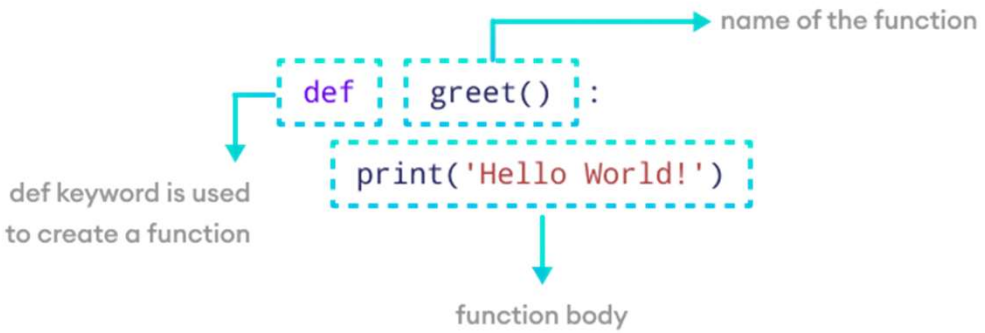
Basic Functions

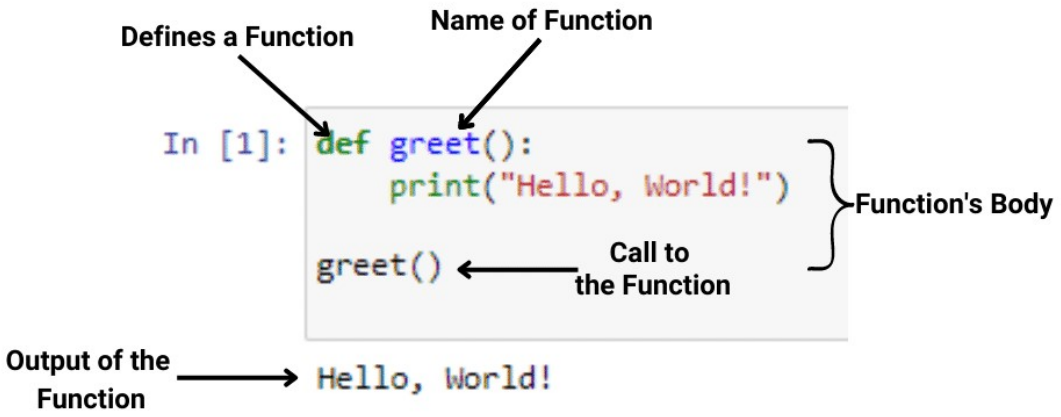
Part 1

What is a Function?



- Functions in Python are a fundamental concept that allow you to encapsulate code for reuse and organization.
- A function is defined using the `def` keyword, followed by the function name and parentheses. The code block within the function is indented.

Calling a Function



- To execute the function, you call it by its name followed by parentheses.

Arguments and Parameters

- Functions can take arguments, which are specified in the parentheses during function definition. These arguments are accessible within the function as variables.

```
# Pass single argument to a function
def hello(name):
    print('Hello,', name)
```

- When you call a function with arguments, the values of those arguments are copied to their corresponding parameters inside the function.

Multiple Arguments

```
# Pass two arguments
def func(name, job):
    print(name, 'is a', job)

func('Bob', 'developer')
# Prints Bob is a developer
```

- You can send as many arguments as you like, separated by commas `,`.

Types of Arguments

- Python handles function arguments in a very flexible manner, compared to other languages. It supports multiple types of arguments in the function definition. Here's the list:
  - Positional Arguments
  - Keyword Arguments
  - Default Arguments
  - Variable Length Positional Arguments (*args)
  - Variable Length Keyword Arguments (**kwargs)

Positional Arguments

- The most common are positional arguments, whose values are copied to their corresponding parameters in order.
- The only downside of positional arguments is that you need to pass arguments in the order in which they are defined.

```
def func(name, job):
    print(name, 'is a', job)

func('Bob', 'developer')
# Prints Bob is a developer
```

Keyword Arguments

- To avoid positional argument confusion, you can pass arguments using the names of their corresponding parameters.
- In this case, the order of the arguments no longer matters because arguments are matched by name, not by position.

```
# Keyword arguments can be put in any order
def func(name, job):
    print(name, 'is a', job)

func(name='Bob', job='developer')
# Prints Bob is a developer

func(job='developer', name='Bob')
# Prints Bob is a developer
```

Default Arguments

- You can specify default values for arguments when defining a function. The default value is used if the function is called without a corresponding argument.
- In short, defaults allow you to make selected arguments optional.

Variable Length Arguments (*args and **kwargs)

- Variable length arguments are useful when you want to create functions that take unlimited number of arguments. Unlimited in the sense that you do not know beforehand how many arguments can be passed to your function by the user.

*args

- When you prefix a parameter with an asterisk * , it collects all the unmatched positional arguments into a tuple. Because it is a normal tuple object, you can perform any operation that a tuple supports, like indexing, iteration etc.

```
def print_arguments(*args):
    print(args)

print_arguments(1, 54, 60, 8, 98, 12)
# Prints (1, 54, 60, 8, 98, 12)
```

**kwargs

- The ** syntax is similar, but it only works for keyword arguments. It collects them into a new dictionary, where the argument names are the keys, and their values are the corresponding dictionary values.

```
def print_arguments(**kwargs):
    print(kwargs)

print_arguments(name='Bob', age=25, job='dev')
# Prints {'name': 'Bob', 'age': 25, 'job': 'dev'}
```

Return Value

- To return a value from a function, simply use a `return` statement. Once a return statement is executed, nothing else in the function body is executed.

```
# Return sum of two values
def sum(a, b):
    return a + b

x = sum(3, 4)
print(x)
# Prints 7
```

Return Multiple Values

- Python has the ability to return multiple values, something missing from many other languages. You can do this by separating return values with a comma.

```
# Return addition and subtraction in a tuple
def func(a, b):
    return a+b, a-b

result = func(3, 2)

print(result)
# Prints (5, 1)
```

Docstring

- You can attach documentation to a function definition by including a string literal just after the function header. Docstrings are usually triple quoted to allow for multi-line descriptions.

```
def hello():
    """This function prints
       message on the screen"""
    print('Hello, World!')
```

Questions

- **1.** Write a Python function to find the maximum of three numbers.

Questions

- **2.** Write a Python function to sum all the numbers in a list.
  Sample List : (8, 2, 3, 0, 7)
  Expected Output : 20

Questions

- **3.** Write a Python function to multiply all the numbers in a list.
  Sample List : (8, 2, 3, -1, 7)
  Expected Output : -336

Questions

- **4.** Write a Python program to reverse a string.
  Sample String : "1234abcd"
  Expected Output : "dcba4321"

Questions

- **5.** Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.

Questions

- **6.** Write a Python function to check whether a number falls within a given range.

Questions

- **7.** Write a Python function that accepts a string and counts the number of upper and lower case letters.
  Sample String : 'The quick Brow Fox'
  Expected Output :
  No. of Upper case characters : 3
  No. of Lower case Characters : 12

Questions

- **8.** Write a Python function that takes a list and returns a new list with distinct elements from the first list.
  Sample List : [1,2,3,3,3,3,4,5]
  Unique List : [1, 2, 3, 4, 5]

Thank you

More on functions in next class