**What is String in Python?**

A string is a sequence of characters.

A character is simply a symbol. For example, the English language has 26 characters.

Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used.

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding. You can learn more about Unicode from here.

**How to create a string in Python?**

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

# all of the following are equivalent

my\_string = 'Hello'

print(my\_string)

my\_string = "Hello"

print(my\_string)

my\_string = '''Hello'''

print(my\_string)

# triple quotes string can extend multiple lines

my\_string = """Hello, welcome to

the world of Python"""

print(my\_string)

When you run the program, the output will be:

Hello

Hello

Hello

Hello, welcome to

the world of Python

## How to access characters in a string?

We can access individual characters using indexing and a range of characters using slicing. Index starts from 0. Trying to access a character out of index range will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator (colon).

str = 'programiz'

print('str = ', str)

#first character

print('str[0] = ', str[0])

#last character

print('str[-1] = ', str[-1])

#slicing 2nd to 5th character

print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character

print('str[5:-2] = ', str[5:-2])

If we try to access index out of the range or use decimal number, we will get errors.

# index must be in range

my\_string[15]

...

IndexError: string index out of range

# index must be an integer

my\_string[1.5]

...

TypeError: string indices must be integers

Slicing can be best visualized by considering the index to be between the elements as shown below.

If we want to access a range, we need the index that will slice the portion from the string.

## 

## How to change or delete a string?

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name.

my\_string = 'programiz'

my\_string[5] = 'a'

...

TypeError: 'str' object does not support item assignment

my\_string = 'Python'

my\_string

'Python'

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

del my\_string[1]

...

TypeError: 'str' object doesn't support item deletion

>>> del my\_string

>>> my\_string

...

NameError: name 'my\_string' is not defined

## Python String Operations

There are many operations that can be performed with string which makes it one of the most used datatypes in Python.

### Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The **+** operator does this in Python. Simply writing two string literals together also concatenates them.

The **\*** operator can be used to repeat the string for a given number of times.

str1 = 'Hello'

str2 ='World!'

# using +

print('str1 + str2 = ', str1 + str2)

# using \*

print('str1 \* 3 =', str1 \* 3)

Writing two string literals together also concatenates them like **+** operator.

If we want to concatenate strings in different lines, we can use parentheses.

# two string literals together

'Hello ''World!'

'Hello World!'

# using parentheses

s = ('Hello '

... 'World')

s

'Hello World'

### Iterating Through String

Using **for loop** we can iterate through a string. Here is an example to count the number of 'l' in a string.

count = 0

for letter in 'Hello World':

if(letter == 'l'):

count += 1

print(count,'letters found')

### String Membership Test

We can test if a sub string exists within a string or not, using the keyword in.

'a' in 'program'

True

'at' not in 'battle'

False

### Built-in functions to Work with Python

Various built-in functions that work with sequence, works with string as well.

Some of the commonly used ones are enumerate() and len(). The enumerate() function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.

Similarly, len() returns the length (number of characters) of the string.

str = 'cold'

# enumerate()

list\_enumerate = list(enumerate(str))

print('list(enumerate(str) = ', list\_enumerate)

#character count

print('len(str) = ', len(str))

## Python String Formatting

### Escape Sequence

If we want to print a text like -He said, "What's there?"- we can neither use single quote or double quotes. This will result into SyntaxError as the text itself contains both single and double quotes.

print("He said, "What's there?"")

...

SyntaxError: invalid syntax

print('He said, "What's there?"')

...

SyntaxError: invalid syntax

One way to get around this problem is to use triple quotes. Alternatively, we can use escape sequences.

An escape sequence starts with a backslash and is interpreted differently. If we use single quote to represent a string, all the single quotes inside the string must be escaped. Similar is the case with double quotes. Here is how it can be done to represent the above text.

# using triple quotes

print('''He said, "What's there?"''')

# escaping single quotes

print('He said, "What\'s there?"')

# escaping double quotes

print("He said, \"What's there?\"")

Here is a list of all the escape sequence supported by Python.

|  |  |
| --- | --- |
| Escape Sequence in Python | |
| **Escape Sequence** | **Description** |
| \newline | Backslash and newline ignored |
| \\ | Backslash |
| \' | Single quote |
| \" | Double quote |
| \a | ASCII Bell |
| \b | ASCII Backspace |
| \f | ASCII Formfeed |
| \n | ASCII Linefeed |
| \r | ASCII Carriage Return |
| \t | ASCII Horizontal Tab |
| \v | ASCII Vertical Tab |
| \ooo | Character with octal value ooo |
| \xHH | Character with hexadecimal value HH |

Here are some examples

print("C:\\Python32\\Lib")

C:\Python32\Lib

print("This is printed\nin two lines")

This is printed

in two lines

print("This is \x48\x45\x58 representation")

This is HEX representation

### Raw String to ignore escape sequence

Sometimes we may wish to ignore the escape sequences inside a string. To do this we can place r or R in front of the string. This will imply that it is a raw string and any escape sequence inside it will be ignored.

print("This is \x61 \ngood example")

This is a

good example

print(r"This is \x61 \ngood example")

This is \x61 \ngood example

### The format() Method for Formatting Strings

The format() method that is available with the string object is very versatile and powerful in formatting strings. Format strings contains curly braces {} as placeholders or replacement fields which gets replaced.

We can use positional arguments or keyword arguments to specify the order.

# default(implicit) order

default\_order = "{}, {} and {}".format('John','Bill','Sean')

print('\n--- Default Order ---')

print(default\_order)

# order using positional argument

positional\_order = "{1}, {0} and {2}".format('John','Bill','Sean')

print('\n--- Positional Order ---')

print(positional\_order)

# order using keyword argument

keyword\_order = "{s}, {b} and {j}".format(j='John',b='Bill',s='Sean')

print('\n--- Keyword Order ---')

print(keyword\_order)

The format() method can have optional format specifications. They are separated from field name using colon. For example, we can left-justify <, right-justify > or center ^ a string in the given space. We can also format integers as binary, hexadecimal etc. and floats can be rounded or displayed in the exponent format. There are a ton of formatting you can use. Visit here for all the string formatting available with the format() method.

# formatting integers

"Binary representation of {0} is {0:b}".format(12)

'Binary representation of 12 is 1100'

# formatting floats

"Exponent representation: {0:e}".format(1566.345)

'Exponent representation: 1.566345e+03'

# round off

"One third is: {0:.3f}".format(1/3)

'One third is: 0.333'

# string alignment

"|{:<10}|{:^10}|{:>10}|".format('butter','bread','ham')

'|butter | bread | ham|'

### Old style formatting

We can even format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

x = 12.3456789

print('The value of x is %3.2f' %x)

The value of x is 12.35

print('The value of x is %3.4f' %x)

The value of x is 12.3457

## Common Python String Methods

There are numerous methods available with the string object. The format() method that we mentioned above is one of them. Some of the commonly used methods are lower(), upper(), join(), split(), find(), replace() etc. Here is a complete list of all the built-in methods to work with strings in Python.

"SaKeeB".lower()

'sakeeb'

"SaKeeB".upper()

'SAKEEB'

"This will split all words into a list".split()

['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']

' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])

'This will join all words into a string'

'Happy New Year'.find('ew')

7

'Happy New Year'.replace('Happy','Brilliant')

'Brilliant New Year'

# Python Program to Sort Words in Alphabetic Order

# Program to sort alphabetically the words form a string provided by the user

# change this value for a different result

my\_str = "Hello this Is an Example With cased letters"

# uncomment to take input from the user

#my\_str = input("Enter a string: ")

# breakdown the string into a list of words

words = my\_str.split()

# sort the list

words.sort()

# display the sorted words

print("The sorted words are:")

for word in words:

print(word)

**String Function**

|  |  |
| --- | --- |
| Python String capitalize() | Converts first character to Capital Letter |
| Python String center() | Pads string with specified character |
| Python String casefold() | converts to casefolded strings |
| Python String count() | returns occurrences of substring in string |
| Python String endswith() | Checks if String Ends with the Specified Suffix |
| Python String expandtabs() | Replaces Tab character With Spaces |
| Python String encode() | returns encoded string of given string |
| Python String find() | Returns the index of first occurrence of substring |
| Python String format() | formats string into nicer output |
| Python String index() | Returns Index of Substring |
| Python String isalnum() | Checks Alphanumeric Character |
| Python String isalpha() | Checks if All Characters are Alphabets |
| Python String isdecimal() | Checks Decimal Characters |
| Python String isdigit() | Checks Digit Characters |
| Python String isidentifier() | Checks for Valid Identifier |
| Python String islower() | Checks if all Alphabets in a String are Lowercase |
| Python String isnumeric() | Checks Numeric Characters |
| Python String isprintable() | Checks Printable Character |
| Python String isspace() | Checks Whitespace Characters |
| Python String istitle() | Checks for Titlecased String |
| Python String isupper() | returns if all characters are uppercase characters |
| Python String join() | Returns a Concatenated String |
| Python String ljust() | returns left-justified string of given width |
| Python String rjust() | returns right-justified string of given width |
| Python String lower() | returns lowercased string |
| Python String upper() | returns uppercased string |
| Python String swapcase() | swap uppercase characters to lowercase; vice versa |
| Python String lstrip() | Removes Leading Characters |
| Python String rstrip() | Removes Trailing Characters |
| Python String strip() | Removes Both Leading and Trailing Characters |
| Python String partition() | Returns a Tuple |
| Python String maketrans() | returns a translation table |
| Python String rpartition() | Returns a Tuple |
| Python String translate() | returns mapped charactered string |
| Python String replace() | Replaces Substring Inside |
| Python String rfind() | Returns the Highest Index of Substring |
| Python String rindex() | Returns Highest Index of Substring |
| Python String split() | Splits String from Left |
| Python String rsplit() | Splits String From Right |
| Python String splitlines() | Splits String at Line Boundaries |
| Python String startswith() | Checks if String Starts with the Specified String |
| Python String title() | Returns a Title Cased String |
| Python String zfill() | Returns a Copy of The String Padded With Zeros |
| Python String format\_map() | Formats the String Using Dictionary |

**Other Functions**

|  |  |
| --- | --- |
| Python any() | Checks if any Element of an Iterable is True |
| Python all() | returns true when all elements in iterable is true |
| Python ascii() | Returns String Containing Printable Representation |
| Python bool() | Converts a Value to Boolean |
| Python bytearray() | returns array of given byte size |
| Python bytes() | returns immutable bytes object |
| Python compile() | Returns a Python code object |
| Python complex() | Creates a Complex Number |
| Python enumerate() | Returns an Enumerate Object |
| Python filter() | constructs iterator from elements which are true |
| Python float() | returns floating point number from number, string |
| Python input() | reads and returns a line of string |
| Python int() | returns integer from a number or string |
| Python iter() | returns iterator for an object |
| Python len() | Returns Length of an Object |
| Python max() | returns largest element |
| Python min() | returns smallest element |
| Python map() | Applies Function and Returns a List |
| Python ord() | returns Unicode code point for Unicode character |
| Python reversed() | returns reversed iterator of a sequence |
| Python slice() | creates a slice object specified by range() |
| Python sorted() | returns sorted list from a given iterable |
| Python sum() | Add items of an Iterable |
| Python zip() | Returns an Iterator of Tuples |