Python Programming Language

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

facebook.com/cs.MohamedAyman

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 7

Strings

Course Roadmap

Part 2: Python Collections and Strings

Lecture 7: Strings

Lecture 8: Lists

Lecture 9: Tuples

Lecture 10: Dictionaries

Lecture 11: Sets

Lecture 12: Numbers

Lecture Agenda

We will discuss in this lecture the following topics

1- Introduction to String

2- Basic String Operations

3- String Special Operators

4- String Formatting Operator

5- Built-in String Functions


Let's
START UP

Lecture Agenda

Section 1: Introduction to String

Section 2: Basic String Operations

Section 3: String Special Operators

Section 4: String Formatting Operator

Section 5: Built-in String Functions

Introduction to String

- Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable
- Python does not support a character type, these are treated as strings of length one, thus also considered a substring. To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring.

```
x = 'python'
y = x
print(x)                    python
print(y)                    python
x += ' language'
print(x)                    python language
print(y)                    python
```

Python Strings

- String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.
- Python allows either pair of single or double quotes. Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end. The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.
- In Python, Updating or deletion of characters from a String is not allowed. This will cause an error because item assignment or item deletion from a String is not supported. Although deletion of entire String is possible with the use of a built-in del keyword. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name.

Python Strings

Example:

```
x = "Hello Programming"
print(x)
print(len(x))
print(x[2])
print(x[2:9])
print(x[:4])
print(x[11:])
print(x[-1])
print(x[-5:])
print(x[:-7])
print(x[-8:-2])
print(x[4:-2])
print(x[-8:14])
print(x[:])
```

Output:

```
Hello Programming
17
l
llo Pro
Hell
amming
g
mming
Hello Prog
grammi
o Programmi
gramm
Hello Programming
```

H	e	l	l	o		P	r	o	g	r	a	m	m	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Lecture Agenda

- ✔ Section 1: Introduction to String
- **Section 2: Basic String Operations**
- Section 3: String Special Operators
- Section 4: String Formatting Operator
- Section 5: Built-in String Functions

Basic String Operations

- 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).
- In Python, Updating or deletion of characters from a String is not allowed. This will cause an error because item assignment or item deletion from a String is not supported. Although deletion of entire String is possible with the use of a built-in del keyword. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name.
- You can update an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether.

Python Strings

Example:

```
x = "Hello Python"
print(x)
print(len(x))
print(x[2:5] + x[9:12])
print(x[:6] + 'World')
print(x * 2)
print(x[:5] * 3)
print(2 * x[6:])
print((x[6:8] + ' ') * 3)
```

Output:

```
Hello Python
12
llohon
Hello World
Hello PythonHello Python
HelloHelloHello
PythonPython
Py Py Py
```

H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Python Strings

Example:

```
x = 'Hello World'
print(x)
x = x[:6] + 'Python'
print(x)
x = 'Hi' + x[5:]
print(x)
x = x[:3] + 'everyone in ' + x[3:]
print(x)
x = x[:12] + 'for' + x[14:]
print(x)
x = x[:3] + x[-6:]
print(x)
x = x + '!'
print(x)
```

Output:

```
Hello World

Hello Python

Hi Python

Hi everyone in Python

Hi everyone for Python

Hi Python

Hi Python!
```

Lecture Agenda

- ✔ Section 1: Introduction to String
- ✔ Section 2: Basic String Operations
- **Section 3: String Special Operators**
- Section 4: String Formatting Operator
- Section 5: Built-in String Functions

String Special Operators

- There are many operations that can be performed with string which makes it one of the most used data types 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.
- The [in] membership operator returns True if a substring exists in the given string. The [not in] membership operator returns True if a substring does not exist in the given string.
- You can reverse order of string by using this special operator [: : -1]

String Special Operators

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give - HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1

Python Strings

Example:

```
x = 'Hello Python'
print(x)
print(x[2:5] + x[9:12])
print(x[:6] + 'World')
print(x * 2)
print(x[:5] * 3)
print(2 * x[6:-1])
print((x[6:8] + ' ') * 3)
print('P' in x)
print('p' not in x)
print(x[::-1])
```

Output:

```
Hello Python
llohon
Hello World
Hello PythonHello Python
HelloHelloHello
PythonPython
Py Py Py
True
True
nohtyP olleH
```

Lecture Agenda

- ✔ Section 1: Introduction to String
- ✔ Section 2: Basic String Operations
- ✔ Section 3: String Special Operators
- **Section 4: String Formatting Operator**
- Section 5: Built-in String Functions

String Formatting Operator

- One of Python coolest features is the string format operator %. This operator is unique to strings.
- Python uses C style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".
- Simple positional formatting is probably the most common use-case. Use it if the order of your arguments is not likely to change and you only have very few elements you want to concatenate. Since the elements are not represented by something as descriptive as a name this simple style should only be used to format a relatively small number of elements.

String Formatting Operator

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%d	signed decimal integer
%u	unsigned decimal integer
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number

String Formatting Operator

Example:

```
x = 87.654321
print('The value of x is %.0f' % x)
print('The value of x is %.1f' % x)
print('The value of x is %.2f' % x)
print('The value of x is %.3f' % x)
print('The value of x is %.4f' % x)
```

Output:

```
The value of x is 88
The value of x is 87.7
The value of x is 87.65
The value of x is 87.654
The value of x is 87.6543
```

String Formatting Operator

Example:

```
print('we print an integer %d in a string' % (5))
print('we print two integers %d and %d in a string' % (6, 8))
```

Output:

```
we print an integer 5 in a string
we print two integers 6 and 8 in a string
```

Example:

```
print('we print a float %.2f in a string' % (5.2))
print('we print two floats %.2f and %.2f in a string' % (5.2, 7.3))
```

Output:

```
we print a float 5.20 in a string
we print two floats 5.20 and 7.30 in a string
```

String Formatting Operator

Example:

```
print('we print a character %c in a string' % ('e'))
print('we print two characters %c and %c in a string' % ('e', 'i'))
```

Output:

```
we print a character e in a string
we print two characters e and i in a string
```

Example:

```
print('we print a string %s in a string' % ('hello python'))
print('we print two strings %s and %s in a string' % ('hello', 'world'))
```

Output:

```
we print a string hello python in a string
we print two strings hello and world in a string
```

String Formatting Operator

Example:

```
# Default Order
x = '{} is a {} {}'.format('python', 'programming', 'language')
print(x)

# Positional Formatting
x = '{1} is a {0} {2}'.format('programming', 'python', 'language')
print(x)

# Keyword Formatting
x = '{a} is a {b} {c}'.format(c = 'language', b = 'programming', a = 'python')
print(x)
```

Output:

```
python is a programming language
python is a programming language
python is a programming language
```

Triple Quotes

- Python triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINEs, TABs, and any other special characters
- The syntax for triple quotes consists of three consecutive single or double quotes

```
x = '''this is a long string that is made up of several lines and non-printable
charachters such as TAB (\t\t) and they will show up that way when displayed.
NEWLINES within the string, whether explicitly given like this within the
brackets [\n], or just a NEWLINE within the variable assignment will also show up.
'''
print(x)
```

```
this is a long string that is made up of several lines and non-printable
charachters such as TAB (        ) and they will show up that way when displayed.
NEWLINES within the string, whether explicitly given like this within the
brackets [
], or just a NEWLINE within the variable assignment will also show up.
```

Lecture Agenda

- ✔ Section 1: Introduction to String
- ✔ Section 2: Basic String Operations
- ✔ Section 3: String Special Operators
- ✔ Section 4: String Formatting Operator
- Section 5: Built-in String Functions

Built-in String Functions

1- len() Method
2- split() Method
3- max(), min() Methods
4- count() Method
5- replace() Method
6- find(), rfind() Methods
7- index(), rindex() Methods
8- lower(), upper() Methods
9- startswith(), endswith() Methods
10- isalpha(), isnumeric(), isalnum() Methods
11- islower(), isupper() Methods
12- swapcase() Method
13- lstrip(), rstrip(), strip() Methods
14- ljust(), rjust(), center() Methods
15- join() Method
16- enumerate() Method

len() Method

Example:

```
x = 'python'
print(x)
print(len(x))

x = 'hello python'
print(x)
print(len(x))
```

Output:

```
python
6

hello python
12
```

split() Method

Example:

```
x = 'this is string example...wow!!'
print(x)
print(x.split())
print(x.split('i'))
print(x.split('i', 2))
print(x.split('str'))
```

Output:

```
this is string example...wow!!
['this', 'is', 'string', 'example...wow!!']
['th', 's ', 's str', 'ng example...wow!!']
['th', 's ', 's string example...wow!!']
['this is ', 'ing example...wow!']
```

max(), min() Methods

Example:

```
a = 'this is my string example...wow!!'
print(max(a))
b = 'this is string example...wow!!'
print(max(b))
c = 'this is string...wow!!'
print(max(c))

print(max(a, b, c))
```

Output:

```
y

x

w

this is string example...wow!!
```

max(), min() Methods

Example:

```
a = 'this_is_my_string_example...wow!!'
print(min(a))
b = 'this_is_string_example...wow'
print(min(b))
c = 'this_is_string_wow'
print(min(c))

print(min(a, b, c))
```

Output:

```
!
.
_
this_is_my_string_example...wow!!
```

count() Method

Example:

```
x = 'this is string example'
print(x)
print(x.count('i'))
print(x.count('i', 4))
print(x.count('i', 4, 7))
print(x.count('is'))
print(x.count('is', 2))
print(x.count('is', 2, 7))
print(x.count('is', 2, 6))
```

Output:

```
this is string example
3
2
1
2
2
2
1
```

t	h	i	s		i	s		s	t	r	i	n	g		e	x	a	m	p	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

replace() Method

Example:

```
x = 'this is string example'
print(x)
print(x.replace('i', 'e'))
print(x.replace('i', 'e', 2))
print(x.replace('is', 'was'))
print(x.replace('is', 'was', 1))
```

Output:

```
this is string example
thes es streng example
thes es string example
thwas was string example
thwas is string example
```

find(), rfind() Methods

Example:

```
x = 'this is string example'
print(x)
print(x.find('i'))
print(x.find('i', 3))
print(x.find('i', 7, 20))
print(x.find('i', 12, 22))
print(x.find('is'))
print(x.find('is', 3))
print(x.find('is', 8, 20))
```

Output:

```
this is string example
2
5
11
-1
2
5
-1
```

t	h	i	s		i	s		s	t	r	i	n	g		e	x	a	m	p	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

find(), rfind() Methods

Example:

```
x = 'this is string example'
print(x)
print(x.rfind('i'))
print(x.rfind('i', 12))
print(x.rfind('i', -11))
print(x.rfind('i', 3, 9))
print(x.rfind('i', 0, 4))
print(x.rfind('is'))
print(x.rfind('is', 7))
print(x.rfind('is', 2, 5))
```

Output:

```
this is string example
11
-1
11
5
2
5
-1
2
```

t	h	i	s		i	s		s	t	r	i	n	g		e	x	a	m	p	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

index(), rindex() Methods

Example:

```
x = 'this is string example'
print(x)
print(x.index('i'))
print(x.index('i', 3))
print(x.index('i', 7, 20))
print(x.index('i', 12, 22))


print(x.index('is'))
print(x.index('is', 3))
print(x.index('is', 8, 20))
```

Output:

```
this is string example
2
5
11
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    print(x.index('i', 12, 22))
ValueError: substring not found

2
5
Traceback (most recent call last):
  File "main.py", line 11, in <module>
    print(x.index('is', 8, 20))
ValueError: substring not found
```

index(), rindex() Methods

Example:

```
x = 'this is string example'
print(x)
print(x.rindex('i'))
print(x.rindex('i', 12))

print(x.rindex('i', -11))
print(x.rindex('i', 3, 9))
print(x.rindex('i', 0, 4))

print(x.rindex('is'))
print(x.rindex('is', 7))

print(x.rindex('is', 2, 5))
```

Output:

```
this is string example
11
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    print(x.rindex('i', 12))
ValueError: substring not found
11
5
2

5
Traceback (most recent call last):
  File "main.py", line 10, in <module>
    print(x.rindex('is', 7))
ValueError: substring not found
2
```

lower(), upper() Methods

Example:

```
x = 'EXAMPLE'
print(x)
print(x.lower())
x = 'STRING EXAMPLE'
print(x)
print(x.lower())
x = 'STRING_EXAMPLE'
print(x)
print(x.lower())
x = 'EXAMPLE123'
print(x)
print(x.lower())
x = '123'
print(x)
print(x.lower())
x = 'Example'
print(x)
print(x.lower())
```

Output:

```
EXAMPLE
example

STRING EXAMPLE
string example

STRING_EXAMPLE
string_example

EXAMPLE123
example123

123
123

Example
example
```

lower(), upper() Methods

Example:

```
x = 'example'
print(x)
print(x.upper())
x = 'string example'
print(x)
print(x.upper())
x = 'string_example'
print(x)
print(x.upper())
x = 'example123'
print(x)
print(x.upper())
x = '123'
print(x)
print(x.upper())
x = 'Example'
print(x)
print(x.upper())
```

Output:

```
example
EXAMPLE
string example
STRING EXAMPLE
string_example
STRING_EXAMPLE
example123
EXAMPLE123
123
123
Example
EXAMPLE
```

Practice

Problems

1- Implement a function which count the number of characters in a string
2- Implement a function which splits a string with a given character and return the result into a list
3- Implement a function which determines the character of maximum ASCII code in a string
4- Implement a function which determines the character of minimum ASCII code in a string
5- Implement a function which counts the number of occurrences of a substring into a given string
6- Implement a function which finds all indices of a substring into a given string in ascending order
7- Implement a function which finds all indices of a substring into a given string in descending order
8- Implement a function which replaces all old substring with a new substring into a given string
9- Implement a function which converts a string into lower representation
10- Implement a function which converts a string into upper representation
11- Implement a function which counts the number of vowel characters in a string
12- Implement a function which replaces any vowel character in a string with dot
13- Implement a function which finds all indices of the vowel characters in a string (ascending/descending order)
14- Implement a function which counts the number of lower and upper characters in a string
15- Implement a function which divides the ip address into 4 numbers

Built-in String Functions

1- len() Method

2- split() Method

3- max(), min() Methods

4- count() Method

5- replace() Method

6- find(), rfind() Methods

7- index(), rindex() Methods

8- lower(), upper() Methods

9- startswith(), endswith() Methods

10- isalpha(), isnumeric(), isalnum() Methods

11- islower(), isupper() Methods

12- swapcase() Method

13- lstrip(), rstrip(), strip() Methods

14- ljust(), rjust(), center() Methods

15- join() Method

16- enumerate() Method

startswith(), endswith() Methods

Example:

```
x = 'this is string example'
print(x)
print(x.startswith('this'))
print(x.startswith('is'))
print(x.startswith('is', 2))
print(x.startswith('is', 5, 10))
print(x.endswith('example'))
print(x.endswith('exam'))
print(x.endswith('exam', 0, -3))
print(x.endswith('is', 0, 7))
```

Output:

```
this is string example
True
False
True
True
True
False
True
True
```

t	h	i	s		i	s		s	t	r	i	n	g		e	x	a	m	p	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

isalpha(), isnumeric(), isalnum() Methods

Example:

```
x = 'example'
print(x.isalpha())
print(x.isnumeric())
print(x.isalnum())

x = 'string example'
print(x.isalpha())
print(x.isnumeric())
print(x.isalnum())

x = 'string_example'
print(x.isalpha())
print(x.isnumeric())
print(x.isalnum())
```

Output:

```
True
False
True

False
False
False

False
False
False
```

isalpha(), isnumeric(), isalnum() Methods

Example:

```
x = 'example123'
print(x.isalpha())
print(x.isnumeric())
print(x.isalnum())

x = '123'
print(x.isalpha())
print(x.isnumeric())
print(x.isalnum())

x = '. !@#$%'
print(x.isalpha())
print(x.isnumeric())
print(x.isalnum())
```

Output:

```
False
False
True

False
True
True

False
False
False
```

islower(), isupper() Methods

Example:

```
x = 'example'
print(x.islower())
print(x.isupper())

x = 'EXAMPLE'
print(x.islower())
print(x.isupper())

x = 'string example'
print(x.islower())
print(x.isupper())

x = 'STRING EXAMPLE'
print(x.islower())
print(x.isupper())

x = 'string_example'
print(x.islower())
print(x.isupper())
```

Output:

```
True
False

False
True

True
False

False
True

True
False
```

islower(), isupper() Methods

Example:

```
x = 'STRING_EXAMPLE'
print(x.islower())
print(x.isupper())

x = 'example123'
print(x.islower())
print(x.isupper())

x = 'EXAMPLE123'
print(x.islower())
print(x.isupper())

x = '123'
print(x.islower())
print(x.isupper())

x = 'Example'
print(x.islower())
print(x.isupper())
```

Output:

```
False
True

True
False

False
True

False
False

False
False
```

swapcase() Method

Example:

```
x = 'example'
print(x.swapcase())

x = 'EXAMPLE'
print(x.swapcase())

x = 'string example'
print(x.swapcase())

x = 'STRING EXAMPLE'
print(x.swapcase())

x = 'string_example'
print(x.swapcase())
```

Output:

```
EXAMPLE

example

STRING EXAMPLE

string example

STRING_EXAMPLE
```

swapcase() Method

Example:

```
x = 'STRING_EXAMPLE'
print(x.swapcase())

x = 'example123'
print(x.swapcase())

x = 'EXAMPLE123'
print(x.swapcase())

x = '123'
print(x.swapcase())

x = 'Example'
print(x.swapcase())
```

Output:

```
string_example

EXAMPLE123

example123

123

eXAMPLE
```

lstrip(), rstrip(), strip() Methods

Example:

```
x = '        this is string example        '
print('|' + x + '|')
print('|' + x.lstrip() + '|')
print('|' + x.rstrip() + '|')
print('|' + x.strip() + '|')

x = '$$$$$$$$this is string example$$$$$$$$'
print('|' + x + '|')
print('|' + x.lstrip('$') + '|')
print('|' + x.rstrip('$') + '|')
print('|' + x.strip('$') + '|')

x = '^!^*!**^this is string example^!^*!**^'
print('|' + x + '|')
print('|' + x.lstrip('!*^') + '|')
print('|' + x.rstrip('!*^') + '|')
print('|' + x.strip('!*^') + '|')
```

Output:

```
|        this is string example        |
|this is string example        |
|        this is string example|
|this is string example|

|$$$$$$$$this is string example$$$$$$$$|
|this is string example$$$$$$$$|
|$$$$$$$$this is string example|
|this is string example|

|^!^*!**^this is string example^!^*!**^|
|this is string example^!^*!**^|
|^!^*!**^this is string example|
|this is string example|
```

ljust(), rjust(), center() Methods

Example:

```
x = 'this is string example'
print(len(x))
print('|' + x + '|')
print('|' + x.ljust(30)       + '|')
print('|' + x.ljust(30, '$')  + '|')
print('|' + x.rjust(30)       + '|')
print('|' + x.rjust(30, '$')  + '|')
print('|' + x.center(30)      + '|')
print('|' + x.center(30, '$') + '|')
print('|' + x.center(29, '@') + '|')
```

Output:

```
22
|this is string example|
|this is string example        |
|this is string example$$$$$$$$|
|        this is string example|
|$$$$$$$$this is string example|
|    this is string example    |
|$$$$this is string example$$$$|
|@@@@this is string example@@@|
```

join() Method

Example:

```
x = 'python'
print(x)
print('$'.join(x))
x = ['ab', 'cd', 'ef']
print(x)
print('*|*'.join(x))
x = ('3.5', '7.1', '4.8')
print(x)
print('@'.join(x))
x = {'a':1, 'b':2, 'c':3}
print(x)
print('%^'.join(x))
x = {'e1', 'e2', 'e3'}
print(x)
print('!'.join(x))
```

Output:

```
python
p$y$t$h$o$n

['ab', 'cd', 'ef']
ab*|*cd*|*ef

('3.5', '7.1', '4.8')
3.5@7.1@4.8

{'a': 1, 'b': 2, 'c': 3}
a%^b%^c

{'e2', 'e1', 'e3'}
e2!e1!e3
```

enumerate() Method

Example:

```
x = 'python'
print(x)

e = enumerate(x)
print(type(e))
print(list(e))

e = enumerate(x, 3)
print(type(e))
print(list(e))
```

Output:

```
python

<class 'enumerate'>
[(0, 'p'), (1, 'y'), (2, 't'), (3, 'h'), (4, 'o'), (5, 'n')]

<class 'enumerate'>
[(3, 'p'), (4, 'y'), (5, 't'), (6, 'h'), (7, 'o'), (8, 'n')]
```

enumerate() Method

Example:

```
x = 'python'
print(x)

for i in enumerate(x):
    print(i)



for i, j in enumerate(x):
    print(i, j)
```

Output:

```
 python

(0, 'p')
(1, 'y')
(2, 't')
(3, 'h')
(4, 'o')
(5, 'n')
0 p
1 y
2 t
3 h
4 o
5 n
```

Practice

Problems

1- Implement a function which checks if a string is starts with a given substring or not
2- Implement a function which checks if a string is ends with a given substring or not
3- Implement a function which checks if a string is numeric or not
4- Implement a function which checks if a string is alphabets or not
5- Implement a function which checks if a string is alphabets and numeric or not
6- Implement a function which checks if a sting is lower case or not
7- Implement a function which checks if a string is upper case or not
8- Implement a function which convert the upper letters to lower case and the lower letters to upper case
9- Implement a function which takes a string and a sequence as parameters, then joins the given string between all items of the given sequence
10- Implement a function which takes a sequence as a parameter, then return a list of tuples (index, item)
11- Implement a function which validates website address which start with http or www and end with com
12- Implement a function which validates email address which contain only characters and digits
13- Implement a function which validates password which contain lower and upper characters and digits and special characters with length 8 characters or more

Built-in String Functions

1- len() Method

2- split() Method

3- max(), min() Methods

4- count() Method

5- replace() Method

6- find(), rfind() Methods

7- index(), rindex() Methods

8- lower(), upper() Methods

9- startswith(), endswith() Methods

10- isalpha(), isnumeric(), isalnum() Methods

11- islower(), isupper() Methods

12- swapcase() Method

13- lstrip(), rstrip(), strip() Methods

14- ljust(), rjust(), center() Methods

15- join() Method

16- enumerate() Method

Lecture Agenda

- ✔ Section 1: Introduction to String
- ✔ Section 2: Basic String Operations
- ✔ Section 3: String Special Operators
- ✔ Section 4: String Formatting Operator
- ✔ Section 5: Built-in String Functions


DO
MORE.