What are strings?

In python, anything that you enclose between single or double quotation marks is considered a string. A string is essentially a sequence or array of textual data. Strings are used when working with Unicode characters.

Example:

```
name = "Ayush"
print("Hello, " + name)
```
Output: Hello, Ayush

```
print('He said, "I want to eat an apple".')
```
Multiline Strings:
If our string has multiple lines, we can create them like this:
```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```
Accessing Characters of a String:

In Python, string is like an array of characters. We can access parts of string by using its index which starts from 0.
Square brackets can be used to access elements of the string.

```
print(name[0])
print(name[1])
```
Looping through the string

We can loop through strings using a for loop like this:

```
for character in name:
    print(character)
```
Above code prints all the characters in the string name one by one!

String Slicing & Operations on String

Length of a String:

We can find the length of a string using len() function.

Example:

```
fruit = "Mango"
len1 = len(fruit)
```

```
print("Mango is a", len1, "letter word.")
```

Output:

Mango is a 5 letter word.

String as an array

A string is essentially a sequence of characters also called an array. Thus we can access the elements of this array.

Example:

```
pie = "ApplePie"
print(pie[:5])
print(pie[6])     #returns character at specified index
```

Output:

```
Apple
i
```

Note: This method of specifying the start and end index to specify a part of a string is called slicing.

Slicing Example:

```
pie = "ApplePie"
print(pie[:5])      #Slicing from Start
print(pie[5:])      #Slicing till End
print(pie[2:6])     #Slicing in between
print(pie[-8:])     #Slicing using negative index
```

Output:

```
Apple
Pie
pleP
ApplePie
```

Loop through a String:

Strings are arrays and arrays are iterable. Thus we can loop through strings.

Example:

```
alphabets = "ABCDE"
for i in alphabets:
    print(i)
```

Output:

```
A
B
C
D
E
```

String methods

Python provides a set of built-in methods that we can use to alter and modify the strings.

1. upper() :

The upper() method converts a string to upper case.

Example:

```
str1 = "AbcDEfghIJ"
print(str1.upper())
```

Output:

```
ABCDEFGHIJ
```

2. lower()

The lower() method converts a string to lower case.

Example:

```
str1 = "AbcDEfghIJ"
print(str1.lower())
```

Output:

```
Abcdefghij
```

3. strip() :

The strip() method removes any white spaces before and after the string.

Example:

```
str2 = "     Ayush Spoon     "
print(str2.strip)
```

Output:

```
Ayush Spoon
```

4. rstrip() :

The rstrip() removes any trailing characters in right side of string.

Example:

```
str3 = "Hello !!!"
print(str3.rstrip("!"))
```

Output:

```
Hello
```

5. replace() :

The replace() method replaces all occurences of a string with another string. Example:

```
str2 = "Silver Spoon"
print(str2.replace("Sp", "M"))
```

Output:

```
Silver Moon
```

6. split() :

The split() method splits the given string at the specified instance and returns the separated strings as list items.

Example:

```
str2 = "Silver Spoon"
print(str2.split(" "))       #Splits the string at the
whitespace " ".
```

Output:

['Silver', 'Spoon']

7. capitalize() :

The capitalize() method turns only the first character of the string to uppercase and the rest other characters of the string are turned to lowercase. The string has no effect if the first character is already uppercase.

Example:

```
str1 = "hello"
capStr1 = str1.capitalize()
print(capStr1)
str2 = "hello WorlD"
capStr2 = str2.capitalize()
print(capStr2)
```

Output:

```
Hello
Hello world
```

8. center() :

The center() method aligns the string to the center as per the parameters given by the user.

Example:

```
str1 = "Welcome to the Console!!!"
print(str1.center(50))
```

Output:

```
        Welcome to the Console!!!
```

We can also provide padding character. It will fill the rest of the fill characters provided by the user.

Example:

```
str1 = "Welcome to the Console!!!"
print(str1.center(50, "."))
```

Output:

```
............Welcome to the Console!!!............
```

9. count() :

The count() method returns the number of times the given value has occurred within the given string.

Example:

```
str2 = "Abracadabra"
countStr = str2.count("a")
print(countStr)
```

Output:

```
4
```

10. endswith() :

The endswith() method checks if the string ends with a given value. If yes then return True, else return False.

Example :
```
str1 = "Welcome to the Console !!!"
print(str1.endswith("!!!"))
```
Output:
```
True
```
We can even also check for a value in-between the string by providing start and end index positions.

Example:
```
str1 = "Welcome to the Console !!!"
print(str1.endswith("to", 4, 10))
```
Output:
```
True
```

11. find() :

The find() method searches for the first occurrence of the given value and returns the index where it is present. If given value is absent from the string then return -1.

Example:
```
str1 = "He's name is Dan. He is an honest man."
print(str1.find("is"))
```
Output:
```
10
```
As we can see, this method is somewhat similar to the index() method. The major difference being that index() raises an exception if value is absent whereas find() does not.

Example:
```
str1 = "He's name is Dan. He is an honest man."
print(str1.find("Daniel"))
```
Output:
```
-1
```

12.index() :

The index() method searches for the first occurrence of the given value and returns the index where it is present. If given value is absent from the string then raise an exception.

Example:
```
str1 = "He's name is Dan. Dan is an honest man."
print(str1.index("Dan"))
```
Output:
```
13
```
As we can see, this method is somewhat similar to the find() method. The major difference being that index() raises an exception if value is absent whereas find() does not.

Example:
```
str1 = "He's name is Dan. Dan is an honest man."
print(str1.index("Daniel"))
```
Output:
```
ValueError: substring not found
```

13.isalnum() :

The isalnum() method returns True only if the entire string only consists of A-Z, a-z, 0-9. If any other characters or punctuations are present, then it returns False.

Example 1:
```
str1 = "WelcomeToTheConsole"
print(str1.isalnum())
```
Output:

True

14.isalpha() :

The isalnum() method returns True only if the entire string only consists of A-Z, a-z. If any other characters or punctuations or numbers(0-9) are present, then it returns False.

Example :

```
str1 = "Welcome"
print(str1.isalpha())
```

Output:

True

15.islower() :

The islower() method returns True if all the characters in the string are lower case, else it returns False.

Example:

```
str1 = "hello world"
print(str1.islower())
```

Output:

True

16.isprintable() :

The isprintable() method returns True if all the values within the given string are printable, if not, then return False.

Example :

```
str1 = "We wish you a Merry Christmas"
print(str1.isprintable())
```

Output:

True

17.isspace() :

The isspace() method returns True only and only if the string contains white spaces, else returns False.

Example:

```
str1 = "        "       #using Spacebar
print(str1.isspace())
str2 = "        "       #using Tab
print(str2.isspace())
```

Output:

True
True

18.istitle() :

The istitile() returns True only if the first letter of each word of the string is capitalized, else it returns False.

Example:

```
str1 = "World Health Organization"
print(str1.istitle())
```

Output:

True

Example:

```
str2 = "To kill a Mocking bird"
print(str2.istitle())
```

Output:

```
False
```

19.isupper() :

The isupper() method returns True if all the characters in the string are upper case, else it returns False.

Example :

```
str1 = "WORLD HEALTH ORGANIZATION"
print(str1.isupper())
```

Output:

```
True
```

20.startswith() :

The endswith() method checks if the string starts with a given value. If yes then return True, else return False.

Example :

```
str1 = "Python is a Interpreted Language"
print(str1.startswith("Python"))
```

Output:

```
True
```

21.swapcase() :

The swapcase() method changes the character casing of the string. Upper case are converted to lower case and lower case to upper case.

Example:

```
str1 = "Python is a Interpreted Language"
print(str1.swapcase())
```

Output:

```
pYTHON IS A iNTERPRETED lANGUAGE
```

22.title() :

The title() method capitalizes each letter of the word within the string.

Example:

```
str1 = "He's name is Dan. Dan is an honest man."
print(str1.title())
```

Output:

```
He'S Name Is Dan. Dan Is An Honest Man.
```

if-else Statements

if-else is used in Python to make decisions in your program.
It helps your program take different actions based on conditions (True/False).
Based on this, the conditional statements are further classified into following types:

- if
- if-else
- if-else-elif
- nested if-else-elif.

Structure:

```
if condition:
    # code to run if condition is True
else:
    # code to run if condition is False
```

Example1:

```
age = 21
age_normal = 18
if (age> age_normal):
    print("You can drive car.")
else:
    print("You can not drive car.")
```

Output:

You can not drive car.

Example2:

```
marks = 75

if marks >= 90:
    print("Grade: A")
elif marks >= 75:
    print("Grade: B")
elif marks >= 60:
    print("Grade: C")
else:
    print("Fail")
```

Output:

```
Grade: B
```

Match case statement

match-case is used to check **multiple conditions** (like switch-case in C, C++, Java).
It was introduced in **Python 3.10** and later versions

Important Points:

- Use `match-case` only in **Python 3.10 or newer**.
- `case _:` works like the **default** case.
- You can use `|` to check **multiple values** in a single case.
- Use `case _ if condition:` for **conditional matching**.

Example1:

```
fruit = "apple"
match fruit:
    case "apple":
        print("It's
red or green.")
    case "banana":
        print("It's
yellow.")
    case _:

print("Unknown
fruit.")
```

Example2:

```
value = 2
match value:
    case 1 | 2 | 3:

print("Between 1 and
3")
    case 4:
        print("It's
four")
    case _:
    print("Something
else")
```

Example3:

```
number = 10
match number:
    case _ if number %
2 == 0:
        print("Even
number")
    case _:
        print("Odd
number")
```

Introduction to Loops

Sometimes a programmer wants to execute a group of statements a certain number of times. This can be done using loops. Based on this loops are further classified into following main types;

- for loop
- while loop

The for Loop

for loops can iterate over a sequence of iterable objects in python. Iterating over a sequence is nothing but iterating over strings, lists, tuples, sets and dictionaries.

Example1: Iterating over a string:

```
name = 'Abhishek'
for i in name:
    print(i, end=", ")
```

Output:

```
A, b, h, i, s, h, e, k,
```

Notes: Similarly, we can use loops for lists, sets and dictionaries.

Example2: Iterating over a list:

```
colors = ["Red",
"Green","Blue","Yellow"]
for x in colors:
    print(x)
```

Output:

```
Red
Green
Blue
Yellow
```

range():

What if we do not want to iterate over a sequence? What if we want to use for loop for a specific number of times?
Here, we can use the range() function.

Example:

```
for k in range(5):
```

Example3:

```
for k in range(4,9):
```

`    print(k)` **Output:** 0 1 2 3 4	`    print(k)` **Output:** 4 5 6 7 8

Break and continue Statements in Python

Python provides two important statements to control the flow of loops:

1. break Statement

- The `break` statement **terminates the loop immediately**, even if the loop condition is still true.
- It is generally used when we want to **exit a loop early**, based on a condition.

Example:

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```

Output:

```
1
2
3
4
```

When `i` becomes 5, the `break` statement is executed, and the loop stops.

2. continue Statement

- The `continue` statement **skips the current iteration** and moves to the **next iteration** of the loop.
- It is used when we want to **ignore some values or conditions** but still continue the loop.

Example:

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

Output:

```
1
2
4
5
```

What is a Function in Python?

A **function** is a reusable block of code that performs a specific task. It helps make the code modular and reduces repetition.

Types of Functions

1. **Built-in Functions** – Already provided by Python (e.g. `print()`, `len()`, `range()`)
2. **User-defined Functions** – Created by the user to perform custom tasks

Defining a Function

```
def function_name(parameters):
    # code block
    return result
```

`def`: Keyword to define a function
`parameters`: Optional values the function can receive
`return`: Sends a value back to where the function was called

Example:

```
def greet(name):
    return "Hello, " + name
print(greet("Amit"))
# Output: Hello, Amit
```

Calling a Function

You "call" a function by using its name followed by parentheses:

```
function_name(arguments)
```

Parameters vs Arguments

- **Parameter**: Variable in the function definition
- **Argument**: Actual value passed during the function call

Function with Default Parameters

```
def greet(name="Guest"):
    return "Hello, " + name

print(greet())          # Output: Hello, Guest
print(greet("Riya"))    # Output: Hello, Riya
```

Return Statement

Used to send the output from a function:

```
def add(a, b):
    return a + b
```

Function Without Parameters

```
def show_message():
    print("Welcome to Python!")
show_message()
```

List in Python

Lists are ordered collection of data items.
They store multiple items in a single variable.
List items are separated by commas and enclosed within square brackets [].
Lists are changeable meaning we can alter them after creation.

Example 1:

```
lst1 = [1,2,2,3,5,4,6]
lst2 = ["Red", "Green", "Blue"]
print(lst1)
print(lst2)
```

Output:

```
[1, 2, 2, 3, 5, 4, 6]
['Red', 'Green', 'Blue']
```

Example 2:

```
details = ["Abhijeet", 18, "FYBScIT", 9.8]
print(details)
```

Output:

```
['Abhijeet', 18, 'FYBScIT', 9.8]
```

Basics of List Indexing

Indexes start from 0 (not 1).
You use square brackets [] to access elements.

```
my_list = ['apple', 'banana', 'cherry']
print(my_list[0])  # Output: 'apple'
print(my_list[1])  # Output: 'banana'
print(my_list[2])  # Output: 'cherry'
```

Negative Indexing

Negative numbers count from the end of the list.

```
print(my_list[-1])  # 'cherry'
print(my_list[-2])  # 'banana'
print(my_list[-3])  # 'apple'
```

IndexError

Trying to access an index that doesn't exist will raise an error:

```
print(my_list[5])  # IndexError: list index out of range
```

Modifying List Elements Using Index

```
my_list[1] = 'mango'
print(my_list)  # ['apple', 'mango', 'cherry']
```

Using Index in Loops

```
for i in range(len(my_list)):
    print(f"Element at index {i} is {my_list[i]}")
```

List Slicing (a related concept)

```
print(my_list[0:2])   # ['apple', 'mango'] — from index 0 to 1
print(my_list[:])     # full list
print(my_list[::2])   # skip every other element
```

List Comprehension

List comprehensions are used for creating new lists from other iterables like lists, tuples, dictionaries, sets, and even in arrays and strings.

Syntax:

```
List = [Expression(item) for item in iterable if Condition]
```

Expression: It is the item which is being iterated.

Iterable: It can be list, tuples, dictionaries, sets, and even in arrays and strings.

Condition: Condition checks if the item should be added to the new list or not.

Example 1: Accepts items with the small letter "o" in the new list

```
names = ["Milo", "Sarah", "Bruno", "Anastasia", "Rosa"]
```

```
namesWith_O = [item for item in names if "o" in item]
print(namesWith_O)
```

Output: `['Milo', 'Bruno', 'Rosa']`

Example 2: Accepts items which have more than 4 letters

```
names = ["Milo", "Sarah", "Bruno", "Anastasia", "Rosa"]
namesWith_O = [item for item in names if (len(item) > 4)]
print(namesWith_O)
```

Output: ['Sarah', 'Bruno', 'Anastasia']

List Methods

1.sort()

This method sorts the list in ascending order. The original list is updated

Example 1:

```
colors = ["voilet", "indigo", "blue", "green"]
colors.sort()
print(colors)
num = [4,2,5,3,6,1,2,1,2,8,9,7]
num.sort()
print(num)
```

Output:

```
['blue', 'green', 'indigo', 'voilet']
[1, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.What if you want to print the list in descending order?

We must give reverse=True as a parameter in the sort method.

Example:

```
colors = ["voilet", "indigo", "blue", "green"]
colors.sort(reverse=True)
print(colors)
num = [4,2,5,3,6,1,2,1,2,8,9,7]
num.sort(reverse=True)
print(num)
```

Output:

```
['voilet', 'indigo', 'green', 'blue']
[9, 8, 7, 6, 5, 4, 3, 2, 2, 2, 1, 1]
```

Note : The reverse parameter is set to False by default.

3.reverse()

Example:

```
colors = ["voilet", "indigo", "blue", "green"]
colors.reverse()
print(colors)
num = [4,2,5,3,6,1,2,1,2,8,9,7]
num.reverse()
print(num)
```

Output:

```
['green', 'blue', 'indigo', 'voilet']
[7, 9, 8, 2, 1, 2, 1, 6, 3, 5, 2, 4]
```

4.index()

This method returns the index of the first occurrence of the list item.

Example:

```
colors = ["voilet", "green", "indigo", "blue", "green"]
print(colors.index("green"))

num = [4,2,5,3,6,1,2,1,3,2,8,9,7]
print(num.index(3))
```

Output:

```
1
3
```

5.count()

Returns the count of the number of items with the given value.

Example:

```
colors = ["voilet", "green", "indigo", "blue", "green"]
print(colors.count("green"))
```

Output:

```
2
```

6.copy()

Returns copy of the list. This can be done to perform operations on the list without modifying the original list.

Example:

```
colors = ["voilet", "green", "indigo", "blue"]
newlist = colors.copy()
print(colors)
print(newlist)
```

Output:

```
['voilet', 'green', 'indigo', 'blue']
['voilet', 'green', 'indigo', 'blue']
```

7.append():

This method appends items to the end of the existing list.

Example:

```
colors = ["voilet", "indigo", "blue"]
colors.append("green")
print(colors)
```

Output:

```
['voilet', 'indigo', 'blue', 'green']
```

8.insert():

This method inserts an item at the given index. User has to specify index and the item to be inserted within the insert() method.

Example:

```
colors = ["voilet", "indigo", "blue"]
#           [0]         [1]       [2]

colors.insert(1, "green")    #inserts item at index 1
# updated list: colors = ["voilet", "green", "indigo", "blue"]
#        indexs                [0]       [1]      [2]       [3]

print(colors)
```

Output:

```
['voilet', 'green', 'indigo', 'blue']
```

9.extend():

This method adds an entire list or any other collection datatype (set, tuple, dictionary) to the existing list.

Example:

```
#add a list to a list
colors = ["voilet", "indigo", "blue"]
rainbow = ["green", "yellow", "orange", "red"]
colors.extend(rainbow)
print(colors)
```

Output:

```
['voilet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red']
```

10.Concatenating two lists:

You can simply concatenate two lists to join two lists.

Example:

```
colors = ["voilet", "indigo", "blue", "green"]
colors2 = ["yellow", "orange", "red"]
print(colors + colors2)
```

Output:

['voilet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red']

String Formatting in Python

String formatting is a way to **insert variables or values** into strings.

1. Using format() method (Python 3 and above)

```
name = "Alice"
age = 25
print("My name is {} and I am {} years old.".format(name,
age))
```

With index numbers:

```
print("My name is {0} and I am {1} years old. {0} is learning
Python.".format(name, age))
```

With named placeholders:

```
print("My name is {n} and I am {a} years
old.".format(n="Alice", a=25))
```

2. Using f-strings (Python 3.6+)

```
name = "Bob"
age = 30
print(f"My name is {name} and I am {age} years old.")
```

You can use expressions inside f-strings:

```
print(f"Next year I will be {age + 1} years old.")
```

3. Using % operator (Old method, still supported)

```
name = "Charlie"
age = 28
print("My name is %s and I am %d years old." % (name, age))
```