Functions

A function is a reusable block of code or programming statements designed to perform a certain task. To define or declare a function, Python provides the *def* keyword. The following is the syntax for defining a function. The function block of code is executed only if the function is called or invoked.

Declaring and Calling a Function

When we make a function, we call it declaring a function. When we start using the it, we call it *calling* or *invoking* a function. Function can be declared with or without parameters.

```
# syntax
# Declaring a function
def function_name():
    codes
    codes
# Calling a function
function_name()
```

Function without Parameters

Function can be declared without parameters.

Example:

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

def add_two_numbers ():
    num_one = 2
    num_two = 3
    total = num_one + num_two
    print(total)
add_two_numbers()
```

Function Returning a Value- Part 1

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.

```
def generate_full_name ():
    first_name = 'Prasanta'
    last_name = 'Biswal'
    space = ' '
    full_name = first_name + space + last_name
    return full_name
print(generate_full_name())

def add_two_numbers ():
```

```
    num_one = 2
    num_two = 3
    total = num_one + num_two
    return total
print(add_two_numbers())
```

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

```
  # syntax
  # Declaring a function
  def function_name(parameter):
    codes
    codes
  # Calling function
  print(function_name(argument))
```

Example:

```
def greetings (name):
    message = name + ', welcome to Python for Everyone!'
    return message

print(greetings('Prasanta'))

def add_ten(num):
    ten = 10
    return num + ten
print(add_ten(90))

def square_number(x):
    return x * x
print(square_number(2))

def area_of_circle (r):
    PI = 3.14
    area = PI * r ** 2
    return area
print(area_of_circle(10))

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
```

- 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. Let us check a function with two parameters:

```
  # syntax
  # Declaring a function
  def function_name(para1, para2):
    codes
    codes
  # Calling function
  print(function_name(arg1, arg2))
```

Example:

```
def generate_full_name (first_name, last_name):
    space = ' '
      full_name = first_name + space + last_name
      return full_name
print('Full Name: ', generate_full_name('Prasanta','Biswal'))

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

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

print('Age: ', calculate_age(2024, 1819))

def weight_of_object (mass, gravity):
    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))
```

Passing Arguments with Key and Value

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

```
# syntax
# Declaring a function
def function_name(para1, para2):
    codes
    codes
# Calling function
print(function_name(para1 = 'John', para2 = 'Doe')) # the order of
arguments does not matter here
```

Example:

```
def print_fullname(firstname, lastname):
    space = ' '
    full_name = firstname  + space + lastname
    print(full_name)
print(print_fullname(firstname = 'Prasanta', lastname = 'Biswal'))

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

Function Returning a Value- Part 2

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.

- Returning a string: **Example:**

```
def print_name(firstname):
    return firstname
print_name('Prasanta') # Prasanta

def print_full_name(firstname, lastname):
    space = ' '
    full_name = firstname  + space + lastname
    return full_name
print_full_name(firstname='Prasanta', lastname='Biswal')
```

- Returning a number:

Example:

```
def add_two_numbers (num1, num2):
    total = num1 + num2
    return total
print(add_two_numbers(2, 3))

def calculate_age (current_year, birth_year):
    age = current_year - birth_year
    return age;
print('Age: ', calculate_age(2019, 1819))
```

- Returning a boolean: **Example:**

```
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
```

- Returning a list: **Example:**

```
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))
```

Function with Default Parameters

Sometimes we pass default values to parameters, when we invoke the function. If we do not pass arguments when calling the function, their default values will be used.

```
# syntax
# Declaring a function
def function_name(param = value):
    codes
    codes
# Calling function
function_name()
function_name(arg)
```

Example:

```
def greetings (name = 'Peter'):
    message = name + ', welcome to Python for Everyone!'
    return message
print(greetings())
print(greetings('Prasanta'))

def generate_full_name (first_name = 'Prasanta', last_name = 'Biswal'):
    space = ' '
    full_name = first_name + space + last_name
    return full_name

print(generate_full_name())
print(generate_full_name('David','Smith'))

def calculate_age (birth_year,current_year = 2024):
    age = current_year - birth_year
    return age;
print('Age: ', calculate_age(1821))

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 surface of the Moon
```

Arbitrary Number of Arguments

If we do not know the number of arguments we pass to our function, we can create a function which can take arbitrary number of arguments by adding * before the parameter name.

```
# syntax
# Declaring a function
def function_name(*args):
    codes
    codes
# Calling function
function_name(param1, param2, param3,..)
```

Example:

```
def sum_all_nums(*nums):
    total = 0
    for num in nums:
        total += num     # same as total = total + num
    return total
print(sum_all_nums(2, 3, 5)) # 10
```

Default and Arbitrary Number of Parameters in Functions

```
def generate_groups (team,*args):
    print(team)
    for i in args:
        print(i)
print(generate_groups('Team-1','Prasanta','Brook','David','Eyob'))
```

Function as a Parameter of Another Function

```
#You can pass functions around as parameters
def square_number (n):
    return n * n
def do_something(f, x):
    return f(x)
print(do_something(square_number, 3)) # 27
```