- A function is a block of code which only runs when it is called. We can pass data, known as parameters, into a function.

In [1]:
```
#creating a function
def function_name():
    print("my function")
```

In [2]:
```
#calling the function
def function_name():
    print("my function")

function_name()
```

```
my function
```

In [3]:
```
# ARGUMENTS: information can be passed into functions as arguments
#creating the function
def function_name(arg1):
    print(arg1,"to python")   #print(arg1 + "to python")

#calling the fucntion
function_name('welcome')
```

```
welcome to python
```

In [4]:
```
def add(num1: int, num2: int, word: str):
    num3 = num1 + num2
    a = word
    return num3, a

add(5, 10, "hi there")
```

Out[4]:
```
(15, 'hi there')
```

In [5]:
```
def is_prime(n):
    if n in [2,3]:
        return True
    if (n==1) or (n%2==0):
        return False
    r = 3
    while r *r <= n:
        if n%r == 0:
            return False
        r += 2
        return True

#calling the function
print(is_prime(13))
print(is_prime(32))
```

```
True
False
```

# Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following 4 types of function arguments.

- Default argument
- Keyword arguments (named arguments)
- Positional arguments
- Arbitrary arguments (variable-length arguments *args and* *kwargs)

In [6]:
```
#Default arguments: A default argument is a parameter that assumes a default v
#once we have a default argument, all the arguments to its right must also have

def func(a,c,b=10):
    print("a:", a, '\n', "b:",b, '\n', 'c:', c)

func(20,30)
```

```
a: 20
 b: 10
 c: 30
```

In [7]:
```
def my_function(country = "India"):
    print("I am from " + country)

my_function()
my_function("Sweden")
my_function("Norway")
my_function("Brazil")
```

```
I am from India
I am from Sweden
I am from Norway
I am from Brazil
```

In [8]:
```
# Keyword Arguments: The idea is to allow the caller to specify the argument n
def student(firstname, lastname):
    print(firstname, lastname)


# Keyword arguments
student(firstname='abc', lastname='def')
student(lastname='def', firstname='abc')
```

```
abc def
abc def
```

In [9]:
```
#Positional Arguments
#We used the Position argument during the function call so that the first argu
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)

# argument is given in order
print("Case-1:")
nameAge("abc", 23)

# argument is not in order
print("\nCase-2:")
nameAge(23, "abc")
```

```
Case-1:
Hi, I am abc
My age is  23

Case-2:
Hi, I am 23
My age is  abc
```

In [10]:
```
#  Arbitrary Keyword  Arguments
#In Python Arbitrary Keyword Arguments, *args, and **kwargs can pass a variable

# *args in Python (Non-Keyword Arguments)
# **kwargs in Python (Keyword Arguments)

#Checkout my github page for more examples on Arbitary Keyword Arguments
https://github.com/Vaishnavi-Chandrashekar/Python-Concepts/blob/main/args%20an
```

```
  Input In [10]
    https://github.com/Vaishnavi-Chandrashekar/Python-Concepts/blob/main/arg
s%20and%20kwargs.ipynb (https://github.com/Vaishnavi-Chandrashekar/Python-Con
cepts/blob/main/args%20and%20kwargs.ipynb)
          ^
SyntaxError: invalid syntax
```

# Return Statement

Syntax: return [expression_list]

In [11]:
```
def my_function(x=10):
    return 5 * x

print(my_function())
print(my_function(30))
```

```
50
150
```

# The pass Statement

Function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

In [12]:
```
def function():
    pass
```