BCA-3001 Python Programming

Unit-3 Part-1: Python Functions

Compilation by Milind Bhatt

```
======================================================================
```

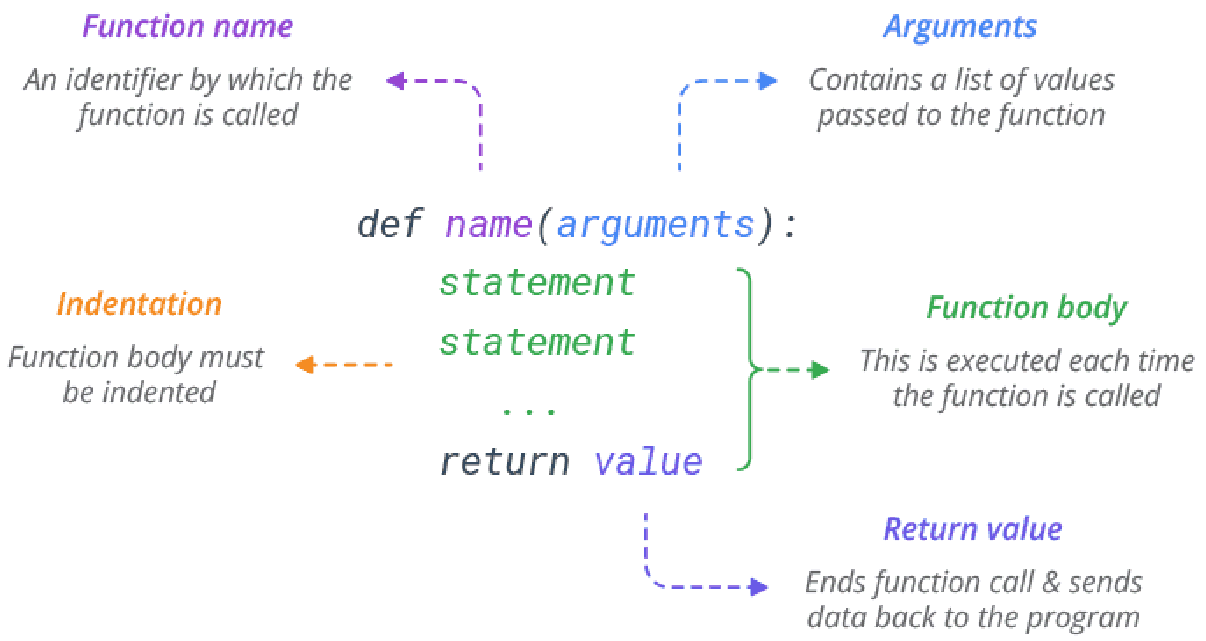
What are Python Functions?

- A function is a block of code that performs a specific task.
- It can take zero or more inputs and may or may not return one or more outputs.
- Functions are reusable and can be called from anywhere in the program.

How to define and call a Function in python?

- To define a function, use the `def` keyword to give it a name, specify the arguments it must receive, and organize the code block.
- When the fundamental framework for a function is finished, we can call it from anywhere in the program.

Example


Function name
An identifier by which the function is called
Arguments
Contains a list of values passed to the function
def name(arguments):
statement
statement
...
return value
Indentation
Function body must be indented
Function body
This is executed each time the function is called
Return value
Ends function call & sends data back to the program

```
In [ ]: # Example Python Code for calling a function
```

```python
# Defining a function
def Myfunction(string):
    #"This prints the value of length of string"
    return len(string)


# Calling the function we defined

ln1=Myfunction("Functions")
ln2=Myfunction("This prints the value of length of string")
print( "Length of the string Functions is: ",ln1)
print( "Length of the string Python is: ", ln2)
```

Pass by Reference vs. Pass by Value

- Everything in Python is considered to be an **object and can be categorized as either ‘Mutable’ or ‘Immutable’**. In the Python programming language, **all parameters are passed by reference**. I
- It shows that if we modify the worth of contention within a capability, the calling capability will similarly mirror the change.

alt text

In [ ]:

```
```

alt text

Example

In [ ]:

```python
# Example Python Code for Pass by Reference vs. Value

def change_number(num):
    num+=10

def change_list(num_list):
    num_list.append(20)

num_val=10

print("num_val before function call:", num_val)
change_number(num_val)
print("num_val after function call:", num_val)

print("-----------------------------------------------")

val_list=[5,10,15]

print("val_list before function call:", val_list)
change_list(val_list)
print("val_list after function call:", val_list)
```

Example

In [ ]:

```
# Example Python Code for Pass by Reference vs. Value
# defining the function
def square(item_list):
    squares = []
    for l in item_list:
        squares.append( l**2 )
    return squares

#calling the defined function
my_list = [17, 52, 8];
my_result = square( my_list )

print("Squares of the list are: ",my_result)
```

Function Arguments

The following are the types of arguments that we can use to call a function:

- 1.Required (positional) arguments
- 2.Keyword arguments
- 3.Default arguments
- 4.Variable-length arguments { width=50%,height:20px }

1). Required arguments

- Required arguments are those supplied to a function during its call in a predetermined positional sequence.
- The number of arguments required in the method call must be the same as those provided in the function's definition.

In [ ]:

```
def greet(name):
    print(f"Hello, {name}!")

# Required argument: 'name'
greet("Vijay")
```

2) Keyword Arguments

- keyword arguments (also known as named arguments) allow you to pass arguments to a function using their parameter names.
- This eliminates the need to remember the order in which arguments are expected by the function.
- When defining a function, you can assign default values to some parameters, turning them into keyword arguments.
- This means that when you call the function, you have the option to explicitly specify values for certain parameters by using their names in any order.

In [ ]:
```python
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is", age)

# Using keyword-only arguments
nameAge(name="Vijay", age=20)
print()
nameAge(age=20, name="Vijay")
```

In [ ]:
```python
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is", age)

# Using keyword-only arguments
nameAge("Vijay",20)
print()
nameAge(20, "Vijay")
```

3) Default Arguments

- A default argument is a boundary that takes as information a default value, assuming that no worth is provided for the argument when the function is called.
- i.e Default arguments are the values used when no such argument is passed.
- Example

In [ ]:
```python
# Python code to demonstrate the use of default arguments
# defining a function
def function( n1, n2 = 20):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)


# Calling the function and passing only one argument
print("\nPassing only one argument")
function(30)

# Now giving two arguments to the function
print("\nPassing two arguments")
function(50,30)
```

4).Variable-length arguments

- We can involve unique characters in Python capabilities to pass many arguments.
- This can be accomplished with one of two types of characters:
-
    - `args` and `kwargs` refer to arguments not based on keywords.

args (Non-Keyworded Arguments):

- This allows you to pass a variable-length, non-keyworded argument list.
- The arguments are collected into a tuple within the function.

- **You can use `*args` to accept more arguments than the formal parameters defined, making your function more versatile**.

kwargs (Keyworded Arguments):

- This allows you to pass a keyworded, variable-length argument list.
- Arguments are collected into a dictionary within the function, accessible by their keys.

In [46]:
```python
## example: args
def sum_all(*args):
    result = 0
    for num in args:
        result += num
    return result

print(sum_all(1, 2, 3, 4, 5))  # Output: 15
print(sum_all(1, 2, 3, 4, 5,6,7,8,9,10))
print(sum_all(7))
print(sum_all())
```

```
15
55
7
0
```

Example : kwargs

In [51]:
```python
## Example : kwargs
def display_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

display_info( name="Alice", age=30, city="New York")
# Output:
# name: Alice
# age: 30
# city: New York
```

```
name: Alice
age: 30
city: New York
```

What are the possible types of functions in Python?

- **Built-in Functions:** These are functions that are built into the Python language and can be used without additional code. Examples include print(), len(), sum(), min(), max(), etc.

- **User-defined Functions:** These are functions created by the programmer to perform a specific task. They are defined using the def keyword and can include parameters and a code block.

- **Recursive Functions:** These are functions that call themselves to perform a task repeatedly until a certain condition is met.

- **Lambda Functions:** These are small anonymous functions that can be defined in a single line of code. They are often used for quick, simple operations.
- **Higher-Order Functions:** These are functions that take other functions as arguments and/or return functions as output.

DEFINE THE LOCAL AND GLOBAL SCOPE OF A VARIABLE

Write a program to show local scope vs global scope.

In [53]:
```
p = 20          #global variable p
def Demo():
    q = 10      #Local variable q
    print('The value of Local variable q:',q)
    print('The value of Global Variable p:',p)

Demo()
print('The value of global variable p:',p)


try:
    print('The value of local variable q:',q)
except Exception as e:
    print(e)
```

```
The value of Local variable q: 10
The value of Global Variable p: 20
The value of global variable p: 20
name 'q' is not defined
```

Can Local and Global Variables with the Same Name are possible in python?

- `Yes` , We can use same name to two variables in a program, provided their scope should be different.
- This mean that a same-named-variable can be used with in local and global scope of visibility.

In [ ]:
```
# Yes, We can use same name to two variables in a program, provided their scope should
# This mean that a same-named-variable can be used with in local and global scope of v
# Example:

S='I Like Python'
def Demo():
    S='I Like Programming'
    print(S)


Demo()
print(S)
```

THE return STATEMENT

- The return statement is used to return a value from the function.

- It is also used to return from a function,
- i.e. break out of the function

Write a program to return the minimum of two numbers

In [ ]:
```
### Write a program to return the minimum of two numbers

def minimum(a,b):
    if a<b:
        return a
    elif b<a:
        return b
    else:
        return 'Both the numbers are equal'

print(minimum(100,55))
print(minimum(100,100))
```

Write a program to pass the radius of a circle as a parameter to a function `area_of_circle(radius)`.

Return the value none if the value of the radius is negative or return the area of the circle

In [ ]:
```
def area_of_Circle(radius):
    if radius<0:
        print(' Try Again, Radius of circle cannot be Negative ')
        return None

    else:
        print('Radius = ',radius)
        return 3.1459*radius**radius

print('Area of Circle =',area_of_Circle(2))
```

What will be the output of the above program?

In [56]:
```
def calc_abs(x):
    if x<0:
        return -x
    elif x>0:
        return x

print(calc_abs(0))
```

```
None
```

Returning Multiple Values

Yes, in python multiple values can be returned at a time.

Example-1 Write a function `calc_arith_op(num1, num2)` to calculate and return at once the result of arithmetic operations such as addition and subtraction.

In [ ]:
```
def calc_arith_opn(num1, num2):
    add=num1+num2
    sub=num1-num2
    mult=num1*num2
    return add,sub,mult  #Return multiple values


print(' ',calc_arith_opn(10,20))

ADD,SUB,MULT=calc_arith_opn(10,20)
print(ADD)
print(SUB)
print(MULT)
```

Example-2 Write a program to return multiple values(as square and cube of a given number)from a function

In [ ]:
```
def compute(num1):
    print('Received Number is = ',num1)
    print("compute function is performing square and cube operation......\n")
    return num1*num1, num1*num1*num1

square,cube=compute(4)
print('Square = ',square,'\nCube = ',cube)
```

RECURSIVE Functions in Python

Write a recursive function which computes the nth Fibonacci number.

Fibonacci numbers can be represented as:

- Fib(0)= 1,
- Fib(1) = 1
- Fib(n)= Fib(n-1)+Fib(n-2).

Write this as a Python code and then find the 8th Fibonacci number.

In [ ]:
```
def fib(n):
    if n==0:
        return 1
    if n==1:
        return 1

    return fib(n-1)+fib(n-2)
```

```
print(' The Value of 8th Fibonacci number = ',fib(8))
```

Write program to Calculate the factorial of a number using recursion.

In [ ]:
```
## Calculate the factorial of a number using recursion.

def factorial(n):
    if n==0:
        return 1
    else if n==1:
        return 1
    return n*factorial(n-1)

print(factorial(5))
```

Scope of a Nested Function

- Defining a function inside a body of another function is called Nested function
- Inner function is only vsible to outer function
- If inner function called outside a outer function, then an `error` **`"name 'function' is not defined "`** will be flashed.

In [1]:
```
# Python code to show how to access variables of a nested functions
# defining a nested function
def word():
    string = 'Python functions tutorial'
    x = 5
    def number():
        print( string )
        print( x )

    number()
word()
try:
    number()
except Exception as e:
    print(e)
```

```
Python functions tutorial
5
name 'number' is not defined
```

Define Lambda functions in Python.

- Lambda functions are named after the Greek letter l (lambda).
- These are **`also known as anonymous functions`**.
- Such kind of functions are **not bound to a name**.
- They only have a code to execute that which is associated with them.

Syntax

`Name = lambda(variables): Code`

Note:

- (a) A lambda function **does not contain a return** statement.

- (b) It contains a **single expression as a body** and not a block of statements as a body

In [ ]:
```
## Example : function without lambda function

def cube(x):
    return x*x*x

print(cube(3))
```

In [5]:
```
## Example : same task of function with lambda function

cube = lambda x: x*x*x   #Define lambda function

print(cube(3))           #Call lambda function
```

```
27
```

In [13]:
```
## Example (without list comprehention) odd even lists
Even_List = []
for x in range(2, 11):
    if x % 2 == 0:
        Even_List.append(lambda x=x: 1*x)
    else:
        Even_List.append(lambda x=x: None)

for elist in Even_List:
    print(elist())

```

```
2
None
4
None
6
None
8
None
10
```

In [15]:
```
## same code with list comprehension
Even_List = [lambda x=x:1*x if (x % 2 == 0) else None for x in range(2, 11)]
#                        sT_tR  iF(CONDITON) else St_Fls
for elist in Even_List:
    print(elist())
```

```
2
None
4
None
6
None
8
None
10
```

Explanation

Creating the List of Lambda Functions:

- Initialize an empty list Even_List.
- Make a for loop to iterate over the range from 2 to 10.
- For each value of x, check if x is even using the condition x % 2 == 0.
- If x is even, we create a lambda function lambda x=x: 1*x and append it to the Even_List.
- If x is not even, we create a lambda function lambda x=x: None and append it to the Even_List

In [ ]:
```
# Example : Give ten times of the numbers in a given range
TenTimes = [lambda x=x: x * 10 for x in range(101, 121)]
for TnT in TenTimes:
    print(TnT() )


''' extended code
for x in range(101, 121):
    TenTimes.append(lambda x=x: x * 10)'''
```

In [ ]:
```
# Que??
def fn(a,b):
    try:
        fn(a+c)
    except NameError:
        print("name error")
    finally:
        print('fun_finally block')

try:
    fn(12,13)
except Excepectons as e:
    print(e)
finally:
    print("call_finally")


## output???
```

In [ ]:
```
#=================== ============================
```

Strong Numbers

- Strong Numbers are the numbers whose sum of factorial of digits is equal to the original number
- For example - The given number is 145, we have to pick each digit and find the factorial 1! = 1, 4! = 24, and 5! = 120.

Algo for the Problem Approach

1. Ask the user to enter an integer number.
2. Find the factorial of each digit in the number using the two while loop.
3. Now, sum up all the factorial number.
4. Check if it is equal to the given number.
5. Print the Output.
6. Exit

In [37]:

```python
# Prog Strong Number
#let

num=int(input("Enter a number:"))
# temporary variable  store copy of the original number
temp=num
sum=0

while(num):
    i=1
    fact=1
    rem=num%10
    while(i<=rem):
        fact=fact*i   # Find factorial of each number
        i=i+1
    sum=sum+fact
    num=num//10
if(sum==temp):
    print("Given number is a strong number")
else:
    print("Given number is not a strong number")
```

```
Given number is a strong number
```

Example Python Code for User-Defined function

In [ ]:

```python
def square( num ):
    return num**2

sqnum = square(6)    # calling a function
print( "The square of the given number is: ", sqnum )

########
sqnum = square(int(input("Enter a number : ")))    # calling a function
print( "The square of the other given number is: ", sqnum )
```

In [ ]:

```python
### Write a program to return the minimum of two numbers

def minimum(a,b):
    if a<b:
        return a
    elif b<a:
        return b
    else:
        return 'Both the numbers are equal'

print(minimum(100,55))
print(minimum(100,100))
```

Write a python program using functions to have a sum of all the numbers in the given range.

In [ ]:
```
# Write a program to illustrate the use of functions.
# Write a function to have a sum of all the numbers in the given range



sumofrange(1,25)
sumofrange(50,75)
sumofrange(90,100)

def sumofrange(x,y):
    s=0;
    for i in range(x,y+1):
        s=s+i
    print('Sum of integers from ',x,' to ',y,' is ',s)
```

Write a program to find the maximum of two numbers.

In [ ]:
```
## Write a program to find the maximum of two numbers.

def printMax(num1,num2):
    #Function Definition
    print(" num1 = ",num1)
    print(" num2 = ",num2)

    if num1>num2:
        print('The Number ',num1,' is Greater than ',num2)
    elif num2>num1:
        print('The Number ',num2,' is Greater than ',num1)
    else:
        print(' Both Numbers ',num1,',and',num2,'are equal')

'''END DEFINITION'''

#call to function printMax
printMax(20,10)
```

Write a program to find the FACTORIAL of a given number.

In [ ]:
```python
# Write a program to find the FACTORIAL of a given number.

def calc_factorial(num):
        if num>=1:
            fact=1
            print('Entered Number is: ',num)
            for i in range(1,num+1):
                fact=fact*i
            print('Factorial of Number ',num,' is = ',fact)
        else:
            print("Invalid Value")

number=int(input('Enter the Number:'))
calc_factorial(number)
```

Write a program to illustrate the use of default values in a function’s definition.

In [ ]:
```python
# Write a program to illustrate the use of default values in a function’s definition.

def greet(name,msg='Welcome to Python!!'):
    print(' Hello ',name,msg)


greet('Ajai')

greet('Sachin','Happy Learning!!')
```

In [ ]:
```python
# Write a program to illustrate the use of default values in a function’s definition.

def greet(msg='Welcome to Python!!',name):
    print(' Hello ',name,msg)


greet('Sachin')

greet('Sachin','Bye')
```

Note: During a function’s definition, any number of parameters in a function can have default values. But once we have a default value to a parameter, all the parameters to its right must also have default values.

For example, if we define a function’s definition as:

def greet(msg='Welcome to Python!!', name): #Error

- **Syntax Error:** `Non-default argument follows default argument`

Write a program to calculate the area of a circle using the formula:

```
Area of Circle = pi*(r*r)
```

- Declare the default parameter value of pi as 3.14 and radius as 1.

In [ ]:
```python
#Write a program to calculate the area of a circle using the formula.

def area_circle(pi=3.14,radius=1):
    area=pi*radius*radius
    print('radius=',radius)
    print(' The area of Circle = ',area)



area_circle()
area_circle(radius=5)
```

What will be the output of the following program?

In [ ]:
```python
def disp_values(a,b=10,c=20):
    print(' a = ',a,' b = ',b,'c= ',c)


disp_values(15) # 15,10,20
disp_values(50,b=30) # 50 30 20
disp_values(c=80,a=25,b=35)  #25 35 80
disp_values(80,25,35)  #80 25 35
```