PYTHON STRINGS

String

- an immutable sequence of Unicode characters.
- Each character has a unique numeric value as per the UNICODE standard.
- To differentiate the string from numbers and other identifiers, the sequence of characters is included within single, double or triple quotes in its literal representation.

```
>>> 'Welcome To TutorialsPoint'
'Welcome To TutorialsPoint'
>>> "Welcome To TutorialsPoint"
'Welcome To TutorialsPoint'
>>> '''Welcome To TutorialsPoint'''
'Welcome To TutorialsPoint'
>>> """Welcome To TutorialsPoint"""
'Welcome To TutorialsPoint'
```

A string is a non-numeric data type. Obviously, we cannot use arithmetic operators with string operands. Python raises TypeError in such a case.

```
var = "Welcome To TutorialsPoint"
print (type(var))
```

```
<class 'str'>
```

```
var = 'Welcome to "Python Tutorial" from Tuto
print ("var:", var)

var = "Welcome to 'Python Tutorial' from Tuto
print ("var:", var)
```

```
var = '''
Welcome To
Python Tutorial
from TutorialsPoint
'''

print ("var:", var)
```

```
var:
Welcome To
Python Tutorial
from TutorialsPoint
```

Slicing String

- a string is an ordered sequence of Unicode characters.
- Each character in the string has a unique index in the sequence.
- The index starts with 0. First character in the string has its positional index 0. The index keeps incrementing towards the end of string.

Slicing String

If a string variable is declared as var="HELLO PYTHON", index of each character in the string is as follows –

```
>>> var="HELLO PYTHON"
>>> var[0]
'H'
>>> var[7]
'Y'
>>> var[11]
'N'
>>> var[12]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

H	E	L	L	O		P	Y	T	H	O	N
0	1	2	3	4	5	6	7	8	9	10	11

Slicing String

In case of negative indexing, the character at the end has -1 index and the index decrements from right to left, as a result the first character H has -12 index.

```
>>> var[-1]
'N'
>>> var[-5]
'Y'
>>> var[-12]
'H'
>>> var[-13]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

H	E	L	L	O		P	Y	T	H	O	N
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Slicing String

In Python, string is an immutable object. The object is immutable if it cannot be modified in-place, once stored in a certain memory location. You can retrieve any character from the string with the help of its index, but you cannot replace it with another character.

```
var="HELLO PYTHON"
var[7]="y"
print (var)
```

```
Traceback (most recent call last):
 File "C:\Users\users\example.py", line 2, in <module>
  var[7]="y"
  ~~~^^^
TypeError: 'str' object does not support item assignment
```

Slicing String

Python defines ":" as string slicing operator. It returns a substring from the original string. Its general usage is

substr=var[x:y]

The first operand **x** is the index of the first character of the desired slice. The second operand **y** is the index of the character next to the last in the desired string.

```
var="HELLO PYTHON"
print ("var:",var)
print ("var[3:8]:", var[3:8])
print ("var[-9:-4]:", var[-9:-4])
```

```
var: HELLO PYTHON
var[3:8]: LO PY
var[-9:-4]: LO PY
```

Note: Both the operands for Python's Slice operator are optional.

Slicing String

```
var="HELLO PYTHON"
print ("var:",var)
print ("var[0:5]:", var[0:5])
print ("var[:5]:", var[:5])
```

```
var="HELLO PYTHON"
print ("var:",var)
print ("var[0:12]:", var[0:12])
print ("var[:]:", var[:])
```

```
var="HELLO PYTHON"
print ("var:",var)
print ("var[6:12]:", var[6:12])
print ("var[6:]:", var[6:])
```

```
var="HELLO PYTHON"
print ("var:",var)
print ("var[-1:7]:", var[-1:7])
print ("var[7:0]:", var[7:0])
```

Modify Strings

- Converting a String to a List
- Using the Array Module
- Using the StringIO Class

Converting a String to a List

```
s1="WORD"
print ("original string:", s1)
l1=list(s1)

l1.insert(3,"L")

print (l1)

s1=''.join(l1)
print ("Modified string:", s1)
```

```
original string: WORD
['W', 'O', 'R', 'L', 'D']
Modified string: WORLD
```

Since both string and list objects are sequences, they are interconvertible. Hence, if we cast a string object to a list, modify the list either by insert(), append() or remove() methods and convert the list back to a string, to get back the modified version.

We have a string variable s1 with WORD as its value. With list() built-in function, let us convert it to a l1 list object, and insert a character L at index 3. The we use the join() method in str class to concatenate all the characters.

Using the Array Module

```
import array as ar

s1="WORD"
print ("original string:", s1)

sar=ar.array('u', s1)
sar.insert(3,"L")
s1=sar.tounicode()

print ("Modified string:", s1)
```

```
original string: WORD
Modified string: WORLD
```

To modify a string, construct an array object. Python standard library includes array module. We can have an array of Unicode type from a string variable. Items in the array have a zero based index. So, we can perform array operations such as append, insert, remove etc. Let us insert L before the character D. Now, with the help of tounicode() method, get back the modified string

Using the StringIO Class

```
original string: WORD
Modified string: WORLD
```

```
import io

s1="WORD"
print ("original string:", s1)

sio=io.StringIO(s1)
sio.seek(3)
sio.write("LD")
s1=sio.getvalue()

print ("Modified string:", s1)
```

Python's io module defines the classes to handle streams. The StringIO class represents a text stream using an in-memory text buffer. A StringIO object obtained from a string behaves like a File object. Hence we can perform read/write operations on it. The getvalue() method of StringIO class returns a string.

String Concatenation

The "+" operator is well-known as an addition operator, returning the sum of two numbers. However, the "+" symbol acts as string concatenation operator in Python. It works with two string operands, and results in the concatenation of the two. The characters of the string on the right of plus symbol are appended to the string on its left. Result of concatenation is a new string.

String Concatenation

```
str1="Hello"
str2="World"
print ("String 1:",str1)
print ("String 2:",str2)
str3=str1+str2
print("String 3:",str3)
```

```
String 1: Hello
String 2: World
String 3: HelloWorld
```

* acts as a repetition operator in Python.

```
>>> "Hello"*3
'HelloHelloHello'
```

```
str1="Hello"
str2="World"
blank=" "
print ("String 1:",str1)
print ("String 2:",str2)
str3=str1+blank+str2
print("String 3:",str3)
```

```
String 1: Hello
String 2: World
String 3: Hello World
```

String Concatenation

```
str1="Hello"
str2="World"
print ("String 1:",str1)
print ("String 2:",str2)
str3=str1+str2*3
print("String 3:",str3)
str4=(str1+str2)*3
print ("String 4:", str4)
```

Both the string operators, (*) the repetition operator and (+) the concatenation operator, can be used in a single expression. The "*" operator has a higher precedence over the "+" operator.

```
String 3: HelloWorldWorldWorld
```

```
String 4: HelloWorldHelloWorldHelloWorld
```

NOTE: Apart from + and *, no other arithmetic operator symbols can be used with string operands.

String Formatting

String formatting is the process of building a string representation dynamically by inserting the value of numeric expressions in an already existing string. Python's string concatenation operator doesn't accept a non-string operand. Hence, Python offers following string formatting techniques –

- Using % operator for substitution
- Using format() method of str class
- Using f-string syntax
- Using String Template class

Escape Characters

In Python, a string becomes a raw string if it is prefixed with "r" or "R" before the quotation symbols.

```
>>> normal="Hello\nWorld"
>>> print (normal)
Hello
World
>>> raw=r"Hello\nWorld"
>>> print (raw)
Hello\nWorld
```

Sr.No	Escape Sequence & Meaning
1	**\<newline>** Backslash and newline ignored
2	**** Backslash (\)
3	**\'** Single quote (')
4	**\"** Double quote (")
5	**\a** ASCII Bell (BEL)
6	**\b** ASCII Backspace (BS)
7	**\f** ASCII Formfeed (FF)
8	**\n** ASCII Linefeed (LF)
9	**\r** ASCII Carriage Return (CR)
10	**\t** ASCII Horizontal Tab (TAB)
11	**\v** ASCII Vertical Tab (VT)
12	**\ooo** Character with octal value ooo
13	**\xhh** Character with hex value hh

String Methods

Python's built-in str class defines different methods. They help in manipulating strings. Since string is an immutable object, these methods return a copy of the original string, performing the respective processing on it.

The string methods can be classified in following categories –

- Case conversion
- Alignment
- Split and join
- Boolean
- Find and replace
- Formatting
- Translate

Python - String Methods (tutorialspoint.com)

Case conversion

Sr.No.	Method & Description
1	**capitalize()** Capitalizes first letter of string
2	**casefold()** Converts all uppercase letters in string to lowercase. Similar to lower(), but works on UNICODE characters alos
3	**lower()** Converts all uppercase letters in string to lowercase.
4	**swapcase()** Inverts case for all letters in string.
5	**title()** Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
6	**upper()** Converts lowercase letters in string to uppercase.

Alignment

Sr.No.	Methods & Description
1	**center(width, fillchar)** Returns a string padded with fillchar with the original string centered to a total of width columns.
2	**ljust(width[, fillchar])** Returns a space-padded string with the original string left-justified to a total of width columns.
3	**rjust(width,[, fillchar])** Returns a space-padded string with the original string right-justified to a total of width columns.
4	**expandtabs(tabsize = 8)** Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
5	**zfill (width)** Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).

Split and join

Sr.No.	Method & Description
1	**lstrip()** Removes all leading whitespace in string.
2	**rstrip()** Removes all trailing whitespace of string.
3	**strip()** Performs both lstrip() and rstrip() on string
4	**rsplit()** Splits the string from the end and returns a list of substrings
5	**split()** Splits string according to delimiter (space if not provided) and returns list of substrings.
6	**splitlines()** Splits string at NEWLINEs and returns a list of each line with NEWLINEs removed.
7	**partition()** Splits the string in three string tuple at the first occurrence of separator
8	**rpartition()** Splits the string in three string tuple at the ladt occurrence of separator
9	**join()** Concatenates the string representations of elements in sequence into a string, with separator string.
10	**removeprefix()** Returns a string after removing the prefix string
11	**removesuffix()** Returns a string after removing the suffix string

Boolean

Sr.No.	Methods & Description
1	**isalnum()** Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
2	**isalpha()** Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
3	**isdigit()** Returns true if the string contains only digits and false otherwise.
4	**islower()** Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
5	**isnumeric()** Returns true if a unicode string contains only numeric characters and false otherwise.
6	**isspace()** Returns true if string contains only whitespace characters and false otherwise.
7	**istitle()** Returns true if string is properly "titlecased" and false otherwise.
8	**isupper()** Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
9	**isascii()** Returns True is all the characters in the string are from the ASCII character set
10	**isdecimal()** Checks if all the characters are decimal characters
11	**isidentifier()** Checks whether the string is a valid Python identifier
12	**isprintable()** Checks whether all the characters in the string are printable

Find and replace

Sr.No.	Method & Description
1	**count(sub, beg ,end)** Counts how many times sub occurs in string or in a substring of string if starting index beg and ending index end are given.
2	**find(sub, beg, end)** Determine if sub occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
3	**index(sub, beg, end)** Same as find(), but raises an exception if str not found.
4	**replace(old, new [, max])** Replaces all occurrences of old in string with new or at most max occurrences if max given.
5	**rfind(sub, beg, end)** Same as find(), but search backwards in string.
6	**rindex(sub, beg, end)** Same as index(), but search backwards in string.
7	**startswith(sub, beg, end)** Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring sub; returns true if so and false otherwise.
8	**endswith(suffix, beg, end)** Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.

Applications???

Any questions???

REFERENCES:

- Learn Python Programming. (2023). https://www.tutorialsteacher.com/python
- Python Tutorial. (2022). https://www.w3resource.com/python/python-tutorial.php
- Python Tutorial. (n.d.). https://www.tutorialspoint.com/python/index.htm
- Python Tutorial. (n.d.). https://www.w3schools.com/python/default.asp