Python 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(), input(), etc.
  - We can also create our 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.
  - By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.
  - A function can return data as a result.

Calling a Function

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, we can execute it by calling it from another function or directly from the Python prompt.

Creating a Function

In Python a function is defined using the `def` keyword:

Example

```
def my_first_function():
  print("\n This is my first function.")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_first_function():
  print("\n This is my first function.")

my_first_function()
```

```
This is my first function.
```

Arguments

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses.
- Eg:
    - c=a+b
    - print(c)
    - 'c' is an argument.

- *Arguments* are often shortened to *args* in Python documentations.
- You can add as many arguments as you want, just separate them with a comma.

- The following example has a function with one argument (frnd_name).
- When the function is called, we pass along a frnd_name, which is used inside the function to print the full name:

```
def my_function(frnd_name):
  print(frnd_name + " is my friend")

my_function("Karthik")
my_function("Devi")
my_function("Selin")
```

```
Karthik is my friend
Devi is my friend
Selin is my friend
```

Number of Arguments

- By default, a function must be called with the correct number of arguments.
- Meaning that if a function expects 2 arguments, then we have to call the function with 2 arguments, not more, and not less.

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Karthik", "Raja")
```

```
Karthik Raja
```

- If you try to call the function with 1 or 3 arguments, you will get an error:

Example

This function expects 2 arguments, but gets only 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Karthik")
```

```
Traceback (most recent call last):
  File "prgm19-Functions.py", line 37, in <module>
    my_function("Karthik")
TypeError: my_function() missing 1 required positional argument: 'lname'
```

Default Parameter Value

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

Example

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

my_function("Sweden")
my_function("America")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from America
I am from India
I am from Brazil
```

Passing a List as an Argument

* We 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.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

Example

```
def my_function(subjects):
  for x in subjects:
    print(x)

sub = ["Science", "Mathematics", "Chemistry"]
my_function(sub)
```

```
Science
Mathematics
Chemistry
```

Return Values

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

Example

```
def cube_of_number(x):
  return x*x*x

print(cube_of_number(2))
print(cube_of_number(3))
print(cube_of_number(10))
```

```
8
27
1000
```

Eg: Reusing the functions

```
def add_sub(option,n1,n2):
  if(option==1):
    return(n1+n2)

  else:
    return(n1-n2)

print(add_sub(1,9,6))
print(add_sub(11,9,6))
print(add_sub(1,2,3))
print(add_sub(2,2,3))
```

```
15
3
5
-1
```

The pass Statement

- `Function` definitions cannot be empty.

  ```
  # assume a python program has only the following line.

  def myfunction():
  ```

  ```
  (base) F:\CSE1001\Python-Programs>python prgm19-Functions.py
    File "prgm19-Functions.py", line 96

                        ^
  SyntaxError: unexpected EOF while parsing
  ```

- But if you for some reason have a `function` definition with no content, put in the `pass` statement to avoid getting an error.

Example

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

Activity:

1. Write a python program with the following requirements.
    - Read a list of integers from users.
    - Pass the list (say **list_numbers**) to a function (say **function_processing_numbers).**
    - Return the sum of all positive numbers.
    - Return the sum of all negative numbers.
    - Return the sum of all odd numbers
    - Return the sum of all even numbers.
    - Print the above results legibly.