SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107
An Autonomous Institution
Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF CSE (IoT & CYBER SECURITY INCLUDING BLOCKCHAIN TECHNOLOGY)

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- **A readable, dynamic, pleasant, flexible, fast and powerful language**

Recap

- “break” statement is used terminate the loop in between the iterations
- “continue” statement is used to skip an iteration
- “pass” statement acts as a placeholders for future code
- Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task.
- Flow of execution is the order in which statements are executed

Agenda

- Functions
  - Arguments vs parameters
  - Types of arguments
- Fruitful functions
- Local and global scope

3.4 Functions

Arguments vs parameters

Arguments	Parameters
Arguments are used when a function is called	Parameters are used when function is to be defined
An argument is a value passed to a function when calling the function.	A parameter is a named entity in a function definition that specifies an argument that the function can accept.
Very often, there is a 1:1 mapping between arguments and...	...the parameters defined in the function.
Arguments can be constants, local variables or objects	Parameters cannot be a constant.
There are keyword arguments, and there are positional arguments. Anything that is not a keyword argument is a positional argument.	Keyword parameters and positional parameters

cont'd

3.4 Functions

Arguments vs parameters

The scope of the arguments are relevant only in the called function	The scope of the parameters have valid scope only within the function where they occur
Types of Arguments **Positional/Required**: Arguments without a name. **keyword**: Arguments with a name. **default**: a value provided in a function declaration that is automatically assigned. It is more precise to refer to this as "parameter with default value" **Variable-length arguments:** Variable length argument make function calls with arbitrary number of arguments	**Types of Parameters** **positional-or-keyword:** parameters in a function definition, with or without default values. **positional-only:** Only found in builtin/extension functions. **var-positional:** This is the *args. **keyword-only**: parameters that come after a * or *args, with or without default values. **var-keyword:** This is the **args

3.4 Functions

Types of Arguments

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

3.4 Functions

Types of Arguments

1. Required arguments

- Required arguments are the arguments passed to a function in correct positional order.
- The number of arguments in the function call should match exactly with the function definition.

3.4 Functions

Types of Arguments

1. Required arguments

Example:

```
#Function definition where str is the required argument
def display(str):
        print(str)

#main script
str="hello"
display(str)
```

Output:

```
hello
>>> |
```

3.4 Functions

Types of Arguments

2. Keyword arguments

- When keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

3.4 Functions

Types of Arguments

2. Keyword arguments

Example:

```
def display(val1,val2,val3):
        print("The string is:",val1)
        print("The integer is:",val2)
        print("The float is:",val3)

#Main script-Fucntion calling with keyword arguments
display(val3=58.62,val1="hello",val2=28)
```

Output:

```
The string is: hello
The integer is: 28
The float is: 58.62
```

3.4 Functions

Types of Arguments

3. Default arguments

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

3.4 Functions

Types of Arguments

3.Default arguments

Example:

```
def printinfo(name, age = 35 ):
        "This prints a passed info into this function"
        print("Name: ",name)
        print("Age ",age)
        return
#Calling printinfo function
printinfo(age=50, name="miki" )
printinfo(name="miki")
```

Output:

```
Name:  miki
Age  50
Name:  miki
Age  35
```

3.4 Functions

Types of Arguments

4. Variable-length arguments

- Variable length argument make function calls with arbitrary number of arguments
- These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.
- **Syntax:**

```
def functionname([formal_args,] *var_args_tuple ):
        "function_docstring"
        function body
        return [expression]
```

3.4 Functions

Types of Arguments

4. Variable-length arguments

Example:

```
# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print ("Output is: ")
   print (arg1)
   for var in vartuple:
      print (var)
   return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Output:

```
Output is:
10
Output is:
70
60
50
```

3.4 Functions – Fruitful functions

- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return None.
- If a function returns some value then it is called as fruitful function
- Def: A function with a return value is called fruitful function

3.4 Functions –Fruitful functions

Example:

```
# Function definition is here
def sum( arg1, arg2 ):
        # Add both the parameters and return them."
        total = arg1 + arg2
        print("Inside the function : ", total)
        return total;

# Now you can call sum function
total = sum( 10, 20 );
print("Outside the function : ", total)
```

Output:

```
Inside the function :  30
Outside the function :  30
```

Scope of Variables

Local and global Scope

- All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.
- The scope of a variable determines the portion of the program where you can access a particular identifier.
- There are two basic scopes of variables in Python
  1. Global variables
  2. Local variables

Scope of Variables

Local and global Scope

- Variables that are defined inside a function body have a local scope, and those defined outside have a global scope
- This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions.
- When you call a function, the variables declared inside it are brought into scope.

Scope of Variables

Local and global Scope

Example:

```
total = 0;
# This is global variable.
def sum( arg1, arg2 ):
#Add both the parameters and return them."
        total = arg1 + arg2; # Here total is local variable.
        print("Inside the function local total : ", total)
        return total;
# Calling sum function
sum( 10, 20 );
print("Outside the function global total : ", total)
```

Output:

```
Inside the function local total :  30
Outside the function global total :  0
```

Scope of Variables

Local and global Scope

- In Python, global keyword allows you to modify the variable outside of the current scope.
- It is used to create a global variable and make changes to the variable in a local context.

Scope of Variables

"global" Keyword:

Example:

```
#create a function:
def myfunction():
  global x
  x = "hello"

#execute the function:
myfunction()

#x should now be global, and accessible in the global scope.
print(x)
```

Output:

```
hello
>>> |
```

Summary

- Values present in the function calling statement are called arguments
- Variables used in the function header are called parameters
- Required, keyword, default and variable-length are types of arguments
- Variable can be created with local and global scopes
- Global keyword creates a global variable inside a block


THANK YOU