Function

In [ ]:
```
Definition : A function is a reusable block of code that perform specific task,
                returns the output.


Core Usages:

    1- Function hide implemenation details and provide a simple interface
    2- Function is use for reusability(write ones and use multiple times)
    3- Modularity: Break complex promplex problem into smaller pieces.

Synax:

def <name_of_the_function>(parametes):
    <body_of_statement>
    return <returns_the_value>
```

Type of Function:

1- Built-in Function

2- User defined Function

3- Anonymous Function

In [ ]:
```
# 1- Built-in Function

print("Hello")
len("john")
type()
input()
int()
str()
dict()
```

In [1]:
```
# 2- User defined Function

# 1

def greet():
    return f"Hello world"

greet()
```

Out[1]:
```
'Hello world'
```

In [10]:
```
# 2

# Function
def greet(name):
    return f"Hello : {name}"
```

```
# calling the function
print(greet("John"))
print(greet("Bob"))
```

```
Hello : John
Hello : Bob
```

In [ ]:
```
# Defined input Paraters types

# 1- Default arguments(parameters)

# 2- Required arguments

# 3- keyword arguments

# 4- variable-length arguments
```

In [17]:
```
# 1- Default arguments(parameters)

def user_name(name = "Bob"):
    return name

print(user_name())

# 2
def greet_with_title(name = "John",title = "Mr"):
    return f"Hello {title} {name}"

greet_with_title("bob")
```

```
Bob
```

Out[17]:
```
'Hello Mr bob'
```

In [20]:
```
# 2- Required arguments(parameters)

def add(value1, value2):
    sum_output = value1 + value2
    return sum_output

add(10,20)
```

Out[20]:
```
30
```

In [22]:
```
# 3- Keyword arguments

def print_the_name(name, age):
    return f" {name} {age}"

print_the_name(name = "John Cena", age = 40)
```

Out[22]:
```
' John Cena 40'
```

In [25]:
```
# 4 Variable length arguments

def names(*names):
    for name in names:
        print(name)
```

```
Lst = ["amit","pranav","deepak","raghav"]
names(Lst)
```

```
['amit', 'pranav', 'deepak', 'raghav']
```

In [ ]:

```
```

In [ ]:

```
what is the *args vs **kwargs?
```

In [7]:

```
def square(x,y,z,m):
    z1  = x * y * z * m
    return z1

print(square(10,2,3,4))
```

```
240
```

In [13]:

```
def square(*square):
    square = [s * s for s in square]
    return square

print(square(2,3,4,5,6,6,7,8,8,98,6,6,5,5,55,5,5))
```

```
[4, 9, 16, 25, 36, 36, 49, 64, 64, 9604, 36, 36, 25, 25, 3025, 25, 25]
```

In [2]:

```
def sum_all(*n):
    z = sum(n)
    return z

print(sum_all(2,3,4,5,6,6,7,8,8,98,6,6,5,5,55,5,5,4,4,4,4,4,4,32,223,423,4,24344
```

```
25284
```

In [15]:

```
# **kwargs

def names(**names):
    for name in names.items():
        print(name)

names(name="John",age= 22,location= "NYC",mobile_number=232322232)
```

```
('name', 'John')
('age', 22)
('location', 'NYC')
('mobile_number', 232322232)
```

In [5]:

```
def print_info(**info):
    for value in info.items():
        print(f"{value}")

print_info(name="Alice",age= 30,job="engineer",course="graduate")
```

```
('name', 'Alice')
('age', 30)
('job', 'engineer')
('course', 'graduate')
```

In [10]:

```
def print_info(**info):
    for value in info.items():
        print(f"{value}")
```

```
print_info(name="Alice",age= 30,job="engineer",course="graduate",city="delhi",ed
```

```
('name', 'Alice')
('age', 30)
('job', 'engineer')
('course', 'graduate')
('city', 'delhi')
('education', 'mumbai')
```

Anonymous Function

In [11]:
```
def add(m,n):
    z = m + n
    return z
add(10,20)
```

Out[11]:
```
30
```

In [12]:
```
add_value = lambda x,y: x + y
add_value(10,20)
```

Out[12]:
```
30
```

In [16]:
```
hello_world = lambda : "hello world"
hello_world()
```

Out[16]:
```
'hello world'
```

In [17]:
```
v1 = lambda : 2 * 2
v1()
```

Out[17]:
```
4
```

In [27]:
```
lst = [2,4,4,5,66,57,67,67,8678,678,78,7]
list(filter(lambda x : (x % 2 == 0),lst))
```

Out[27]:
```
[2, 4, 4, 66, 8678, 678, 78]
```

In [25]:
```
double = lambda x :  x * 2 * 2
double(24)
```

Out[25]:
```
96
```

In [ ]:
```
# Recurvise Function (a function call itself is called recursive function)

#factorial = 5 =  5 x 4 x 3 x 2 x 1 = 120
```

In [30]:
```
def factorial(n):
    if n == 1:
        return 1

    return n * factorial(n-1)

factorial(5)
```

Out[30]:
```
120
```

In [ ]:
```
# fibonacci number

0, 1, 1, 2, 3, 5, 8, 13, 21

0,1,1,2,3,5,8

0,1,1,2,3,5,8,13,21,34,55
```

In [31]:
```
def fibonacci_number(n):
    if n <= 1:
        return n

    return fibonacci_number(n-1) + fibonacci_number(n-2)

fibonacci_number(6)
```

Out[31]:
```
8
```

In [32]:
```
fibonacci_number(10)
```

Out[32]:
```
55
```

In [ ]: