**UNIT 3 PART-I STRINGS**

**STRINGS**

**STRINGS : Accessing Characters and Substrings in a String, String Methods, Basic String operations, String Slicing, Testing, Searching, Comparing and manipulating Strings.**

**STRINGS:**

**Defination of a String** : Astring is a sequence of characters **.** String is an immutable sequence data type .Python stringis a sequence of Unicode characters that is enclosed in the quotations marks.

A string can be either created by using single quotes (' ') or double quotes(" "). Python treats single quotes the same as double quotes.

**Creating string using single quotes and double qoutes**

A string can be created using either single quotes or double quotes .However,if your string **contains double** quotes use **single quotes**. And if your string contains **single quotes** use **double quotes.**

SYNTAX :strname='text' or strname="text"

example :

1)str='Hii everyone!Welcome to python'#strname='text'

print(str) # string using single quotes

**Output:**

Hii everyone!Welcome to python

2)str="Hii everyone!Welcome to python" # strname="text"

print(str) # string using double quotes

**Output:**

Hii everyone!Welcome to python

3)# if your string contains double quotes use single quotes

str='Hii everyone!"Welcome to python" ' # if your string contains double quotes use single quotes

print(str) # string using single quotes strname='text’

**Output:**

Hii everyone!"Welcome to python"

4)# if your string contains single quotes use double quotes

str="Hii everyone!'Welcome to python' " # if your string contains single quotes use double quotes

print(str) # string using double quotes strname="text"

**Output:**

Hii everyone!'Welcome to python'

**Multiline strings**

In python,it is possible to have a string that spans multiple lines.

To create a multiline string,surround or enclose a string with **triple single quotes**(''' ''')or **triple double quotes**(""" """).

Example :

1)#using single triple quotes

str='''hii

hello

world'''

print(str) # multilinestrings using single triple quotes str='''text'''

**Output:**

hii

hello

world

**ACCESSING CHARACTERS AND SUBSTRINGS IN A STRING**

In Python, individual characters of a String can be accessed by using the **method of Indexing**. Indexing allows negative address references too to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on. This negative indexing is called as backward indexing. The positive indexing is called as forward indexing,example 0 represents the first character , 1 represents the second character, 2 represents the third chaqrcter and so on.

**Indexing**

To access a single character, use indexing.

Indexing uses **square brackets ([ ]**) to access characters.

0 represents the first character , 1 represents the second character of a string and so on.

While accessing an index out of the range will cause an **IndexError**. Only Integers are allowed to be passed as an index, float or other types will cause a **TypeError.**

Example :

1. #forward indexing

str="hi everyone"

print(str[0]) # prints h

print(str[1]) # prints i

**Output:**

h

i

while -1 represents the last character,and -2 represents the second to the last character .

2) #backward indexing

str="hi everyone"

print(str[-1]) # prints e

print(str[-2]) # prints n

**Output:**

**e**

**n**

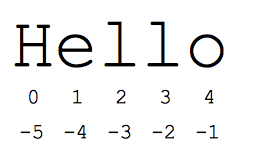


fig1.INDEXINGforward indexing (positive numbers) & backward indexing(negative numbers).

**STRING METHODS**

Python has a set of built-in methods that you can use on strings.

To avail the string method or string function is **stringname.stringfunction()**

**Ex:str.capitalize() , str.lower() , str.upper() , str.title() , str.isdigit(),………etc.**

**Note:** All string methods returns new values. They do not change the original string.

1. **Capitalize String**

The **capitalize()** function returns the string where the first letter is in uppercase.

Example

# capitalize method or capitalize() function

str='hi everyone !welcome to python'

x=str.capitalize()

print(str) # the string remains the same

print(x) # the first letter of the sting converts into capital

**Output:**

hi everyone !welcome to python

Hi everyone !welcome to python

1. **Upper() function**

Convert to upper case

The upper() function returns a copy of the given string but all the letters are in upper case.

Example:

# upper() function

str='hi everyone !welcome to python'

x=str.upper()

print(str) # the string remains the same

print(x) # the sting converts into upper case

**Output:**

hi everyone !welcome to python

HI EVERYONE !WELCOME TO PYTHON

1. **Lower() function**

Convert to lower case

The **lower()** function returns a copy of the given string but all letters are in lower case.

Example

1)# lower() function

str='HI EVERYONE!WELCOE TO PYTHON'

x=str.lower()

print(str) # the string remains the same

print(x) # the sting converts into lower case

**Output:**

HI EVERYONE!WELCOE TO PYTHON

hieveryone!welcoe to python

2)# lower() function

str='Hi Everyone!Welcome To Python'

x=str.lower()

print(str) # the string remains the same

print(x) # the sting converts into lower case

**Output:**

Hi Everyone!Welcome To Python

hieveryone!welcome to python

1. **len() function**

Get the length of the string.

The length of a string is the number of characters it contains.

The len() function returns the length of a string.

It takes one parameter,the string.

**Example**

1)#length of the string

str='Hi Everyone!Welcome To Python'

print(len(str))

**output:**

29

2) #length of the string

str='Hi Everyone!Welcome To Python'

x=len(str)

print("the length of the string is ",x)

**Output:**

the length of the string is 29

1. **replace() function**

Replacing parts of the string

The **replace() function** replaces the occurrences of a specified substring with another substring.

**Syntax:**

string.replace(old,new)

* old- the substring to be replaced,it is case-sensitive.
* new-the new substring that will replace the old substring.

**Note: Old substring is case-sensitive**

**1)**#replace() function

str="Hello World!"

x=str.replace("World","Everyone")

print(str) # before replacing

print(x) # after replacing

**Output:**

Hello World!

Hello Everyone!

1. Now , in this example world will not be replaced with Everyone because the first letter of the world is not in upper case , to replace the old substring with new substring the first letter of the old substring should be in uppercase i.e World.

**Example**

#replace() function

str="Hello World!"

x=str.replace("world","Everyone")

print(str) #before replacing

print(x) #the first letter of the old substring is not in upper case

#so the new substring is not replaced with old substring

**Output:**

Hello World!

Hello World!

1. The replace() function replaces all occurences of the old substring with the new substring.

**Example**

str="Hello World! I Love World! World is amazing!"

x=str.replace("World","Everyone")

print(x)

**Output:**

Hello Everyone! I Love Everyone! Everyone is amazing!

4)

But you can specify how many occurences you would like to be replaced, on the third argument.

In this example,onlytwo occurences of the substring **World** will be replaced with Everyone.

**Example**

#replace() function replaces all occurences of the old substring with the new substring

str="Hello World! I Love World! World is amazing!"

x=str.replace("World","Everyone",2)

print(str)

print(x)

**Output:**

Hello World! I Love World! World is amazing!

Hello Everyone! I Love Everyone! World is amazing!

1. **Check if a value is present in a string**

* To check if a substring is present in a string,use the in keyword.

It returns True if the substring is found .

Note that evaluation is case-senstive .

**Example:**

str="python is fun to learn"

x="learn" in str

print(x)

**Output:**

True

Alternatively, you can use the not inkeyword ,it returns True if the substring is not found .

Example :

str="python is fun to learn"

x="world" not in str #it returns true if the substring is not found in the string

print(x)

**Output:**

True

1. **find() Method**

\*The find() method finds the first occurrence of the specified value.

\*The find() method returns -1 if the value is not found.

\*The find() method is almost the same as the [index()](https://www.w3schools.com/python/ref_string_index.asp) method, the only difference is that **2)**

the index() method raises an exception if the value is not found.

1)# The find() method finds the first occurrence of the specified value

# find() method or find() function

str="Hello, welcome to my world."

x=str.find("Hello")

print(x)

**Output:**

0

# The find() method returns -1 if the value is not found.

# find() method or find() function

str="Hello, welcome to my world."

x=str.find("Hi")

print(x)

**Output:**

-1

1. **count() method**

The count() method returns the number of times a specified value appears in the string.

#count() method

str="python is interpreted ,python is interactive and python is object-orirented "

x=str.count("python")

print(x)

**Output:**

3

1. **index() Method**

**#index() method**

1)str="Hello, welcome to my world."

x=str.index("welcome")

print(x)

**Output:**

7

2)txt="Hello, welcome to my world."

x =txt.index("e")

print(x)

Output:

1

# 10)isalpha() Method

The isalpha() method returns True if all the characters are alphabet letters (a-z).

Example of characters that are not alphabet letters: (space)!#%&? etc.

**1)**#isaplpha() method

str="hiiipython"

x=str.isalpha()

print(x)

**Output:**

True

**2)**#isaplpha() method

str="hiii python"

x=str.isalpha()

print(x)

**Output:**

False

**3)**#isaplpha() method

str="hiii python3@2020"

x=str.isalpha()

print(x)

**Output:**

False

**11)**

# isdigit() Method

The isdigit() method returns True if all the characters are digits, otherwise False.

Exponents, like ², are also considered to be a digit.

# Example:

# #isdigit() method

# str="59111321"

# x=str.isdigit()

# print(x) # True

**Output:**

True

# 12) title() Method

The title() method returns a string where **the first character in every word is upper case**. Like a header, or a title.

If the word contains a number or a symbol, the first letter after that will be converted to upper case.

#title() method

str="hii world! welcome to python!enjoy the world of python!"

x=str.title()

print(str) #original string

print(x) #string after title() method

**Output:**

hii world! welcome to python!enjoy the world of python!

Hii World! Welcome To Python!Enjoy The World Of Python!

# 13)swapcase() Method

The swapcase() method returns a string where all the upper case letters are lower case and vice versa.

# Example:

# 1)#swapcase() method

# str="Hiii Python"

# x=str.swapcase()

# print(x)

# Output:

# hIIIpYTHON

# 2)#swapcase() method

# str="Hello World"

# x=str.swapcase()

# print(x)

# Output:

# hELLOwORLD

# These are some string methods in detail.There are many string methods or string functions and they are as follows

**Note:** All string methods returns new values. They do not change the original string.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Converts the first character to upper case |
| [casefold()](https://www.w3schools.com/python/ref_string_casefold.asp) | Converts string into lower case |
| [center()](https://www.w3schools.com/python/ref_string_center.asp) | Returns a centered string |
| [count()](https://www.w3schools.com/python/ref_string_count.asp) | Returns the number of times a specified value occurs in a string |
| [encode()](https://www.w3schools.com/python/ref_string_encode.asp) | Returns an encoded version of the string |
| [endswith()](https://www.w3schools.com/python/ref_string_endswith.asp) | Returns true if the string ends with the specified value |
| [expandtabs()](https://www.w3schools.com/python/ref_string_expandtabs.asp) | Sets the tab size of the string |
| [find()](https://www.w3schools.com/python/ref_string_find.asp) | Searches the string for a specified value and returns the position of where it was found |
| [format()](https://www.w3schools.com/python/ref_string_format.asp) | Formats specified values in a string |
| format\_map() | Formats specified values in a string |
| [index()](https://www.w3schools.com/python/ref_string_index.asp) | Searches the string for a specified value and returns the position of where it was found |
| [isalnum()](https://www.w3schools.com/python/ref_string_isalnum.asp) | Returns True if all characters in the string are alphanumeric |
| [isalpha()](https://www.w3schools.com/python/ref_string_isalpha.asp) | Returns True if all characters in the string are in the alphabet |
| [isascii()](https://www.w3schools.com/python/ref_string_isascii.asp) | Returns True if all characters in the string are ascii characters |
| [isdecimal()](https://www.w3schools.com/python/ref_string_isdecimal.asp) | Returns True if all characters in the string are decimals |
| [isdigit()](https://www.w3schools.com/python/ref_string_isdigit.asp) | Returns True if all characters in the string are digits |
| [isidentifier()](https://www.w3schools.com/python/ref_string_isidentifier.asp) | Returns True if the string is an identifier |
| [islower()](https://www.w3schools.com/python/ref_string_islower.asp) | Returns True if all characters in the string are lower case |
| [isnumeric()](https://www.w3schools.com/python/ref_string_isnumeric.asp) | Returns True if all characters in the string are numeric |
| [isprintable()](https://www.w3schools.com/python/ref_string_isprintable.asp) | Returns True if all characters in the string are printable |
| [isspace()](https://www.w3schools.com/python/ref_string_isspace.asp) | Returns True if all characters in the string are whitespaces |
| [istitle()](https://www.w3schools.com/python/ref_string_istitle.asp) | Returns True if the string follows the rules of a title |
| [isupper()](https://www.w3schools.com/python/ref_string_isupper.asp) | Returns True if all characters in the string are upper case |
| [join()](https://www.w3schools.com/python/ref_string_join.asp) | Joins the elements of an iterable to the end of the string |
| [ljust()](https://www.w3schools.com/python/ref_string_ljust.asp) | Returns a left justified version of the string |
| [lower()](https://www.w3schools.com/python/ref_string_lower.asp) | Converts a string into lower case |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Returns a left trim version of the string |
| [maketrans()](https://www.w3schools.com/python/ref_string_maketrans.asp) | Returns a translation table to be used in translations |
| [partition()](https://www.w3schools.com/python/ref_string_partition.asp) | Returns a tuple where the string is parted into three parts |
| [replace()](https://www.w3schools.com/python/ref_string_replace.asp) | Returns a string where a specified value is replaced with a specified value |
| [rfind()](https://www.w3schools.com/python/ref_string_rfind.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rindex()](https://www.w3schools.com/python/ref_string_rindex.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rjust()](https://www.w3schools.com/python/ref_string_rjust.asp) | Returns a right justified version of the string |
| [rpartition()](https://www.w3schools.com/python/ref_string_rpartition.asp) | Returns a tuple where the string is parted into three parts |
| [rsplit()](https://www.w3schools.com/python/ref_string_rsplit.asp) | Splits the string at the specified separator, and returns a list |
| [rstrip()](https://www.w3schools.com/python/ref_string_rstrip.asp) | Returns a right trim version of the string |
| [split()](https://www.w3schools.com/python/ref_string_split.asp) | Splits the string at the specified separator, and returns a list |
| [splitlines()](https://www.w3schools.com/python/ref_string_splitlines.asp) | Splits the string at line breaks and returns a list |
| [startswith()](https://www.w3schools.com/python/ref_string_startswith.asp) | Returns true if the string starts with the specified value |
| [strip()](https://www.w3schools.com/python/ref_string_strip.asp) | Returns a trimmed version of the string |
| [swapcase()](https://www.w3schools.com/python/ref_string_swapcase.asp) | Swaps cases, lower case becomes upper case and vice versa |
| [title()](https://www.w3schools.com/python/ref_string_title.asp) | Converts the first character of each word to upper case |
| [translate()](https://www.w3schools.com/python/ref_string_translate.asp) | Returns a translated string |
| [upper()](https://www.w3schools.com/python/ref_string_upper.asp) | Converts a string into upper case |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Fills the string with a specified number of 0 values at the beginning |

**BASIC STRING OPERATIONS**

In python, String operators represent the different types of operations that can be employed on the program’s string type of variables. Python allows several string operators that can be applied on the python string are as below:

* Assignment operator(=)
* Concatenate operator(+)
* String repetition operator(\*)
* String slicing operator([])
* String comparison operator(“==” & “!=”)
* Membership operator(“in”& “not in”)
* Escape sequence operator( \)
* String formatting operator(“%” & “{}”)

**1)Assignment operator(=)**

Python string can be assigned to any variable with an assignment operator “= “. Python string can be defined with either single quotes [‘ ’], double quotes[“ ”] or triple quotes[‘’’  ‘’’]. var\_name = “string” assigns “string” to variable var\_name.

**syntax:**

varaiblename="string"

**Example:**

# assignment operator "="

str="hii everyone !welcome to the world of python!"

print(str)

**Output:**  
hii everyone !welcome to the world of python!

**2) Concatenate operator: “+.”**

Two strings can be concatenated or join using the “+” operator in python.

Concatenating strings simply means combining or adding strings together. We can combine as many as strings we want.

To combine strings , use plus sign( + ).

**Example:**

#concatenating the strings

str1="hii everyone!"

str2="welcome to the world of python programming"

str=str1+str2 #concatenating the two strings str1 & str2

print(str)

**Output:**  
hiieveryone!welcome to the world of python programming

1. **String repetition operator: “\*.”**

The same string can be repeated in python by n times using string\*n.

**Example:**

#String repetition operator (\*)

str="Python is amazing!"

print(str\*5)

**Output:**

Python is amazing!Python is amazing!Python is amazing!Python is amazing!Python is amazing!

1. **String slicing operator“[]”**

Characters from a specific index of the string can be accessed with the string[index] operator. The index is interpreted as a positive index starting from 0 from the left side and a negative index starting from -1 from the right side.

To access a range of characters,usesilicing. Slicing uses square brackets( [] ). These square brackets are called as silicing operators ( [ ] ).

The square brackets can contain two integers separated by a colon( : ). The first integer is the start index, the second integer is the end index (exclusive).

**Example** SYNTAX : str[startindex:stopindex]

It prints the startindex value to the stopindex value excluding the stopindex element.

#slicing str[startindex:endindex]

str="Hi Everyone"

print(str[0:5]) # prints Hi Ev

**Output:**

Hi Everyon

(\*\*\*REFER STRING SLICING TOPIC\*\*\*)

**5)String comparison operator(“==” & “!=”)**

The string comparison operator in python is used to compare two strings.

* “==” operator returns Boolean True if two strings are the same and return Boolean False if two strings are not the same.
* “!=” operator returns Boolean True if two strings are not the same and return Boolean False if two strings are the same.

These operators are mainly used along with if condition to compare two strings where the decision is to be taken based on string comparison.

**Example:**

**1)#string comparision operator**

str1=5

str2=5

print(str1==str2)

**Output:**  
True

**2)#string comparision operator**

str1="Apples"

str2="Bananas"

print(str1==str2)

**Output:**

False

**3)#string comparision operator**

str1=5

str2=4

print(str1!=str2)

**Output:**

True

4)#string comparision operator

str1=5

str2=5

print(str1!=str2)

**Output:**

False

**6) Membership operator(“in” & “not in”)**

Membership operators are used to check if a sequence is present in an object like strings,list,etc.

There are 2 membership operators:

|  |  |
| --- | --- |
| Operator | Name |
| In | **The in Operator** |
| **not in** | **The not in Operator** |

**The in Operator**

The in operator returns True if a sequence or value is present in an object.

Example:

1. This example prints True because the sequence "python" is present in the str variable.

str="welcome to the world of python programming"

x="python" in str

print(x)

**Output:**

True

1. This example prints False because the sequence "hii" is not present in the str variable.

str="welcome to the world of python programming"

x="hii" in str

print(x)

**Output:**

False

**The not in operator**

The not in operator returns True if a sequence or value is NOT present in an object.

The not in operator returns False if a sequence or value is present in an object.

**Example:**

1. This example prints True because the sequence “vegetables” is NOT present in the string variable .

#the not in operator

str="I love fruits"

x="vegetables" not in str

print(x)

**Output:**

True

2)

This example prints False because the sequence “fruits” is NOT present in the string variable .

#the not in operator

str="I love fruits"

x="fruits" not in str

print(x)

**Output:**

False

**7) Escape sequence operator( \)**

To insert a non-allowed character in the given input string, an escape character is used. An escape character is a “\” or “backslash” operator followed by a non-allowed character. An example of a non-allowed character in python string is inserting double quotes in the string surrounded by double-quotes.

Escaping characters is important in handling strings.

It helps us to make sure that our strings are recognized as a pieces of text, and not as part of code.

**Example:**

1)#this will produce a syntax error

str='let's learn Python'

print(str)

**Output:**

Invalid Syntax error

2)#this will NOT produce a syntax error

str='let\'s learn Python'

print(str)

**Output:**

let's learn Python

**8) String formatting operator(“%” & “{}”)**

**Method-1**

String formatting operator is used to format a string as per requirement. To insert another type of variable along with string, the “%” operator is used along with python string. “%” is prefixed to another character indicating the type of value we want to insert along with the python string. Please refer to the below table for some of the commonly used different string formatting specifiers:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| %d | Signed decimal integer |
| %u | Unsigned decimal integer |
| %c | Character |
| %s | String |
| %f | Floating-point real number |

**Example:**

#String formatting operator("%" & "{}")

name="Python"

year=1991

str1="Hello World! welcome to %s "%(name)

str2="%s is developed by Gudio van Rossum in the year %d"%(name,year)

str3="%s is interpreted,interactive and object-oriented programming language"%(name)

print(str1)

print(str2)

print(str3)

**Output:**

Hello World! welcome to Python

Python is developed by Gudio van Rossum in the year 1991

Python is interpreted,interactive and object-oriented programming language

**Method-2**

In Python, we can format a string by adding substring(s) within it.

The format( ) function allows us to format strings.

**Placeholders { }**

Placeholders help us control which part of the string should be formatted.

They are defined using curly braces { } .

In this example, we will concatenate (add) a substring to where the curly braces are placed.

#Python Formatting strings

x="I love {} very much!"

str=x.format("Python")

print(x) # original string

print(str) # formatted string

**Output:**

I love {} very much!

I love Python very much!

**STRING SLICING**

To access a range of characters,usesilicing. Slicing uses square brackets( [] ).

The square brackets can contain two integers separated by a colon( : ). The first integer is the start index, the second integer is the end index (exclusive).

1. str(startindex:stopindex)

It prints the startindex value to the stopindexvalue excluding the stopindex element.

1. str(startindex:)

If we donot mention the stopindex value then it prints the value form startindex to the end of the string(including the last element ).

1. str(:stopindex)

If we donot mention the start index value then it prints the value form startindex of the string to the end of the stopindex(excluding the stopindexelement ).

1. str(:)

If we donot mention the start index and stop index then it prints the string as the same as the original string.

Example

1)str(startindex:stopindex)

It prints the startindex value to the stopindex value excluding the stopindex element.

#slicingstr[startindex:endindex]

str="Hi Everyone"

print(str[0:5]) # prints Hi Ev

**Output:**

Hi Ev

2)str(startindex:)

If we donot mention the stopindex value then it prints the value form startindex to the end of the string(including the last element ).

#slicing str[startindex:]

str="Hi Everyone"

print(str[3:]) # prints Everyone

**Output:**

Everyone

1. str(:stopindex)

If we donot mention the start index value then it prints the value form startindex of the string to the end of the stopindex (excluding the stopindexelement ).

#slicing str[:stopindex]

str="Hi Everyone"

print(str[:10]) # prints Hi Everyon

**Output:**

Hi Everyon

4)str(:)

If we donot mention the start index and stop index then it prints the string as the same as the original string.

#slicing str[:]

str="Hi Everyone"

print(str[:]) # prints Hi Everyone

**Output:**

Hi Everyone

**String Comparison in Python**

**Method1: sing** relational operators

The relational operators compare the Unicode values of the characters of the strings from the zeroth index till the end of the string. It then returns a boolean value according to the operator used.

**Example:**

print("AIML" == "AIML")

print("Aiml" < "aiml")

print("Aiml" > "aiml")

print("Aiml" != "Aiml")

**Output:**

True

True

False

False

**Method 2:**Using [**is**](https://www.geeksforgeeks.org/python-membership-identity-operators-not-not/) and [**is not**](https://www.geeksforgeeks.org/python-membership-identity-operators-not-not/)

The **==** operator compares the values of both the operands and checks for value equality. Whereas **is** operator checks whether both the operands refer to the same object or not. The same is the case for != and **is not**.

Let us understand this with an example:

str1 = "CMR"

str2 = "CMR"

str3 = str1

print("ID of str1 =", hex(id(str1)))

print("ID of str2 =", hex(id(str2)))

print("ID of str3 =", hex(id(str3)))

print(str1 is str1)

print(str1 is str2)

print(str1 is str3)

str1 += "IT"

str4 = "CMRIT"

print("\nID of changed str1 =", hex(id(str1)))

print("ID of str4 =", hex(id(str4)))

print(str1 **is** str4)

**Output:**

ID of str1 = 0x7f6037051570

ID of str2 = 0x7f6037051570

ID of str3 = 0x7f6037051570

True

True

True

ID of changed str1 = 0x7f60356137d8

ID of str4 = 0x7f60356137a0

False

The object ID of the strings may vary on different machines. The object IDs of str1, str2 and str3 were the same therefore they the result is True in all the cases. After the object id of str1 is changed, the result of str1 and str2 will be false. Even after creating str4 with the same contents as in the new str1, the answer will be false as their object IDs are different.

Vice-versa will happen with **is not**.

**TESTING STRINGS**

String class in python has various inbuilt methods which allows to check for different types of strings.

| **Method name** | **Method Description** |
| --- | --- |
| isalnum() | Returns True if string is alphanumeric |
| isalpha() | Returns True if string contains only alphabets |
| isdigit() | Returns True if string contains only digits |
| isidentifier() | Return True is string is valid identifier |
| islower() | Returns True if string is in lowercase |
| isupper() | Returns True if string is in uppercase |
| isspace() | Returns True if string contains only whitespace |

Example:

>>> s = "welcome to python"

>>> s.isalnum()

False

>>> "Welcome".isalpha()

True

>>> "2012".isdigit()

True

>>> "first Number".isidentifier()

False

>>> s.islower()

True

>>> "WELCOME".isupper()

True

>>> " **\t**".isspace()

True

**Searching for Substrings**

| **Method Name** | **Methods Description** |
| --- | --- |
| endswith(s1: str): bool | Returns True if strings ends with substring s1 |
| startswith(s1: str): bool | Returns True if strings starts with substring s1 |
| count(substring): int | Returns number of occurrences of substring the string |
| find(s1): int | Returns lowest index from where s1 starts in the string, if string not found returns -1 |
| rfind(s1): int | Returns highest index from where s1 starts in the string, if string not found returns -1 |

>>> s = "welcome to python"

>>> s.endswith("thon")

True

>>> s.startswith("good")

False

>>> s.find("come")

3

>>> s.find("become")

-1

>>> s.rfind("o")

15

>>> s.count("o")

3

>>>

A sequence is a datatype that represents a group of elements. The purpose of any sequence is to store and process group elements. In python, strings, lists, tuples and dictionaries are very important sequence datatypes.

# LIST:

A List is data type which can store different types of elements. And we can also store duplicate data also.

## Creating a List:

Creating a list is as simple as putting different comma-separated values between square brackets [ ] .

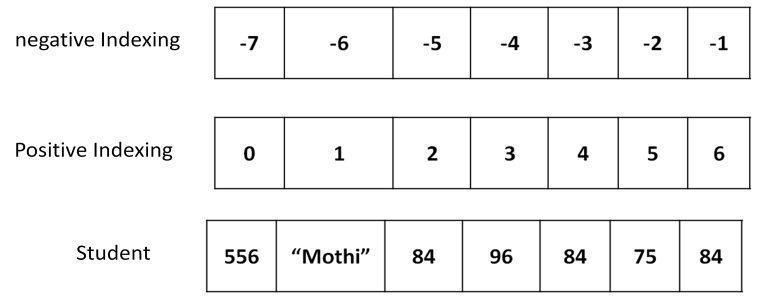
student = [556, ‚Mothi‛, 84, 96, 84, 75, 84, 7.35 ]

We can create empty list without any elements by simply writing empty square brackets as: student=[ ]

We can create a list by embedding the elements inside a pair of square braces []. The elements in the list should be separated by a comma (,).

## Accessing/Reading Values in list:

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. To view the elements of a list as a whole, we can simply pass the list name to print function.



## Ex:

student = [556, ‚Mothi‛, 84, 96, 84, 75, 84 ] print(student)

print(student[0]) # Access 0th element print(student[0:2]) # Access 0th to 1st elements print(student[2: ]) # Access 2nd to end of list elements print(student[ :3]) # Access starting to 2nd elements print(student[ : ]) # Access starting to ending elements print(student[-1]) # Access last index value print(student[-1:-7:-1]) # Access elements in reverse order

## Output:

[556, ‚Mothi‛, 84, 96, 84, 75, 84]

Mothi

[556, ‚Mothi‛]

[84, 96, 84, 75, 84]

[556, ‚Mothi‛, 84]

[556, ‚Mothi‛, 84, 96, 84, 75, 84]

84

[84, 75, 84, 96, 84, ‚Mothi‛]

## Creating lists using range() function:

We can use range() function to generate a sequence of integers which can be stored in a list. To store numbers from 0 to 10 in a list as follows.

numbers = list( range(0,11) )

print(numbers) # [0,1,2,3,4,5,6,7,8,9,10]

To store even numbers from 0 to 10in a list as follows. numbers = list( range(0,11,2) ) print(numbers) # [0,2,4,6,8,10]

## Looping on lists:

We can also display list by using for loop (or) while loop. The len( ) function useful to know the numbers of elements in the list. while loop retrieves starting from 0th to the last element i.e. n-1

## Ex-1:

numbers = [1,2,3,4,5] for i in numbers:

print(i,end="")

## Output:

1 2 3 4 5

## Updating and deleting lists:

Lists are *mutable*. It means we can modify the contents of a list. We can append, update or delete the elements of a list depending upon our requirements.

Appending an element means adding an element at the end of the list. To, append a new element to the list, we should use the append() method.

## Example:

lst=[1,2,4,5,8,6]

print(lst) # [1,2,4,5,8,6]

lst.append(9)

print(lst) # [1,2,4,5,8,6,9]

Updating an element means changing the value of the element in the list. This can be done by accessing the specific element using indexing or slicing and assigning a new value.

## Example:

lst=[4,7,6,8,9,3]

print(lst) # [4,7,6,8,9,3]

lst[2]=5 # updates 2nd element in the list

print(lst) # [4,7,5,8,9,3]

lst[2:5]=10,11,12 # update 2nd element to 4th element in the list

print(lst) # [4,7,10,11,12,3]

Deleting an element from the list can be done using *‘del’* statement. The *del* statement takes the position number of the element to be deleted.

## Example:

lst=[5,7,1,8,9,6]

del lst[3] # delete 3rd element from the list i.e., 8

print(lst) # [5,7,1,9,6]

If we want to delete entire list, we can give statement like **del lst.**

## Concatenation of Two lists:

We can simply use „+‟ operator on two lists to join them. For example, „x‟ and „y‟ are two lists. If we wrte x+y, the list „y‟ is joined at the end of the list „x‟.

## Example:

x=[10,20,32,15,16]

y=[45,18,78,14,86]

print(x+y) # [10,20,32,15,16,45,18,78,14,86]

## Repetition of Lists:

We can repeat the elements of a list „n‟ number of times using „\*‟ operator.

x=[10,54,87,96,45]

print(x\*2) # [10,54,87,96,45,10,54,87,96,45]

## Membership in Lists:

We can check if an element is a member of a list by using „in‟ and „not in‟ operator. If the element is a member of the list, then „in‟ operator returns **True** otherwise returns **False**. If the element is not in the list, then „not in‟ operator returns **True** otherwise returns **False**.

## Example:

x=[10,20,30,45,55,65] a=20

print(a in x) # True a=25

print(a in x) # False a=45

print(a not in x) # False a=40

print(a not in x) # True

## Methods in Lists:

1. **A.index(X):** Returns the first occurrence of X in the list A.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  print(A.index(45)) | 3 |

1. **A.append(X):** Adds the element X at the end of list A.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  A.append(97) print(A) | [1, 3, 13, 45, 56, 62,  45, 58, 89, 13, 3, 56, 1,  97] |

1. **A.insert( i,X ):** Inserts X to the list A in ith position.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  A.insert(4,"CSE") print(A) | [1, 3, 13, 45, 'CSE', 56, 62, 45, 58,  89, 13, 3, 56, 1] |

1. **B=A.copy():** copies all the list elements of A into B.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  B=A.copy() print("B=", B) | B= [1, 3, 13, 45, 56, 62, 45, 58,  89, 13, 3, 56, 1] |

1. **A.extend(B):** Appends list B to list A.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,]  B=[62,45,58,89,13,3,56,1]  A.extend(B) print("A=",A)  print("B=",B) | A= [1, 3, 13, 45, 56, 62,  45, 58, 89, 13, 3, 56, 1]  B= [62, 45, 58, 89, 13,  3, 56, 1] |

1. **A.count(X):** Returns number of occurrences of X in the list A.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  print(A.count(45)) | 2 |

1. **A.remove(X):** Removes X from the list X.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  A.remove(13) print(A) | [1, 3, 45, 56, 62, 45,  58, 89, 13, 3, 56, 1] |

1. **A.pop():** Removes the ending element from list A.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  A.pop() print(A) | [1, 3, 13, 45, 56, 62,  45, 58, 89, 13, 3, 56] |

1. **A.sort():** Sorts the elements of list A into ascending order.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  A.sort() print(A) | [1, 1, 3, 3, 13, 13, 45,  45, 56, 56, 58, 62, 89] |

1. **A.reverse():** Reverses the sequence of elements in the list.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  A.reverse() print(A) | [1, 56, 3, 13, 89, 58,  45, 62, 56, 45, 13, 3, 1] |

1. **A.clear():** Removes all elements from the list A.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  A.clear() print(A) | [ ] |

1. **max(A):** Returns biggest element in the list A.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  print(max(A)) | 89 |

1. **min(A):** Returns smallest element in the list B.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,3,13,45,56,62,45,58,89,13,3,56,1]  print(min(A)) | 1 |

1. **len(A):** Returns the length of list A

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=[1,2,3,4,5]  print(len(A)) | 5 |

## Nested Lists:

A list within another list is called a *nested list*. We know that a list contains several elements. When we take a list as an element in another list, then that list is called a nested list.

|  |  |
| --- | --- |
| **Program** | **Output** |
| a=[[1,2,3],[4,5,6],[7,8,9]]  print(a) print(a[1]) print(a[1][1]) print(a[2]) | [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  [4, 5, 6]  5  [7, 8, 9] |

# TUPLE:

A Tuple is a python sequence which stores a group of elements or items. Tuples are similar to lists but the main difference is **tuples are immutable** whereas lists are mutable. Once we create a tuple we **cannot modify** its elements. Hence, we **cannot perform** operations like append(), extend(), insert(), remove(), pop() and clear() on tuples. Tuples are generally used to store data which should not be modified and retrieve that data on demand.

## Creating Tuples:

We can create a tuple by writing elements separated by commas inside parentheses( ).

The elements can be same datatype or different types.

tup=(10, 20, 31.5, ‘Gudivada’)

If we do not mention any brackets and write the elements separating them by comma,

then they are taken by default as a tuple.

tup= 10, 20, 34, 47

It is possible to create a tuple from a list. This is done by converting a list into a tuple

using tuple function.

t=tuple(range(2,11,2))

print(t) # display (2,4,6,8,10)

Another way to create a tuple by using range( ) function that returns a sequence.

t=tuple(range(2,11,2))

## Accessing the tuple elements:

## Accessing the tuple elements:

Accessing the elements from a tuple can be done using indexing or slicing. This is same as that of a list. Indexing represents the position number of the element in the tuple. The position starts from 0.

tup=(50,60,70,80,90)

print(tup[0])

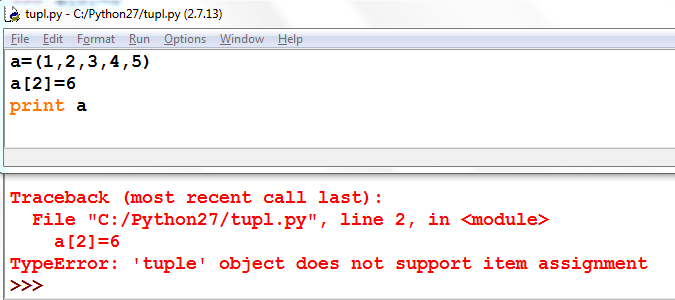
print(tup[1:4])

print(tup[-1]) print(tup[-1:-4:-1])

## Updating and deleting elements:

Tuples are immutable which means you cannot update, change or delete the values of tuple elements.

## Example-1:



## Methods on tuple:

1. **len(T):** Return the length of tuple T.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=(1,3,5,4,5,46,45,8,9,1,2,3)  print(len(A)) | 12 |

1. **max(T):** Returns the maximum value in tuple T.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=(3,5,4,5,42,45,8,9,3)  print(max(A)) | 45 |

1. **min(T):** Returns the minimum value in tuple T.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=(3,5,4,5,42,45,8,9,3)  print(min(A)) | 3 |

1. **T.count(X):** Returns the count of X element in tuple T.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=(3,5,4,5,42,45,8,9,3)  print(A.count(5)) | 2 |

1. **T.index(X):** Returns the first occurrence of the element X in T.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=(3,5,4,5,42,45,8,9,3)  print(A.index(5)) | 1 |

1. **B=sorted(T):** Sorts the elements of tuple T into ascending order and store the result in List.

|  |  |
| --- | --- |
| **Program** | **Output** |
| A=(3,5,4,5,42,45,8,9,3)  B=sorted(A)  print(B) | [3, 3, 4, 5, 5, 8, 9, 42,  45] |

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***