Day 08.

Functions With Inputs, Arguments & parameters.

```
def my-function():
    # Do this.
    # Then do this.
    # At last do this.
```

But, now let assume that the function that you want to execute need some kind of information or data to execute your needs like.

Case I

```
def greet():
    print("Hello")
    print("How are you?")

greet()
```

case II

```
def greet(Abhishek):
    print("Hello" + Abhishek)
    print("How are you" +
          Abhishek + "?")

greet(Abhishek)
```

Case I

The function everytime Execute with same sort of statements and it don't need any kind of Arguments.

Whereas,

Case II

In this function, we are paring ~~pass~~ ~~Parameters~~ Argument into the greet function and it gonna recieve it as ~~Argument~~ Parameter, Now, the ~~Argument~~ Parameter could be used to Execute the function.

Positional vs Keyword Arguments.

In function, we can also pass two parameters lets see How we can do that,

```
def sum (a, b):
    return a+b
```

→ where, we are actually asking for two argument named a & b.

Calling :-

```
Sum(5, 6)
```

→ The value, 5 & 6 are passed in the sum function are parameters.

Now,

```
def sum (a,  b)
         ↑   ↑
Sum  (5,  6)
```

The sequence is maintained in the function

Therefore, 5 would go to a.
& 6 would go to b.

(Positional Argument)
↓
Default.

Syntax

```
def  sum (Arg1, Arg2)
    .
    .
    .

Sum( 5 ,  6)
```

Note.

You must keep in mind about the positing.

There is another way to solve this positional issue via Keyword Arguments.

Keyword Arguments.

In Keyword Arguments, we pass the parameters in the function call via using Argument name.

Using this, we can change the order in function, as the Arguments works as keyword for the value passing.

eg-

```
def my-function (A, b, C)
    # . . . . .
    # . . . . .
    # . . . . . .
```

```
my-function (A = 1, b = 2, C = 3)
```

or.

```
my-function (C = 3, A = 1, b = 2)
```

↳ Both the function call are okay and the Execution is same as we are assigning the values in the function call only, overriding the default series.

Coding Exercise (8.1).

→ PAINT AREA CALCULATION. ←

→ 1 can of Paint can cover 5 square meters

Given a random height and width of wall

Calculate how many cans of paint you'll need to ~~by~~ buy.

⇒ . (height x width) ÷ 5 square meter

= no. of cans.

height → input

width → input.

```
height = int(input(" Height of wall: "))
width  = int(input(" Width of wall: "))
Coverage = 5

def paint_can_cal(height, width, cover):
    return (height x width) ÷ cover

a= paint_can_cal(height = height, width = width,
                 cover = coverage)
print(" The no. of can's required is : "+ a)
```

→ we used Keyword Argument for betterment.

Now there may be cases where no. of can came 1.6, 2.7 or 0.3.

But we know that we can only by whole number of cans.

for that :-

```
import math
```

and, `math.ceil((width x height) ÷ coverage)`

Exercise 8.2

→ Prime number Checker ←

```
def prime_checker(number):
    is_prime = True
    for i in range(2, number):
        if number % i == 0:
            is_prime = False
    if is_prime:
        print("It is a Prime number")
    else:
        print("It is not a Prime number").
```

DAY-08. Project.

Caesar Cipher.

```
def encode(text, shift):
    Cipher_Text = " "
    for letter in text:
        index = alphabet.index(letter)
        index = index + shift.
        new_letter = alphabet[index]
        Cipher_text += new_letter
    print(f" The encoded text is {cipher_Text}")

def decode ( text, shift):
    Raw_Text = " "
    for letter in text:
        index = alphabet.index(letter)
        index = index - shift
        new_letter = alphabet [index]
        Raw_Text += new_letter
    print(f" The Raw message is {Raw_Text}")

alphabet = [ . . . . . ]
```

⋮