→ "Functions in Python" :→

Python function is a block of statements that perform a specific task.

Some benefits of using functions :→

1) Increase Code Readability.
2) Increase Code Reusability.

"Python function Declaration" :→

Syntax to declare a function is :

keyword (def) | Name of function (function_name) | list of parameters (parameters)

```
def function_name(parameters):
    # statements }  → "statements to be executed"
    return expression
```

↓ return value, that function will return

→ Types of functions in Python :

1) Built-in library functions :→ There are standard functions in python that are available to use.

2) User-defined functions :→ We can create our own functions based on our requirements.

→ Creating a function in Python :→

Note :→ We can define a function in Python, using the "def" keyword. We can add any type of functionalities & properties to it as we require.

Example :>

```
def fun():
    print("Welcome to Python Programming")
```

→ Calling a function in Python :-

Note :> we can call a function by simply using its name.

Example: fun()

O/p: Welcome to Python Programming.

→ Python function with parameters :>

Just like C/C++ or Java, in python also we can define the return type of the function and datatype of arguments.

Syntax is as :>

```
def function_name( parameter : data_type ) --> return_type:

    # body of the function
    return expression.
```

Eg:>

```
def add(num1: int, num2: int) --> int:
    num3 = num1 + num2
    return num3

num1, num2 = 5, 15
ans = add(num1, num2)
print(f"The addition of {num1} and {num2} is {ans}.").
```

→ Python function Arguments:

Arguments are the values passed inside the parantheis of the function. A function can have any number of arguments seperated by a comma.

Eg:-
```
def evenodd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd").
```

→ function to find sum of first "n" natural numbers:

```
def sum_natural_numbers(n: int) -> int:
    total = 0
    for i in range(1, n+1):
        total += 1
    return total.

result = sum_natural_numbers(10)
print(f"The sum of the first 10 natural numbers is {result}.").
```

or

```
def sum_natural_numbers(n: int) -> int:
    return( n * (n+1) // 2

result = sum_natural_numbers(10)
print(f"The sum of the first 10 natural numbers is {result}.")
```

→ function to check whether a number is prime or not.

Method 1 :→ "Using sum of divisors"

```
def is_prime(n: int) → bool:
    if n < 2:
        return False

    sum_of_divisors = sum(i for i in range(1, n) if n % i == 0)

    return sum_of_divisors == 1

num = 29
if is_prime(num):
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

Method 2 :→ Check for divisors method.

To determine if n is prime or not, we need to check for divisors from 2 up to $\sqrt{n}$. This is because if n can be factored into two factors a and b (i.e. $n = a \times b$), then at least one of those factors must be less than or equal to $\sqrt{n}$.

if both factors were greater than $\sqrt{n}$, then their product would exceed n.

PTO

Perform divisibility test.

for each integer i from 2 to $\lfloor \sqrt{n} \rfloor$. check if n is divisible by i.

if $n \bmod i = 0$, then n is not prime (it has a divisor other than 1 and itself).

if no such i divides n, then n is prime.

Eg:→ Let's say, we want to check if 29 is prime.

1) Calculate $\sqrt{29}$, which is approx. 5.39.

2) Check for divisors from 2 to $\lfloor 5.39 \rfloor = 5$:

Check 2: $29 \bmod 2 \neq 0$
Check 3: $29 \bmod 3 \neq 0$
Check 4: $29 \bmod 4 \neq 0$
Check 5: $29 \bmod 5 \neq 0$

Since 29 is not divisible by any number from 2 to 5, we can say that 29 is a prime number.

```
def is_prime(n: int) -> bool:
    if n< 2
        return False
    for i in range(2, int(n** 0.5)+1):
        if n%i ==0:
            return False
    return True.
```

[P.T.O]

```
num = 29

if is_prime(num):
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

→ Types of Python Function Arguments :

It supports various types of arguments that can be passed at the time of function call.

We have following function argument types in Python :→

1) Default Arguments,
2) Keyword Arguments (named arguments)
3) Positional Arguments
4) Arbitrary Arguments (Variable-length arguments *args and **kwargs)

1) Default Arguments :

A default argument is a parameter that assumes a default value, if a value is not provided in the function call for that argument.

Eg:-

```
def Myfun(x, y=50):
    print("x:", x)
    print("y:", y)

Myfun(10)
```

O/P.

x: 10
y: 50

Note:→ Like C++, any no. of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

2> "Keyword Arguments":

The idea is to allow the caller to specify the argument name, with values so that the caller does not need to remember the order of parameters.

Eg:→
```
def student (firstname, lastname):
    print (firstname, lastname)

student (firstname = 'Vipul', lastname = 'Sharma')
student (lastname = 'Sharma', firstname = 'Vipul')
```

O/p
```
Vipul Sharma
Vipul Sharma
```

3> "Positional Arguments":

Eg:-
```
def nameAge (name, age):
    print ("Hi, I. am", name)
    print ("My age is ", age)

print("Case-1:")
nameAge("Vipul", 30)
print("\nCase-2:")
nameAge(30,'Vipul')
```

O/p
```
Case-1:
Hi, I am Vipul
My age is 30
Case-2:
Hi, I am 30
My age is Vipul
```

(4) Arbitrary keyword Arguments:

In python Arbitrary Keyword Arguments. *args. and **kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:

1) *args in Python (Non-Keyword Arguments)
2) **kwargs in Python (Keyword Arguments)

Example1: Variable length non-keyword argument:

```
def myfun(*argv):
    for arg in argv:
        print(arg)

myfun('Hello', 'Welcome', 'to', 'Python')
```

O/p.

Hello
welcome
to
Python.

Example2: Variable length keyword arguments:

```
def myfun(**kwargs):
    for key, value in kwargs.items():
        print("%s == %s" %(key, value))

myfun(first = 'Hello', second = 'Welcome', third = 'to',
      fourth = 'Python')
```

O/p: first == Hello
second == welcome
third == to
fourth == Python