Python Lecture 8 – Functions

What are functions?

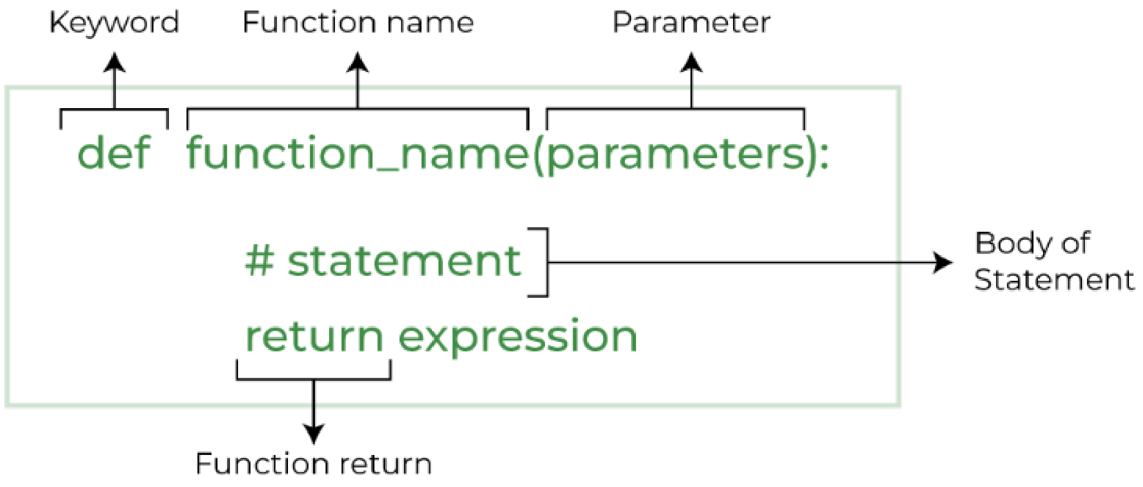
Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions

- Increase Code Readability
- Increase Code Reusability

Important Terms

1. **Syntax:**



   - Note: The function body is indented.
2. **Function Definition:**
   - A function is defined using the **def** keyword, followed by the **function name** and a pair of parentheses which optionally contains parameters.
3. **Parameters:**
   - Parameters are placeholders in a function definition. They allow you to pass values into the function when calling it.
   - A function may or may not have parameters. It can also have multiple parameters.
4. **Statement:**
   - Block of code that runs inside the function. (This is the indented part)
5. **Return (optional):**
   - This returns value from a function. Sometimes we just don't need to return a value so we don't use it at all.
   - The return statement also denotes that the function has ended. Any code after return is not executed.
6. **Arguments:**
   - Arguments are the actual values that you pass to a function when calling it.

7. **Calling a function:**
   - To execute a function, you need to call it. This involves using the function name followed by parentheses. In the parenthesis we also optionally include any arguments that the function takes.
8. **Examples:**
   - Basic Function
```
def greet():
   print("Hello, World!")
#Calling it
greet()
```
   - Basic Function with parameter
```
def greet_with_name(name):
   print("Hello, " + name + "!")
# Calling it
greet_with_name(“Wasiq”)
```
   - Function that returns a value
```
def Even_Odd(number)
   if number%2 == 0:
      type = “Even”
   else:
      type = “Odd”
   return type
# Calling it
Even_Odd(345)
```
   - Function with multiple parameters
```
def add_numbers(a, b):
   result = a + b
   return result
# Function call
sum_result = add_numbers(5, 3)
print("Sum:", sum_result)
```
9. **Parameter Vs Arguments**
   - A parameter is the variable listed inside the parentheses in the function definition.
   - An argument is the value that is sent to the function when it is called.
10. **Named Arguments:**
   - Also known as Keyword Arguments.
   - We can also call the function by mentioning the argument name as the key = value syntax.
   - This way the order of the arguments does not matter.
   - Example:
```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)
#Calling it
my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

NOTE:

- By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.
- If you try to call the function with 1 or 3 arguments, you will get an error.
- We can also set the 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()
    - my_function("Sweden")

Lambda Functions: / Anonymous Functions

In Python, an anonymous function means that a function is without a name. As we already know the **def** keyword is used to define the normal functions and the **lambda** keyword is used to create anonymous functions.

Syntax:

`lambda` ***`arguments`*** `:` ***`expression`*** OR

variable_name = `lambda` ***`arguments`*** `:` ***`expression`***

- A lambda function is a small anonymous function.
- The power of lambda is better shown when you use them as an anonymous function inside another function.
- Use lambda functions when an anonymous function is required for a short period of time.
- A lambda function can take any number of arguments, but can only have one expression.
- The expression is executed and the result is returned.
- Lambda functions are inline functions.

Example: Regular Function Vs Lambda Function

Regular Function:

```
def add(x, y):
   return x + y
```

Lambda Function:

```
add_lambda = lambda x, y: x + y
```

NOTE:

We can use it for single or many arguments but expression will be same.

Example: Lambda Function with Single Argument

```
X = lambda a: a^2

# This will take value of “a” and return its square.

print(X(5))
```

Example: Lambda Function with Two Arguments:

```
Divide = lambda a,b : a/b

# Usage

print(Divide(10,4))
```

Example: Lambda Function that returns a Boolean:

```
is_even = lambda x: x % 2 == 0

# Usage

result1 = is_even(4)
```

Example: Lambda function to find the largest of two numbers

```
find_largest = lambda x, y: max(x, y)

# Here we have used another function inside lambda function.

result = find_largest(10, 7)
```

Example: Realistic Usage of Lambda Function:

Lambda functions are commonly used in functional programming constructs like map, filter, and reduce.

```
numbers = [1, 2, 3, 4, 5]

squared = list(map(lambda x: x**2, numbers))
```

Explanation:

- numbers = [1, 2, 3, 4, 5] defines a list named numbers containing five integer values.
- lambda x: x**2 defines a lambda function that takes a single argument x and returns its square (x**2).
- map(function, iterable) applies the specified function to all the items in the given iterable (in this case, the list numbers).
- The lambda function lambda x: x**2 is applied to each element of the numbers list.
- The result of the map operation is an iterable (a map object).
- The list() function is then used to convert this iterable into a list.
- squared is now a list containing the squared values of each element in the numbers list.