# UNIT II

# FUNCTIONS

## FRUITFUL FUNCTIONS:

- Functions can take arguments and also they can return results. For example: math function will return result. The functions that return result are called **fruitful functions**.
- The functions, like print(), perform an action but don't return a value. These functions are called **void functions**.
- The result returned by a fruitful function is used in the program.
- Example:

  ***a = math.sqrt(5)***

  will give / display the result as: 2.2360679774997898
- When the variable 'a' is not used, then the returned value is lost forever.
- Void functions might display something on the screen or have some other effect, but they don't have a return value.
- When the result of void functions is stored in a variable, a special value called **None** will be returned.
- Example:

  ***result = print_twice('Bing')***
  ***print(result)***

  will give / display the result as: None

## VARIABLE SCOPE AND LIFETIME:

- In Python, variables cannot be accessed anytime anywhere.
- Each variable have two features as:
    - Scope - part of the program where the variable is accessible
    - Lifetime – duration for which the variables exist

**Local Variables:**

- A variable which is defined within a function is local to that function and it is called Local Variable.
- A local variable can be accessed from the point of its definition until the end of the function in which it is defined.
- It exists as long as the function is executing. Function parameters behave like local variables in the function.

**Global Variables:**

- Global variables are those variables which are defined in the main body of the program file.
- They are visible throughout the program file.
- Avoid the use of global variables because they may get altered by mistake and then result in erroneous output.
- To define a variable defined inside a function as global, the global statement is used.

```
var = "Good"
def show():
        global var1
        var1 = "Morning"
        print("In Function var is - ", var)
show()
print("Outside function, var1 is - ", var1)      #accessible as it is global
variable
print("var is - ", var)
```

**OUTPUT**

```
In Function var is -  Good
Outside function, var1 is -  Morning
var is -  Good
```

**Programming Tip:** All variables have the scope of the block.

- Example:

```
num1 = 10    # global variable
print("Global variable num1 = ", num1)
def func(num2):           # num2 is function parameter
        print("In Function - Local Variable num2 = ",num2)
        num3 = 30         #num3 is a local variable
        print("In Function - Local Variable num3 = ",num3)
func(20)      #20 is passed as an argument to the function
print("num1 again = ", num1)       #global variable is being accessed
#Error- local variable can't be used outside the function in which it is defined
print("num3 outside function = ", num3)
```

**OUTPUT**

```
Global variable num1 = 10
In Function - Local Variable num2 = 20
In Function - Local Variable num3 = 30
num1 again =  10
num3 outside function =
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 12, in <module>
    print("num3 outside function = ", num3)
NameError: name 'num3' is not defined
```

**Programming Tip:** Variables can only be used after the point of their declaration

**Local vs Global Variables:**

| LOCAL VARIABLES | GLOBAL VARIABLES |
|---|---|
| These variables are defined inside the function and it is local to that function. | These variables are defined outside / main body of the program. |
| These variables can be accessed only inside the function where it is defined. | These variables can be accessed anywhere inside the program. |
| They are not related to variables defined in same name outside the function. | They are accessible to all functions defined inside the program. |

## FUNCTION COMPOSITION:

- Function Composition refers to the act of calling one function from another.

- Example: Write a function that takes two points and computes the area of the circle. One point refers the center of the circle and another point refers a point on the perimeter.

```
import math
def distance(x1,y1,x2,y2):
    dis = ((x2-x1)**2 + (y2-y1)**2)**0.5
    return dis
def area(rad):
    a = math.pi*rad*rad
    return a
def areaofcircle(xc,yc,xp,yp):
    radius = distance(xc,yc,xp,yp)
    result = area(radius)
    return result
a1 = int(input("Enter the xc coordinate : "))
b1 = int(input("Enter the yc coordinate : "))
a2 = int(input("Enter the xp coordinate : "))
b2 = int(input("Enter the yp coordinate : "))
print("The area of the circle is: ", areaofcircle(a1,b1,a2,b2))
```

```
Enter the xc coordinate : 2
Enter the yc coordinate : 2
Enter the xp coordinate : 3
Enter the yp coordinate : 4
The area of the circle is:  15.707963267948967
```

**RECURSIVE FUNCTION:**

- A recursive function is defined as a function that calls itself again and again until the problem gets solved.
- Every recursive solution has two major cases as follows:
    - **Base case** – The problem is simple enough to be solved directly without making any further calls to the same function.
    - **Recursive case** - The problem is divided into simpler sub parts. Then, the function calls itself but with sub parts of the problem obtained earlier. Third, the result is obtained by combining the solutions of simpler sub parts.

**EXAMPLES OF RECURSIVE FUNCTION:**

- Example: To calculate the **factorial** of a number **using Recursion**

```
def factorial(n):
        fact = 1
        if (n==0):
                return 1
        else:
                fact = n*factorial(n-1)
        return fact
n = int(input("Enter the value of n:"))
f = factorial(n)
print("The factorial is: ", f)
```

```
Enter the value of n:6
The factorial is:  720
```

- Example: To calculate the **factorial** of a number **without using Recursion**

```
n = int(input("Enter the value of n:"))
fact = 1
if (n==0):
```

```
        print("The factorial is: ", fact)
    else:
        for i in range (1,n+1):
            fact=fact*i
        print("The factorial is: ", fact)
```

```
Enter the value of n:6
The factorial is:  720
```

- Example: To generate the **Fibonacci series** of a number **using Recursion**

```
def fibo(n):
    if (n<2):
        return 1
    return (fibo(n-1)+fibo(n-2))
print ("Enter the value of n: ")
n = int(input())
for i in range (n):
    print("Fibonacci Series: ",fibo(i))
```

```
Enter the value of n:
10
Fibonacci Series:  1
Fibonacci Series:  1
Fibonacci Series:  2
Fibonacci Series:  3
Fibonacci Series:  5
Fibonacci Series:  8
Fibonacci Series:  13
Fibonacci Series:  21
Fibonacci Series:  34
Fibonacci Series:  55
```

- Example: To generate the **Fibonacci series** of a number **without using Recursion**

```
print ("Enter the value of n: ")
n = int(input())
print ("Fibonacci series up to {0}".format(n))
f1 = 0
f2 = 1
print(f1)
print(f2)
while f1 < n:
    f1, f2 = f2, f1+f2
    print(f2)
```

```
Enter the value of n:
10
Fibonacci series up to 10
0
1
1
2
3
5
8
13
21
```

**ITERATION vs RECURSION:**

| RECURSION | ITERATION |
|---|---|
| Program code will be short and simple | Program code is large and complex |
| Follows a Divide and Conquer strategy. | Follows sequential processing. |
| More efficient than iteration | Less efficient |
| Difficult to find bugs | Bugs can be found easily. |

**STRINGS:**

- Strings are set of characters represented inside single, double or triple quotes.

- Python has a built-in string class named "str".
- The string can be defined and declared in any of the ways as specified below:

```
name = "India"
graduate = 'N'
country = name
nationality = str("Indian")
```

**INDEXING IN STRINGS:**

- Individual characters in a string are accessed using the subscript ([ ]) operator. The expression/value given in brackets is called an index.
- String index starts with 0 and ends with n – 1, where n is the number of characters in the string. Example: a = "Jansons", a[0] is 'J', a[6] = s.

**TRAVERSING A STRING:**

- A string can be traversed by accessing character(s) from one index to another.
- For example, the following program uses indexing to traverse a string from first character to the last.

```
message = "Hello!"
index = 0
for i in message:
print("message[", index, "] = ", i)
index += 1
```

**OUTPUT**

```
message[ 0 ] =  H
message[ 1 ] =  e
message[ 2 ] =  l
message[ 3 ] =  l
message[ 4 ] =  o
message[ 5 ] =  !
```

**CONCATENATING, APPENDING AND MULTIPLYING STRINGS:**

- Concatenating refers to joining two strings. In Python, two or more strings are concatenated using '+' operator.
- Append refers to adding a string to the end of other string. The operator used for appending the string is +=.
- A string can be multiplied (repeated) 'n number of times using * operator.

```
str1 = "Jansons "
str2 = 'Institute '
print("First string is: ", str1)
print("Second string is: ", str2)
#String Concatenation
print("Concatenated String is: ", str1 + str2)
#String Appending
str2+="of Technology"
print("Appended string is: ", str2)
#String Repeating
print("Repeated string1 is: ", str1 * 3)
```

```
First string is:  Jansons
Second string is:  Institute
Concatenated String is:  Jansons Institute
Appended string is:  Institute of Technology
Repeated string1 is:  Jansons Jansons Jansons
```

**STRINGS ARE IMMUTABLE:**

- Python strings are immutable which means that once they are created, they cannot be changed.

- Whenever an existing string is modified, a new string is created.

```
str1 = "Hello"
print("Str1 is : ", str1)
print("ID of str1 is : ", id(str1))

str2 = "World"
print("Str2 is : ", str2)
```

```
print("ID of str1 is : ", id(str2))
str1 += str2
print("Str1 after concatenation is : ", str1)
print("ID of str1 is : ", id(str1))
str3 = str1
print("str3 = ", str3)
print("ID of str3 is : ", id(str3))

OUTPUT
Str1 is :  Hello
ID of str1 is :  45093344
Str2 is :  World
ID of str1 is :  45093312
Str1 after concatenation is :  HelloWorld
ID of str1 is :  43861792
str3 =  HelloWorld
ID of str3 is :  43861792
```

**STRING FORMATTING OPERATOR:**

- The % operator takes a format string on the left and the corresponding values in a tuple .
- Format strings are %d, %s, %c and %f that act as placeholders for integers, strings, characters and floating point numbers respectively.
- The format operator, %, allow users to construct strings, replacing parts of the strings with the data stored in variables.
- The syntax for the string formatting operation is: **"<Format>" % (<Values>).**

```
name = "Aarish"
age = 8
print("Name = %s and Age = %d" %(name, age))
print("Name = %s and Age = %d" %("Anika", 6))
```

**OUTPUT**

```
Name = Aarish and Age = 8
Name = Anika and Age = 6
```

## BUILT – IN STRING METHODS AND FUNCTIONS:

- There are certain commonly used string methods that can be enlisted as below:

| Function | Usage | Example |
|---|---|---|
| `capitalize()` | This function is used to capitalize first letter of the string. | `str = "hello"`<br>`print(str.capitalize())`<br>**OUTPUT**<br>`Hello` |
| `center(width, fillchar)` | Returns a string with the original string centered to a total of `width` columns and filled with `fillchar` in columns that do not have characters. | `str = "hello"`<br>`print(str.center(10, '*'))`<br>**OUTPUT**<br>`**hello***` |
| `count(str, beg, end)` | Counts number of times str occurs in a string. You can specify beg as 0 and end as the length of the message to search the entire string or use any other value to just search a part of the string. | `str = "he"`<br>`message = "helloworldhellohello"`<br>`print(message. count (str,0, len (message)))`<br>**OUTPUT**<br>`3` |
| `endswith (suffix, beg, end)` | Checks if string ends with suffix; returns `True` if so and `False` otherwise. You can either set `beg = 0` and `end` equal to the length of the message to search entire string or use any other value to search a part of it. | `message = "She is my best friend"`<br>`print(message.endswith("end", 0,len(message)))`<br>**OUTPUT**<br>`True` |

| | | |
|---|---|---|
| `find(str, beg, end)` | Checks if `str` is present in string. If found it returns the position at which `str` occurs in string, otherwise returns `-1`. You can either set `beg = 0` and `end` equal to the length of the message to search entire string or use any other value to search a part of it. | `message = "She is my best friend"`<br>`print(message. find("my",0, len (message)))`<br>**OUTPUT**<br>`7` |
| `index(str, beg, end)` | Same as find but raises an exception if `str` is not found. | `message = "She is my best friend"`<br>`print(message.index("mine", 0, len(message)))`<br>**OUTPUT**<br>`ValueError: substring not found` |
| `rfind(str, beg, end)` | Same as find but starts searching from the end. | `str = "Is this your bag?"`<br>`print(str.rfind("is", 0, len(str)))`<br>**OUTPUT**<br>`5` |
| `rindex(str, beg, end)` | Same as `rindex` but start searching from the end and raises an exception if `str` is not found. | `str = "Is this your bag?"`<br>`print(str.rindex("you", 0, len(str)))`<br>**OUTPUT**<br>`8` |

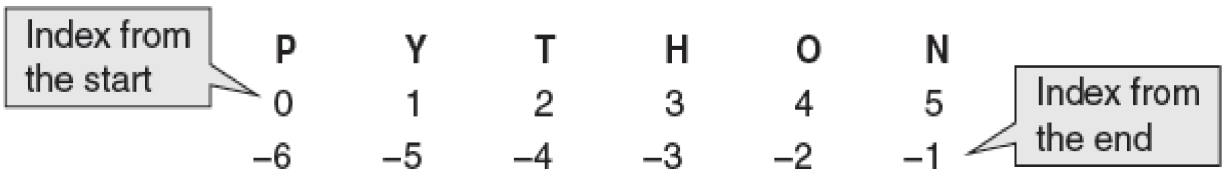
| | | |
|---|---|---|
| `isalnum()` | Returns True if string has at least 1 character and every character is either a number or an alphabet and False otherwise. | `message = "JamesBond007"`<br>`print(message.isalnum())`<br>**OUTPUT**<br>`True` |
| `isalpha()` | Returns True if string has at least 1 character and every character is an alphabet and False otherwise. | `message = "JamesBond007"`<br>`print(message.isalpha())`<br>**OUTPUT**<br>`False` |
| `isdigit()` | Returns True if string contains only digits and False otherwise. | `message = "007"`<br>`print(message.isdigit())`<br>**OUTPUT**<br>`True` |
| `islower()` | Returns True if string has at least 1 character and every character is a lowercase alphabet and False otherwise. | `message = "Hello"`<br>`print(message.islower())`<br>**OUTPUT**<br>`False` |
| `isspace()` | Returns True if string contains only whitespace characters and False otherwise. | `message = "  "`<br>`print(message.isspace())`<br>**OUTPUT**<br>`True` |
| `isupper()` | Returns True if string has at least 1 character and every character is an upper case alphabet and False otherwise. | `message = "HELLO"`<br>`print(message.isupper())`<br>**OUTPUT**<br>`True` |
| `len(string)` | Returns the length of the string. | `str = "Hello"`<br>`print(len(str))`<br>**OUTPUT**<br>`5` |
| `ljust(width[, fillchar])` | Returns a string left-justified to a total of `width` columns. Columns without characters are padded with the character specified in the `fillchar` argument. | `str = "Hello"`<br>`print(str.ljust(10, '*'))`<br>**OUTPUT**<br>`Hello*****` |
| `rjust(width[, fillchar])` | Returns a string right-justified to a total of `width` columns. Columns without characters are padded with the character specified in the `fillchar` argument. | `str = "Hello"`<br>`print(str.rjust(10, '*'))`<br>**OUTPUT**<br>`*****Hello` |

**STRING SLICING:**

- A substring of a string is called a **slice** and slicing refers to the sub part of the string.
- The subset of original string is taken by using [ ] operator. [ ] is known as slicing operator.
- The syntax of slice operation is

***str [start:end]***

- Start specifies the beginning index of the substring and end is the index of the last character.
- Indexing of a string starts with 0 and ends with n – 1.
- Example:




```
str = "PYTHON"
print("str[1:5] = ", str[1:5])   #characters starting at index 1 and extending up
to but not including index 5
print("str[:6] = ", str[:6])     # defaults to the start of the string
print("str[1:] = ", str[1:])      # defaults to the end of the string
print("str[:] = ", str[:])        # defaults to the entire string
print("str[1:20] = ", str[1:20])  # an index that is too big is truncated down to
length of the string
```

**OUTPUT**

```
str[1:5] =  YTHO
str[:6] =  PYTHON
str[1:] =  YTHON
str[:] =  PYTHON
str[1:20] =  YTHON
```

**Programming Tip:** Python does not have any separate data type for characters. They are represented as a single character string.

**FUNCTIONS – ord() AND chr():**

- ord() function returns the ASCII code of the character, while chr() function returns the character represented by a ASCII number.
- Example:

```
ch = 'R'
print(ord(ch))
```

**OUTPUT**

```
82
```

```
print(chr(82))
```

**OUTPUT**

```
R
```

```
print(chr(112))
```

**OUTPUT**

```
p
```

```
print(ord('p'))
```

**OUTPUT**

```
112
```

**COMPARING STRINGS:**

- Python allows to compare strings using the relational operators, <, >, <=, >=, ==, !=.
- These operators compare the strings based on the lexicographical order, i.e: based on ASCII values.
- Note: ASCII values of A – Z is 65 – 90, a – z is 97 – 122.

| Operator | Description | Example |
|---|---|---|
| == | If two strings are equal, it returns True. | >>> "AbC" == "AbC"<br>True |
| != or <> | If two strings are not equal, it returns True. | >>> "AbC" != "Abc"<br>True<br>>>> "abc" <> "ABC"<br>True |
| > | If the first string is greater than the second, it returns True. | >>> "abc" > "Abc"<br>True |
| < | If the second string is greater than the first, it returns True. | >>> "abC" < "abc"<br>True |
| >= | If the first string is greater than or equal to the second, it returns True. | >>> "aBC" >= "ABC"<br>True |
| <= | If the second string is greater than or equal to the first, it returns True. | >>> "ABc" <= "ABc"<br>True |

**ITERATING STRINGS:**

- String can be iterated using for loop and while loop. Example:

```
str = "Welcome to Python"
for i in str:
print(i, end=' ')
```

**OUTPUT**

```
W e l c o m e  t o  P y t h o n
```

```
message = " Welcome to Python "
index = 0
while index < len(message):
    letter = message[index]
    print(letter, end=' ')
    index += 1
```

**OUTPUT**

```
W e l c o m e   t o   P y t h o n
```

**THE STRING MODULE:**

- The string module consists of a number of useful constants, classes and functions that are used to manipulate strings.
- Some of the constants defined in the String module are:

| String Constants | Description |
|---|---|
| string.ascii_letters | Combination of ascii lower case and upper case letters. |
| string.ascii_lowercase | Refers to all lowercase letters a – z |
| string.ascii_uppercase | Refers to all lowercase letters A - Z |
| string.digits | Refers to digits from 0 - 9 |
| string.hexdigits | Refers to hexadecimal digits from 0 – 9, a – f, A – F |
| string.octdigits | Refers to octal digits from 0 – 7 |
| string.punctuation | String of ASCII characters that are considered to be punctuation characters. |
| string.lowercase | String of ASCII characters that are considered lowercase letters. |
| string.uppercase | String of ASCII characters that are considered uppercase letters. |

```
str = "Welcome to the world of Python"
print("Uppercase - ", str.upper())
print("Lowercase - ", str.lower())
print("Split - ", str.split())
print("Join - ", '-'.join(str.split()))
print("Replace - ", str.replace("Python", "Java"))
print("Count of o - ", str.count('o') )
print("Find of - ", str.find("of"))
```

**OUTPUT**

```
Uppercase -  WELCOME TO THE WORLD OF PYTHON
Lowercase -  welcome to the world of python
Split -  ['Welcome', 'to', 'the', 'world', 'of', 'Python']
Join -  Welcome-to-the-world-of-Python
Replace -  Welcome to the world of Java
Count of o -  5
Find of -  21
```

**Programming Tip:** A method is called by appending its name to the variable name using the period as a delimiter.

**2. GCD OF TWO NUMBERS:**

Program:

```
def gcd(x, y):
    if x > y:
        smallnum = y
    else:
        smallnum = x
    for i in range(1, smallnum+1):
        if((x % i == 0) and (y % i == 0)):
            gcd = i
    return gcd

print ("Enter two numbers:")
a=int(input())
b=int(input())
print("The GCD of {0} and {1} is : {2}".format(a,b,gcd(a, b)))
```

Output:

```
Enter two numbers:
16
28
The GCD of 16 and 28 is : 4
```

**3. EXPONENTIATION:**

Program:

```
def exponent(num,power):
    result=1
    if power==0:
        return 1
     for i in range(1,power+1):
        result=num*result
    return result
```

```
number=int(input("Enter the base number: "))
power=int(input("Enter the exponent number: "))
res=exponent(number,power)
print ("The value of %d to the power of %d is: %d" % (number, power, res))
```

Output:

```
Enter the base number: 25
Enter the exponent number: 3
The value of 25 to the power of 3 is: 15625
```

**Observation**

1. Write a function that takes two points and computes the area of the circle. One point refers the center of the circle and another point refers a point on the perimeter.
2. To calculate the factorial of a number using Recursion
3. To generate the Fibonacci series of a number using Recursion
4. Example using minimum 5 string built-in function
5. Write a program using function to find GCD OF TWO NUMBERS
6. Write a program using function to find EXPONENTIATION
7. Write a program using function to check whether two numbers are equal or not.

**Record**

1. **Write a program using function and return statement to check whether a number is even or odd.**
2. **Write a program using function to swap two numbers.**
3. **Write a program using function to calculate simple interest. Suppose the customer is a senior citizen. He is being offered 12 per cent rate of interest; for all other customers, the ROI is 10 per cent.**
4. **Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.**
5. **Write a program using function for Binary Search**
6. **Write a program using function for Linear Search**
7. **Write a program using function for Sorting (Any Algorithm)**