Python Functions

In this class, you'll learn about functions, what a function is, the syntax, components, and types of functions. Also, you'll learn to create a function in Python.

What is a function in Python? ¶

In Python, a **function is a block of organized, reusable (DRY- Don't Repeat Yourself) code with a name** that is used to perform a single, specific task. It can take arguments and returns the value.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it improves efficiency and reduces errors because of the reusability of a code.

Types of Functions

Python support two types of functions

1. Built-in (https://github.com/milaan9/04_Python_Functions/tree/main/002_Python_Functions_Built_in) function
2. User-defined (https://github.com/milaan9/04_Python_Functions/blob/main/Python_User_defined_Functions.ipynb) function

1.**Built-in function**

The functions which are come along with Python itself are called a built-in function or predefined function. Some of them are: `range()` , `print()` , `input()` , `type()` , `id()` , `eval()` etc.

Example: Python `range()` function generates the immutable sequence of numbers starting from the given start integer to the stop integer.

```
>>> for i in range(1, 10):
>>>     print(i, end=' ')

1 2 3 4 5 6 7 8 9
```

1. **User-defined function**

Functions which are created by programmer explicitly according to the requirement are called a user-defined function.

Syntax:

```
def function_name(parameter1, parameter2):
    """docstring"""
    # function body
    # write some action
return value
```

Defining a Function

1. `def` is a keyword that marks the start of the function header.
2. `function_name` to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python (https://github.com/milaan9/01_Python_Introduction/blob/main/005_Python_Keywords_and_Identifiers.ipynb).
3. `parameter` is the value passed to the function. They are optional.
4. `:` (colon) to mark the end of the function header.
5. `function body` is a block of code that performs some task and all the statements in `function body` must have the same **indentation** level (usually 4 spaces).
6. **"""docstring"""** documentation string is used to describe what the function does.
7. `return` is a keyword to return a value from the function.. A return statement with no arguments is the same as return `None` .

> **Note:** While defining a function, we use two keywords, `def` (mandatory) and `return` (optional).

Example:

```
>>> def add(num1,num2):           # Function name: 'add', Parameters: 'num1', 'num2'
>>>     print("Number 1: ", num1) #  Function body
>>>     print("Number 2: ", num2) #  Function body
>>>     addition = num1 + num2    #  Function body
>>>     return addition           # return value


>>> res = add(2, 4)    # Function call
>>> print("Result: ", res)
```

Defining a function without any parameters

Function can be declared without parameters.

In [1]:
```
# Example 1:

def greet():
    print("Welcome to Python for Data Science")

# call function using its name
greet()
```

```
Welcome to Python for Data Science
```

In [2]:
```
# Example 2:

def add_two_numbers ():
    num_one = 3
    num_two = 6
    total = num_one + num_two
    print(total)
add_two_numbers() # calling a function
```

```
9
```

In [3]:
```python
# Example 3:

def generate_full_name ():
    first_name = 'Milaan'
    last_name = 'Parmar'
    space = ' '
    full_name = first_name + space + last_name
    print(full_name)
generate_full_name () # calling a function
```

```
Milaan Parmar
```

Defining a function without parameters and `return` value

Function can also return values, if a function does not have a `return` statement, the value of the function is None. Let us rewrite the above functions using `return`. From now on, we get a value from a function when we call the function and print it.

In [4]:
```python
# Example 1:

def add_two_numbers ():
    num_one = 3
    num_two = 6
    total = num_one + num_two
    return total
print(add_two_numbers())
```

```
9
```

In [5]:
```python
# Example 2:

def generate_full_name ():
    first_name = 'Milaan'
    last_name = 'Parmar'
    space = ' '
    full_name = first_name + space + last_name
    return full_name
print(generate_full_name())
```

```
Milaan Parmar
```

Defining a function with parameters

In a function we can pass different data types(number, string, boolean, list, tuple, dictionary or set) as a parameter.

Single Parameter:

If our function takes a parameter we should call our function with an argument

In [6]:
```python
# Example 1: Gereeting

def greet(name):
    """
    This function greets to the person passed in as a parameter
    """
    print("Hello, " + name + ". Good morning!")    # No output!
```

In [7]:
```
# Example 2:

def sum_of_numbers(n):
    total = 0
    for i in range(n+1):
        total+=i
    print(total)
print(sum_of_numbers(10))  # 55
print(sum_of_numbers(100)) # 5050
```
```
55
None
5050
None
```

Two Parameter:

A function may or may not have a parameter or parameters. A function may also have two or more parameters. If our function takes parameters we should call it with arguments.

In [8]:
```
# Example 1:

def course(name, course_name):
    print("Hello", name, "Welcome to Python for Data Science")
    print("Your course name is", course_name)

course('Arthur', 'Python')   # call function
```
```
Hello Arthur Welcome to Python for Data Science
Your course name is Python
```

Defining a function with parameters and `return` value

In [9]:
```
# Example 1:

def greetings (name):  # single parameter
    message = name + ', welcome to Python for Data Science'
    return message

print(greetings('Milaan'))
```
```
Milaan, welcome to Python for Data Science
```

In [10]:
```
# Example 2:

def add_ten(num):  # single parameter
    ten = 10
    return num + ten
print(add_ten(90))
```
```
100
```

In [11]:
```
# Example 3:

def square_number(x):  # single parameter
    return x * x
print(square_number(3))
```
```
9
```

In [12]:
```
# Example 4:

def area_of_circle (r):  # single parameter
    PI = 3.14
    area = PI * r ** 2
    return area
print(area_of_circle(10))
```
```
314.0
```

In [13]:
```
# Example 5:

def calculator(a, b):  # two parameter
    add = a + b
    return add    # return the addition

result = calculator(30, 6)   # call function & take return value in variable
print("Addition :", result)   # Output Addition : 36
```
```
Addition : 36
```

In [14]:
```
# Example 6:

def generate_full_name (first_name, last_name):  # two parameter
    space = ' '
    full_name = first_name + space + last_name
    return full_name
print('Full Name: ', generate_full_name('Milaan','Parmar'))
```
```
Full Name:  Milaan Parmar
```

In [15]:
```
# Example 7:

def sum_two_numbers (num_one, num_two):  # two parameter
    sum = num_one + num_two
    return sum
print('Sum of two numbers: ', sum_two_numbers(1, 9))
```
```
Sum of two numbers:  10
```

In [16]:
```
# Example 8:

def calculate_age (current_year, birth_year):  # two parameter
    age = current_year - birth_year
    return age;

print('Age: ', calculate_age(2021, 1819))
```
```
Age:  202
```

In [17]:
```
# Example 9:

def weight_of_object (mass, gravity):  # two parameter
    weight = str(mass * gravity)+ ' N' # the value has to be changed to a string first
    return weight
print('Weight of an object in Newtons: ', weight_of_object(100, 9.81))
```
```
Weight of an object in Newtons:  981.0 N
```

Function `return` Statement

In Python, to return value from the function, a **`return`** statement is used. It returns the value of the expression following the returns keyword.

Syntax:

```
def fun():
    statement-1
    statement-2
    statement-3
    .
    .
    return [expression]
```

The **`return`** value is nothing but a outcome of function.

- The **`return`** statement ends the function execution.
- For a function, it is not mandatory to return a value.
- If a **`return`** statement is used without any expression, then the **`None`** is returned.
- The **`return`** statement should be inside of the function block.

Return Single Value

In [18]:
```
print(greet("Cory"))
```

```
Hello, Cory. Good morning!
None
```

Here, **`None`** is the returned value since **`greet()`** directly prints the name and no **`return`** statement is used.

Passing Arguments with Key and Value

If we pass the arguments with key and value, the order of the arguments does not matter.

In [19]:
```
# Example 1:

def print_fullname(firstname, lastname):
    space = ' '
    full_name = firstname  + space + lastname
    print(full_name)
print(print_fullname(firstname = 'Milaan', lastname = 'Parmar'))
```

```
Milaan Parmar
None
```

In [20]:
```
# Example 2:

def add_two_numbers (num1, num2):
    total = num1 + num2
    print(total)
print(add_two_numbers(num2 = 3, num1 = 2)) # Order does not matter
```

```
5
None
```

If we do not **`return`** a value with a function, then our function is returning **`None`** by default. To return a value with a function we use the keyword **`return`** followed by the variable we are returning. We can return any kind of data types from a function.

In [21]:
```
# Example 1:  with return statement

def print_fullname(firstname, lastname):
    space = ' '
    full_name = firstname  + space + lastname
    return full_name
print(print_fullname(firstname = 'Milaan', lastname = 'Parmar'))
```

```
Milaan Parmar
```

In [22]:
```
# Example 2:  with return statement

def add_two_numbers (num1, num2):
    total = num1 + num2
    return total
print(add_two_numbers(num2 = 3, num1 = 2)) # Order does not matter
```

```
5
```

In [23]:
```
# Example 3:

def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""

    if num >= 0:
        return num
    else:
        return -num

print(absolute_value(2))
print(absolute_value(-4))
```

```
2
4
```

In [24]:
```
# Example 4:

def sum(a,b):  # Function 1
    print("Adding the two values")
    print("Printing within Function")
    print(a+b)
    return a+b

def msg():  # Function 2
    print("Hello")
    return

total=sum(10,20)
print('total : ',total)
msg()
print("Rest of code")
```

```
Adding the two values
Printing within Function
30
total :  30
Hello
Rest of code
```

In [25]:
```
# Example 5:

def is_even(list1):
    even_num = []
    for n in list1:
        if n % 2 == 0:
            even_num.append(n)
    # return a list
    return even_num

# Pass list to the function
even_num = is_even([2, 3, 46, 63, 72, 83, 90, 19])
print("Even numbers are:", even_num)
```

```
Even numbers are: [2, 46, 72, 90]
```

Return Multiple Values

You can also return multiple values from a function. Use the return statement by separating each expression by a comma.

In [26]:
```
# Example 1:

def arithmetic(num1, num2):
    add = num1 + num2
    sub = num1 - num2
    multiply = num1 * num2
    division = num1 / num2
    # return four values
    return add, sub, multiply, division

a, b, c, d = arithmetic(10, 2)  # read four return values in four variables

print("Addition: ", a)
print("Subtraction: ", b)
print("Multiplication: ", c)
print("Division: ", d)
```

```
Addition:  12
Subtraction:  8
Multiplication:  20
Division:  5.0
```

Return Boolean Values

In [27]:
```
# Example 1:

def is_even (n):
    if n % 2 == 0:
        print('even')
        return True    # return stops further execution of the function, similar to break
    return False
print(is_even(10)) # True
print(is_even(7)) # False
```

```
even
True
False
```

Return a List

In [28]:

```
# Example 1:

def find_even_numbers(n):
    evens = []
    for i in range(n + 1):
        if i % 2 == 0:
            evens.append(i)
    return evens
print(find_even_numbers(10))
```

```
[0, 2, 4, 6, 8, 10]
```

How to call a function in python?

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

In [29]:

```
greet('Alan')
```

```
Hello, Alan. Good morning!
```

Note: Try running the above code in the Python program with the function definition to see the output.

In [30]:

```
# Example 1:

def wish(name):
    """
    This function wishes to the person passed in as a parameter
    """
    print("Happy birthday, " + name + ". Hope you have a wonderful day!")

wish('Bill')
```

```
Happy birthday, Bill. Hope you have a wonderful day!
```

In [31]:

```
# Example 2:

def greetings (name = 'Clark'):
    message = name + ', welcome to Python for Data Science'
    return message
print(greetings())
print(greetings('Milaan'))
```

```
Clark, welcome to Python for Data Science
Milaan, welcome to Python for Data Science
```

In [32]:
```python
# Example 3:

def generate_full_name (first_name = 'Milaan', last_name = 'Parmar'):
    space = ' '
    full_name = first_name + space + last_name
    return full_name
print(generate_full_name())
print(generate_full_name('Ethan','Hunt'))
```

```
Milaan Parmar
Ethan Hunt
```

In [33]:
```python
# Example 4:

def calculate_age (birth_year,current_year = 2021):
    age = current_year - birth_year
    return age;
print('Age: ', calculate_age(1821))
```

```
Age:  200
```

In [34]:
```python
# Example 5:

def swap(x, y):
    """
    This function swaps the value of two variables
    """
    temp = x;  # value of x will go inside temp
    x = y;     # value of y will go inside x
    y = temp;  # value of temp will go inside y
    print("value of x is:", x)
    print("value of y is:", y)
    return     # "return" is optional

x = 6
y = 9
swap(x, y)     #call function
```

```
value of x is: 9
value of y is: 6
```

In [35]:
```python
# Example 6:

def even_odd(n):
    if n % 2 == 0:   # check number is even or odd
        print(n, 'is a Even number')
    else:
        print(n, 'is a Odd Number')

even_odd(9)   # calling function by its name
```

```
9 is a Odd Number
```

In [36]:
```python
# Example 7:

def weight_of_object (mass, gravity = 9.81):
    weight = str(mass * gravity)+ ' N' # the value has to be changed to string first
    return weight
print('Weight of an object in Newtons: ', weight_of_object(100)) # 9.81 - average gravity on
Earth's surface
print('Weight of an object in Newtons: ', weight_of_object(100, 1.62)) # gravity on the surfa
ce of the Moon
```

```
Weight of an object in Newtons:  981.0 N
Weight of an object in Newtons:  162.0 N
```

Docstrings

The first string after the function header is called the **docstring** and is short for documentation string. It is a descriptive text (like a comment) written by a programmer to let others know what block of code does.

Although **optional**, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.

It is being declared using triple single quotes `''' '''` or triple-double quote `""" """` so that docstring can extend up to multiple lines.

We can access docstring using doc attribute **`__doc__`** for any object like list, tuple, dict, and user-defined function, etc.

In the above example, we have a docstring immediately below the function header.

In [37]:
```
print(greet.__doc__)
```

```
    This function greets to the person passed in as a parameter
```

To learn more about docstrings in Python, visit **Python Docstrings (https://github.com/milaan9/04_Python_Functions/blob/main/Python_Docstrings.ipynb)**.

Function `pass` Statement

In Python, the `pass` is the keyword, which won't do anything. Sometimes there is a situation where we need to define a syntactically empty block. We can define that block using the `pass` keyword.

When the interpreter finds a `pass` statement in the program, it returns no operation.

In [38]:
```
# Example 1:

def addition(num1, num2):
    # Implementation of addition function in comming release
    # Pass statement
    pass

addition(10, 2)
```

💻 Exercises ➞ Functions

Exercises ➞ Level 1

1. Area of a circle is calculated as follows: **area = π x r x r** and **perimeter = 2 x π x r**. Write a function that calculates `area_of_circle` and `perimeter_of_circle` .
2. Write a function called `add_all_nums` which takes arbitrary number of arguments and sums all the arguments. Check if all the list items are number types. If not do give a reasonable feedback.
3. Temperature in **°C** can be converted to **°F** using this formula: **°F = (°C x 9/5) + 32**. Write a function which converts **°C to °F**, `convert_celsius_2_fahrenheit` .
4. Write a function called `check_season` , it takes a month parameter and returns the season: Autumn, Winter, Spring or Summer.
5. Write a function called `calculate_slope` which return the slope of a linear equation
6. Quadratic equation is calculated as follows: **ax^2 + bx + c = 0**. Write a function which calculates solution set of a quadratic equation, `solve_quadratic_eqn` .
7. Declare a function named `print_list` . It takes a list as a parameter and it prints out each element of the list.
8. Declare a function named `reverse_list` . It takes an array as a parameter and it returns the reverse of the array (use loops).

   -
     ```
     print(reverse_list([1, 2, 3, 4, 5]))
     #[5, 4, 3, 2, 1]
     print(reverse_list1(["A", "B", "C"]))
     #["C", "B", "A"]
     ```

9. Declare a function named `capitalize_list_items` . It takes a list as a parameter and it returns a capitalized list of items
10. Declare a function named `add_item` . It takes a list and an item parameters. It returns a list with the item added at the end.

    -
      ```
      food_staff = ['Potato', 'Tomato', 'Mango', 'Milk']
      print(add_item(food_staff, 'Fungi'))  #['Potato', 'Tomato', 'Mango', 'Milk', 'Fungi']
      numbers = [2, 3, 7, 9]
      print(add_item(numbers, 5))      #[2, 3, 7, 9, 5]
      ```

11. Declare a function named `remove_item` . It takes a list and an item parameters. It returns a list with the item removed from it.

    -
      ```
      food_staff = ['Potato', 'Tomato', 'Mango', 'Milk']
      print(remove_item(food_staff, 'Mango'))  # ['Potato', 'Tomato', 'Milk']
      numbers = [2, 3, 7, 9]
      print(remove_item(numbers, 3))  # [2, 7, 9]
      ```

12. Declare a function named `sum_of_numbers` . It takes a number parameter and it adds all the numbers in that range.

    -
      ```
      print(sum_of_numbers(5))  # 15
      print(sum_all_numbers(10)) # 55
      print(sum_all_numbers(100)) # 5050
      ```

13. Declare a function named `sum_of_odds` . It takes a number parameter and it adds all the odd numbers in that range.
14. Declare a function named `sum_of_even` . It takes a number parameter and it adds all the even numbers in that - range.

Exercises ➞ Level 2

1. Declare a function named `evens_and_odds` . It takes a positive integer as parameter and it counts number of evens and odds in the number.

   -
     ```
     print(evens_and_odds(100))
     #The number of odds are 50.
     #The number of evens are 51.
     ```

2. Call your function `factorial` , it takes a whole number as a parameter and it return a factorial of the number
3. Call your function `is_empty` , it takes a parameter and it checks if it is empty or not

4. Write different functions which take lists. They should `calculate_mean` , `calculate_median` , `calculate_mode` , `calculate_range` , `calculate_variance` , `calculate_std` (standard deviation).

Exercises ➞ Level 3

1. Write a function called `is_prime` , which checks if a number is prime.
2. Write a functions which checks if all items are unique in the list.
3. Write a function which checks if all the items of the list are of the same data type.
4. Write a function which check if provided variable is a valid python variable
5. Go to the data folder and access the countries-data.py (https://github.com/milaan9/03_Python_Flow_Control/blob/main/countries_details_data.py) file.

- Create a function called the `most_spoken_languages` in the world. It should return 10 or 20 most spoken languages in the world in descending order
- Create a function called the `most_populated_countries` . It should return 10 or 20 most populated countries in descending order.

```
In [ ]:
```