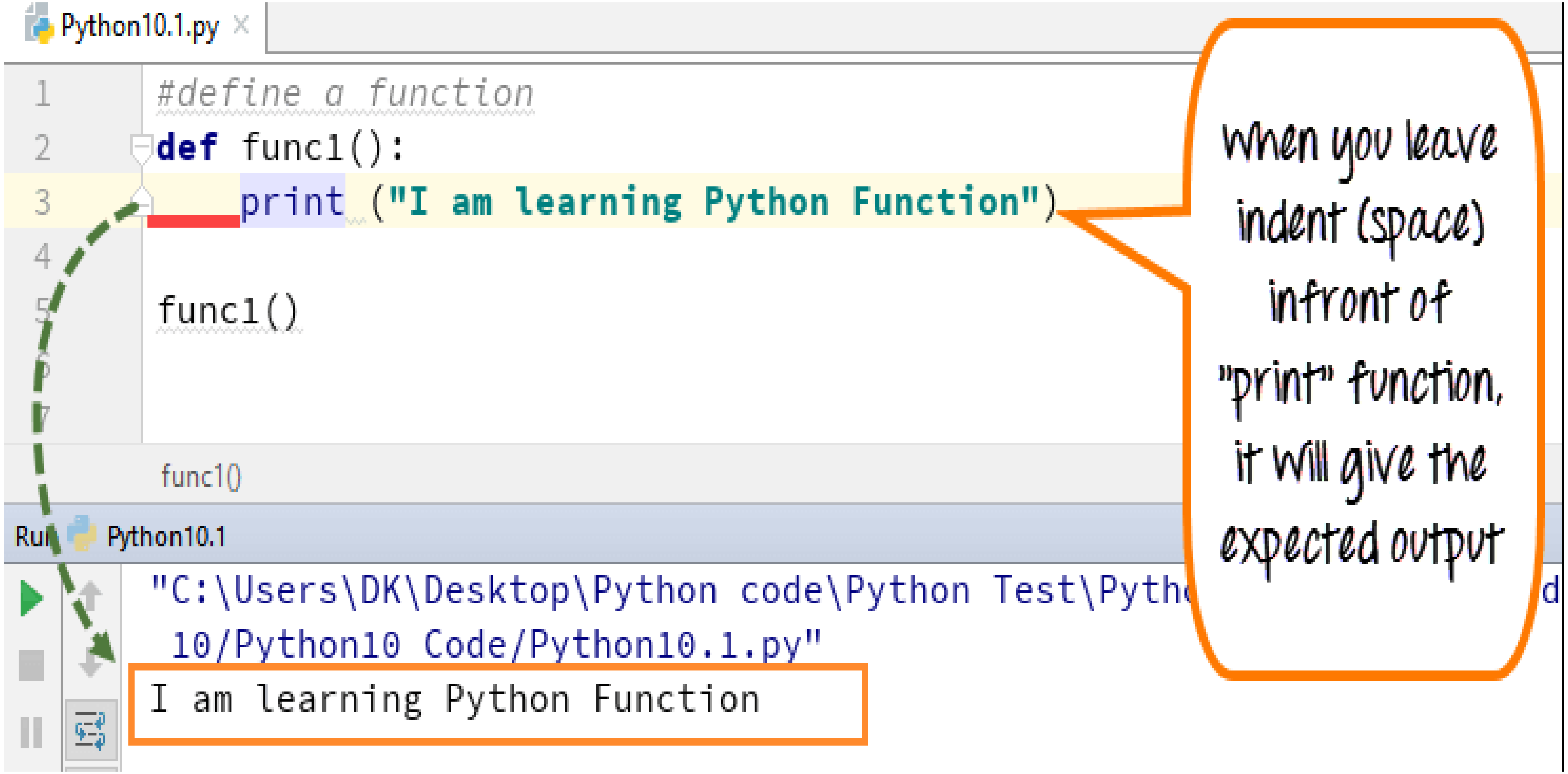
```
#define a function
def func1():
print ("I am learning Python Function")

func1()
```

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

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

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()
```

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

When you leave indent (space) infront of "print" function, it will give the expected output

## Example:

```
def greet():
    """This function displays 'Hello World!'"""“        #docstring
    print('Hello World!')

greet()
```

Output

Hello World!

By default, all the functions return None if the return statement does not exist.

Example: Calling User-defined Function

```
val = greet()
print(val)
```

Output

```
None
```

# Parameters or Arguments: information that are passed into a function

```
def my_function(fname):
  print(fname + " Ref")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])
my_function("Emil", "Tobias", "Linus")
```

- Keyword Arguments

You can also send arguments with the *key* = *value* syntax.

This way the order of the arguments does not matter.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = “a", child2 = “b", child3 = “c")
```

- Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a dictionary of arguments,

If the number of keyword arguments is unknown, add a double ** before the parameter name:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])
my_function(fname = "T", lname = "R")
```

# Default Parameter Value

- If we call the function without argument, it uses the default value:
- Example

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

- Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

```
def my_function(food):
  for x in food:
    print(x)
  print(type(food))


fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

## Return Values

To let a function return a value, use the return statement:

```
def my_function(x):
  return 5 * x

print(my_function(3))

n=my_function(3)
print(my_function(5))
print(my_function(9))
```

## The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

```
def myfunction():
  pass
```

**Example:**

Here we have a function add() that adds two numbers passed to it as parameters. Later after function declaration we are calling the function twice in our program to perform the addition.

```
def add(num1, num2):
    return num1 + num2

sum1 = add(100, 200)
sum2 = add(8, 9)
print(sum1)
print(sum2)
```

Output:

```
300
17
```

## Recursion

A function is said to be a recursive if it calls itself. For example, lets say we have a function abc() and in the body of abc() there is a call to the abc().

```
# Example of recursion in Python to
# find the factorial of a given number
def factorial(num):
    if num == 1:
       return 1
   else:
       return (num * factorial(num - 1))
num = 5
print("Factorial of", num, "is: ", factorial(num))
```

Output:

```
Factorial of 5 is:  120
```

- Generate squares of all the integers from 1 to 50.

- Count the number of characters in a string using a loop.