Python Programming
Unit 3
(KNC-302)

Prepared By

Abhishek Kesharwani

Assistant Professor,UCER Naini,Allahabad

UNIT III

- Function: Parts of A Function
- Execution of A Function
- Keyword and Default Arguments
- Scope Rules.
- Strings
- Length of the string and perform Concatenation and Repeat operations in it.
- Indexing and Slicing of Strings.

Python Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

Example

```
def my_function():
    print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_function():
    print("Hello from a function")
my_function()
```

Parameters

- Information can be passed to functions as parameter.
- Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

```
def my_function(fname):
  print(fname + " Welcome")

my_function("Abhishek")
```

Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function without parameter, it uses the default value:

Example

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
```

```
C:\Users\My Name>python demo_function_param2.py
I am from Sweden
I am from India
I am from Norway
```

Passing a List as a Parameter

- You can send any data types of parameter to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

Example

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

```
C:\Users\My Name>python demo_function_param3.py
apple
banana
cherry
```

Return Values

To let a function return a value, use the return statement:

Example

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Call by reference in Python

- In python, all the functions are called by reference, i.e., all the changes made to the reference inside the function revert back to the original value referred by the reference.
- However, there is an exception in the case of mutable objects since the changes made to the mutable objects like string do not revert to the original string rather, a new string object is made, and therefore the two different objects are printed.

Mutable vs Immutable Objects in Python

- Every variable in python holds an instance of an object.
- There are two types of objects in python i.e. **Mutable** and **Immutable objects**.
- Whenever an object is instantiated, it is assigned a unique object id.
- The type of the object is defined at the runtime and it can’t be changed afterwards. However, it’s state can be changed if it is a mutable object.

- NOTE: mutable objects can change their state or contents and immutable objects can’t change their state or content.

Immutable Objects : These are of in-built types like **int, float, bool, string, unicode, tuple**. In simple words, an immutable object can’t be changed after it is created.

Mutable Objects : These are of type **list , dict , set** . Custom classes are generally mutable.

Example 1 Passing mutable Object (List)

```
def change_list(l):
    l.append(20)
    l.append(30)
    print("list inside function = ",l)

list1 = [10,30,40,50]

change_list(list1)
print("list outside function = ",list1)
```

Output

list inside function = [10, 30, 40, 50, 20, 30]

list outside function = [10, 30, 40, 50, 20, 30]

Example 2 Passing immutable Object (String)

```
def change_string (str):
    str = str + " Hows you";
    print("printing the string inside function :",str);

string1 = "Hi I am there"



change_string(string1)

print("printing the string outside function :",string1)
```

Output

printing the string inside function : Hi I am there Hows you

printing the string outside function : Hi I am there

Types of arguments

There may be several types of arguments which can be passed at the time of function calling.

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

Required Arguments

- required arguments are those arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition.
- If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.

Example

```
def simple_interest(p,t,r):
    return (p*t*r)/100
p = float(input("Enter the principle amount? "))
r = float(input("Enter the rate of interest? "))
t = float(input("Enter the time in years? "))
print("Simple Interest: ",simple_interest(p,r,t))
```

Keyword arguments

- Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.
- The name of the arguments is treated as the keywords and matched in the function calling and definition.
- If the same match is found, the values of the arguments are copied in the function definition.

Example

```
def simple_interest(p,t,r):
    return (p*t*r)/100
print("Simple Interest:
   ",simple_interest(t=10,r=10,p=1900))
```

Output:

Simple Interest: 1900.0

Default Arguments

- Python allows us to initialize the arguments at the function definition.
- If the value of any of the argument is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.

Example

```
def printme(name,age=22):
    print("My name is",name,"and age is",age)
printme(name = "john")
```

Output:

My name is john and age is 22

Variable length Arguments

- Python provides us the flexibility to provide the comma separated values which are internally treated as tuples at the function call.
- However, at the function definition, we have to define the variable with * (star) as *<variable - name >.

Example

```
def printme(*names):
    print("type of passed argument is ",type(nam
    es))
    print("printing the passed arguments...")
    for name in names:
        print(name)
printme("john","David","smith","nick")
```

Output

```
type of passed argument is  <class 'tuple'>
printing the passed arguments...
john
David
smith
nick
```

Python Built-in Functions

- The Python built-in functions are defined as the functions whose functionality is pre-defined in Python.
- The python interpreter has several functions that are always present for use. These functions are known as Built-in Functions.

Python String

- In python, strings can be created by enclosing the character or the sequence of characters in the quotes.
- Python allows us to use single quotes, double quotes, or triple quotes to create the string.

str = "Hi Python !"

Here, if we check the type of the variable str using a python script

print(type(str)),

then it will **print** string (str).

python doesn't support the character data type instead a single character written as 'p' is treated as the string of length 1.

Strings indexing and splitting

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

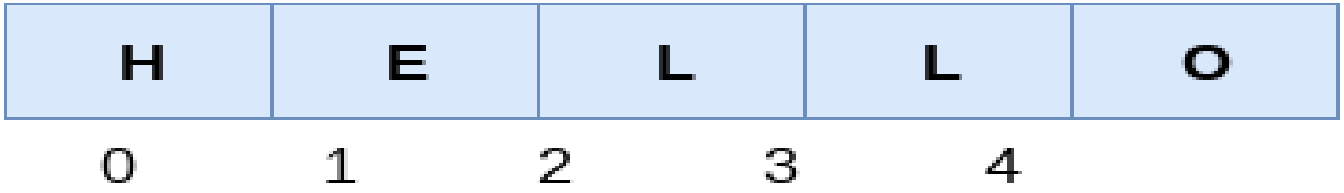
str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

str = "HELLO"



str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

str[:] = 'HELLO'

str[0:] = 'HELLO'

str[:5] = 'HELLO'

str[:3] = 'HEL'

str[0:2] = 'HE'

str[1:4] = 'ELL'

Reassigning strings

- Updating the content of the strings is as easy as assigning it to a new string.
- The string object doesn't support item assignment i.e., A string can only be replaced with a new string since its content can not be partially replaced.
- Strings are immutable in python.

Example 1

```
str = "HELLO"
str[0] = "h"
print(str)
```

Output:

```
TypeError: 'str' object does not support item
    assignment
```

Example 2

```
str = "HELLO"
print(str)
str = "hello"
print(str)
```

Output:

```
HELLO
hello
```

String Operators

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.

in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

Raw String

- s = 'Hi\nHello'
- print(s)

output

- Hi
- Hello

- raw_s = r'Hi\nHello'
- print(raw_s)

Output: Hi\nHello

Built-in String functions

- Python **capitalize()** method converts first character of the string into uppercase without altering the whole string.
- It changes the first character only and skips rest of the string unchanged.

Python Formatting operator

- Python allows us to use the format specifiers used in C's printf statement.
- The format specifiers in python are treated in the same way as they are treated in C.

Example

Integer = 10;

Float = 1.290

String = "Ayush"

print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am string ... My value is %s"%(Integer,Float,String));

Output:

Hi I am Integer ... My value is 10

Hi I am float ... My value is 1.290000

Hi I am string ... My value is Ayush

Python String Count() Method

- It returns the number of occurences of substring in the specified range.
- It takes three parameters, first is a substring, second a start index and third is last index of the range.
- Start and end both are optional whereas substring is required.

Example

```
str = "Hello Javatpoint"
str2 = str.count('t')
# Displaying result
print("occurences:", str2)
```

Output:

```
occurences: 2
```

Example

```
str = "ab bc ca de ed ad da ab bc ca"
oc = str.count('a', 3)
# Displaying result
print("occurences:", oc)
```

Output:

```
occurences: 5
```

Example

```
str = "ab bc ca de ed ad da ab bc ca 12 23 35 62”


oc = str.count('2')
# Displaying result
print("occurences:", oc)
```

Output:

```
occurences: 3
```

Python String endswith() Method

Python **endswith()** method returns true if the string ends with the specified substring, otherwise returns false.

Signature

endswith(suffix[, start[, end]])

Return Type

It returns a boolean value either True or False.

Example

```
str = "Hello this is United."
isends = str.endswith(".")
print(isends)
```

Output:

True

A simple example which returns true because it ends with dot (.).

Example

```
str = "Hello this is javatpoint."
isends = str.endswith("is",10)
# Displaying result
print(isends)
```

Output:

False

Note: Here, we are providing start index of the range from where method starts searching.

Example

```
str = "Hello this is javatpoint."
isends = str.endswith("is",0,13)
# Displaying result
print(isends)
```

Output:

True

Note: It returns true because third parameter stopped the method at index 13

Python String expandtabs() Method

- Python **expandstabs()** method replaces all the characters by sepecified spaces.
- By default a single tab expands to 8 spaces which can be overridden according to the requirement.
- We can pass 1, 2, 4 and more to the method which will replace tab by the these number of space characters.

Signature

```
expandtabs(tabsize=8)
```

tabsize : It is optional and default value is 8.

Example

```
str = "Welcome \t to \t the \t Javatpoint.“
str2 = str.expandtabs(1)
str3 = str.expandtabs(2)
str4 = str.expandtabs(4)
print(str2)
print(str3)
print(str4)
```

Output:

```
Welcome   to   the   Javatpoint.
Welcome    to    the   Javatpoint.
Welcome      to      the    Javatpoint.
```

Python String find() Method

Python **find()** method finds substring in the whole string and returns index of the first match. It returns -1 if substring does not match.

Signature

find(sub[, start[, end]])

Parameters

sub : substring

start : start index a range

end : last index of the range

Return Type

If found it returns index of the substring, otherwise -1.

Example

```
str = "Welcome to the Javatpoint."
str2 = str.find("the")

print(str2)
```

Output:

11

Python String format() Method

- Python **format()** method is used to perform format operations on string.
- While formatting string a delimiter {} (braces) is used to replace it with the value.
- This delimeter either can contain index or positional argument.

Example

```
str = "Java"
str2 = "C#"
str3 = "{} and {} both are programming language
    s".format(str,str2)
print(str3)
```

Output:

Java and C# both are programming languages

Example

```
val = 10
print("decimal: {0:d}".format(val));
# display decimal result
print("hex: {0:x}".format(val));      # display hexadecimal result
print("octal: {0:o}".format(val));    # display octal result
print("binary: {0:b}".format(val));   # display binary result
```

Output:

```
decimal: 10 hex:
octal: 12
binary: 1010
```

Example

```
val = 100000000
print("decimal: {:,}".format(val)); # formatting float value
print("decimal: {:%}".format(56/9));
print("decimal: {:.3%}".format(56/9));
```

Output

```
decimal: 100,000,000
decimal: 622.222222%
decimal: 622.222%
```

Python String isalnum() Method

Python **isalnum()** method checks whether the all characters of the string is alphanumeric or not. A character which is either a letter or a number is known as alphanumeric. It does not allow special chars even spaces.

Signature

- isalnum()

Parameters

- No parameter is required.

Return

- It returns either True or False.

Example

```
str = "Welcome123“
str3 = "Welcome 123"
str2 = str.isalnum()
str4 = str3.isalnum()
print(str2)
print(str4)
```

Output:

True

False

Example

```
str = "123"
str3 = "2.50"
str2 = str.isdecimal()
str4 = str3.isdecimal()
print(str2)
print(str4)
```

Output:

True

False

Python String islower() Method

Python string **islower()** method returns True if all characters in the string are in lowercase. It returns False if not in lowercase.

Signature

- islower()

Parameters

- No parameter is required.

Return

- It returns either True or False.

Example

```
str = "United College“
str2 = str.islower()
print(str2)
```

Output:

False

Example

str = **"hi, my contact is 876562...."**

str2 = str.islower()

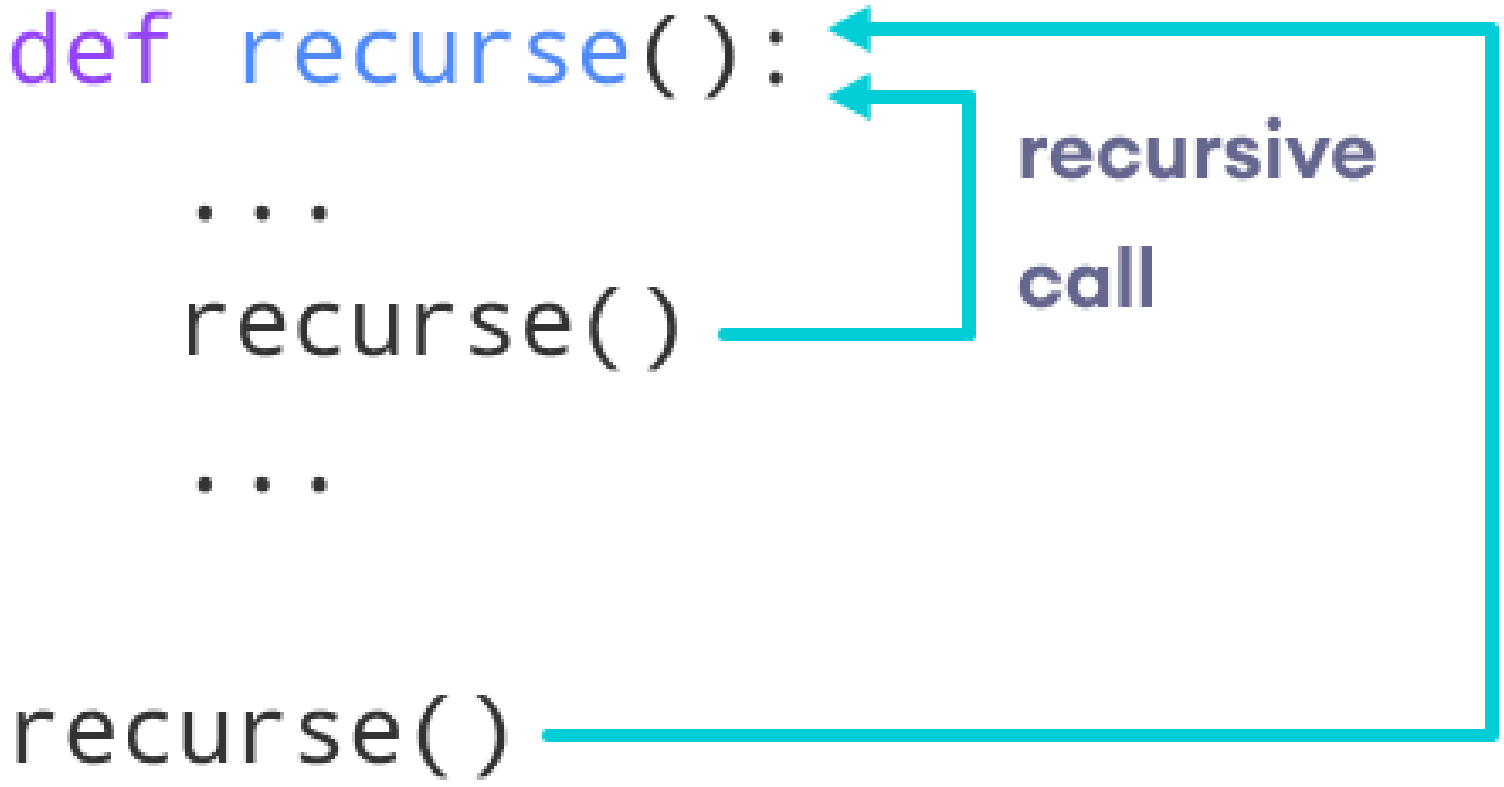
print(str2)

output

True

Recursive Function

- In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.


def recurse():
...
recurse()
...
recurse()
recursive
call

Example of a recursive function

```
def factorial(x):
    if x == 1 or x==0:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

Output

The factorial of 3 is 6

Advantages of Recursion

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

Fibonacci series

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$Fn = Fn-1 + Fn-2$$

with seed values $F0 = 0$ and $F1 = 1$.

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

Fibonacci Number in Python

```
def Fibonacci(n):
    if n < 0:
        print("Incorrect input")
        # First Fibonacci number is 0
    elif n == 1:
        return 0
    # Second Fibonacci number is 1
    elif n == 2:
        return 1
    else:
        return Fibonacci(n - 1) + Fibonacci(n -
2)

print(Fibonacci(int(input("Enter the
number"))))
```