```
"""
Function:
A block of code that performs certain task

Uses of Functions:
1. Code Reusability
2. Debugging
3. Enhances Readability
4. Modularity
5.


Syntax for declaring functions:
def <function_name>(<parameters>):
    <function_body>


Syntax for function call:
function_name(<parameters>)


Notes:
1. A function is executed only when it is called.
2. All the variables declared inside the function have local scope
   and are destroyed once the function execution is completed.
3. A function can have any number of parameters.
4. A function can have an optional return statement.
   return statement is used to return the values which are calculated inside the
function to return from the function
5. When a function encounters a return statement, it will return the value from the
function and the control will exit from the function.
6. A function can be declared once and called unlimited number of times.
7. Functions can have parameters. These can be changed while calling the function.
8. The parameters in the function declaration are called function parameters
   and those in the function call are called function arguments.
9. The Function arguments are positional.
10. All the functions will have a default return value of None
"""
def my_function():
    print("Inside the Function...")
    print("Hello World!!!!")

my_function()



def function_with_parameters(a, b, c): # Parameters
    print(f"a = {a}, b = {b} and c = {c}")


function_with_parameters(10, 20, 30) # Arguments
function_with_parameters(30, 20, 10)


def perform_addition(a, b):
    print(a + b)


perform_addition(10, 20)
```

```
perform_addition(100, 200)
perform_addition(1000, 2000)


"""
Scope and Lifetime of variables:
Scope:
The parts of the program in which the variable can be accessed.

Lifetime:
The duration for which the variable can be accessed.
"""
def my_function():
    a = 10
    print("Inside Function...")
    print("Value of a is ", a)


my_function()
print("Outside Function....")
# print("Value of a is ", a)


"""
Scopes:
1. local scope:
   The variables declared inside the function.
2. global scope:
   The variables declared outside of all the functions.
   the global variables can be created and accessed using the global keyword.
3. nonlocal scope:
   The variables declared in the parent functions can be accessed in all the inner
functions.
"""
print('*' * 50)
b = 10
def func():
    print("Inside Function...")
    print("Value of b = ", b)


func()
print("Outside Function...")
print("Value of b = ", b)


c = 50
print(id(c))
def my_fun():
    global c
    c = 100
    print(id(c))
    print("Inside Function....")
    print("Value of c = ", c) #


my_fun()
print("Outside Function ")
print("Value of c = ", c) #
```

```
def func1():
    global d
    d = 100
    print("Value of d = ", d)


func1()
print("Value of d = ", d)



def foo():
    print("Line 129")
    return 10
    print("Line 131")


a = foo()
print(a)


def bar():
    print("Line 139")

b = bar()
print(b)


"""
Types of Functions:
1. Function without arguments without return
2. Function without arguments with return
3. Function with arguments without return
4. Function with arguments with return
"""

#1. Functions without arguments without return
def print_line():
    print("-----------------------------------")


print_line()


import random
# 2. Functions without arguments with return
def generate_random_number():
    return random.randint(1, 6)


for i in range(5):
    print(generate_random_number())
print(generate_random_number())


# 3. Functions with arguments without return
def print_stars(n):
```

```
    print('*' * n)


print_stars(10)
print_stars(30)


# 4. Functions with arguments with return
def multiply(a, b):
    return a * b

print(multiply(10, 20))
print(multiply(4, 2))


def func1():
    def func2():
        print("Inside Function2...")
    print("Inside Function1....")
    func2()

func1()


def func1():
    x = 100
    def func2():
        x = 500
        print("Inside Function2...")
        print(id(x))
        print("X = ", x)
    print("Inside Function1....")
    print("X = ", x)
    print(id(x))
    func2()
    print("After Calling Function2: ")
    print("X = ", x)

func1()


def func1():
    x = 100
    def func2():
        nonlocal x
        x = 500
        print("Inside Function2...")
        print(id(x))
        print("X = ", x)
    print("Inside Function1....")
    print("X = ", x)
    print(id(x))
    func2()
    print("After Calling Function2: ")
    print("X = ", x)

func1()
# print(x)
```

```
def func1():
    x = 100
    def func2():
        nonlocal x
        x = 200
        print("Inside Function2...")
        print("Value of X = ", x)
    func2()
    print("Inside Function1...")
    print("Value of X = ", x)


func1()


def func1():
    x = 100
    def func2():
        nonlocal x
        x = 200
        print("Inside Function2...")
        print("Value of X = ", x)
        def func3():
            nonlocal x
            x = 500
            print("Inside Function2...")
            print("Value of X = ", x)
        func3()
    func2()
    print("Inside Function1...")
    print("Value of X = ", x)


func1()


"""
Passing Parameters to a Function:
1. Positional Arguments
2. Default Arguments
3. Keyword Arguments
4. Variable Arguments
"""


# 1. Positional Arguments
def add(a, b, c):
    return a + b + c


add(10, 20, 30)


# 2. Default Arguments
def foo(a=0, b=0, c=0, d=0, e=0):
    print(f"Value of a is {a}, b is {b}, c is {c}, d is {d}, e is {e}")
```

```
# foo(1, 2, 3, 4, 5)
# foo(1, 2, 3, 4)
# foo(1, 2, 3, 4, 5, 6)
foo()
foo(1)
foo(1, 2)
foo(1, 2, 3)
foo(1, 2, 3, 4)
foo(1, 2, 3, 4, 5)


# 3. Keyword Arguments
def foo(a=0, b=0, c=0, d=0, e=0):
    print(f"Value of a is {a}, b is {b}, c is {c}, d is {d}, e is {e}")


# foo(1, 0, 0, 0, 5)
foo(a = 1, e = 5)
foo(a = 1, d = 4)


def foo(a, d, e, b=0, c=0):
    print(f"Value of a is {a}, b is {b}, c is {c}, d is {d}, e is {e}")


# foo(1, 2, 3)
# foo(a=1, b = 2, c = 3, d = 4, e = 5)
# foo(a=1, f=5, g=6)


#4. Variable Number of Arguments
# *args, **kwargs
def add(*values):
    print(type(values))
    result = 0
    for item in values:
        result += item
    print(result)


add(1)
add(1, 2)
add(1, 2, 3)
add(1, 2, 3, 4)
add(1, 2, 3, 4, 5)


def print_args(**kwargs):
    print(type(kwargs))
    for key, value in kwargs.items():
        print("Key = ", key, "Value = ", value)


print_args(key1="Value1", k2="v2", key3="Value3", k4="V4")
print_args(key1dfgd="Value1", k2wew="v2", key3wewe="Value3")


def foo(*args, **kwargs):
    print(args)
```

```
    print(kwargs)


foo(1, 2, 3, 4, k1="v1", k2="v2", k3="v3", k4="v4")
foo(1, 2, k1="v1", k2="v2")
```