Python Course - Texts in Python

Digital Career Institute

DCI Digital Career Institute

Goal of the Submodule

The goal of this submodule is to help the learners work with texts in Python. By the end of this submodule, the learners should be able to understand:

- Different types of texts used in Python.
- The easy to handle and work with text including text concatenation and modification.
- How to use the String method.
- Learning the concepts of regular expressions and patterns in Python.

Topics

- **Introduction to Texts in Python**
- **Difference between characters and strings**
- **Working with Strings:**
  - Length and concat methods
  - Concat
- **Modifying Strings:**
  - Using a variety of methods including the strip, upper, lower, split and other methods
- **Using the String methods:**
  - Using a variety of methods including find, replace, split and other methods
- **Using regular Expressions with Python**
  - Introduction to basic Python Regular Expressions to search and replace text

Glossary

Term	Definition
Char	A char is a single character, that is a letter, a digit, a punctuation mark, a tab, a space or something similar.
String	String is a sequence of characters, for e.g. “Hello” is a string of 5 characters.
Built-in method	A ready made functions to be used in Python

Introduction to Python Strings

Introduction to Strings

A **String** is a Python data type to store textual data. **String** characteristics include:

- It is a collection of characters stored in an array format
- We use double quotes to create a **String**
    - If we want to span a String in multiple lines we can use the triple quote, this is ideal for long text.
    - Special characters like Tabs or Newlines can also be used within the triple quotes.
- We can use the backslash character (\) to escape quotes in **Strings**
- **Strings** are arrays of characters, thus we can slice them using the brackets and the character index locations

Difference between characters and Strings

Difference Between Characters and Strings

Character	String
A **character** is a single letter, number, punctuation mark or symbol.	A **string** is a one-dimensional array of characters terminated by a null character.
Character is an element.	A string is a set of characters.
Single or double quotes are used to represent a character.	Single or double quotes are used to represent a string.
Character refers to a single letter, number, space.	String refers to a set of characters.

Character Encoding Systems

String Encoding

Have you ever asked yourself how a character is stored in a computer?

- As everything in a computer, characters are represented by numbers.
- Since this is the case, we need to agree on what number represents which character.
- The set of pairs, number/character is what we call **Encoding**.

Stop for a while, think, how would you do that?

String Encoding

A simple table depicting the character and the corresponding numbers is the solution.

It could have started easier, if different companies did not decided to create its own table.

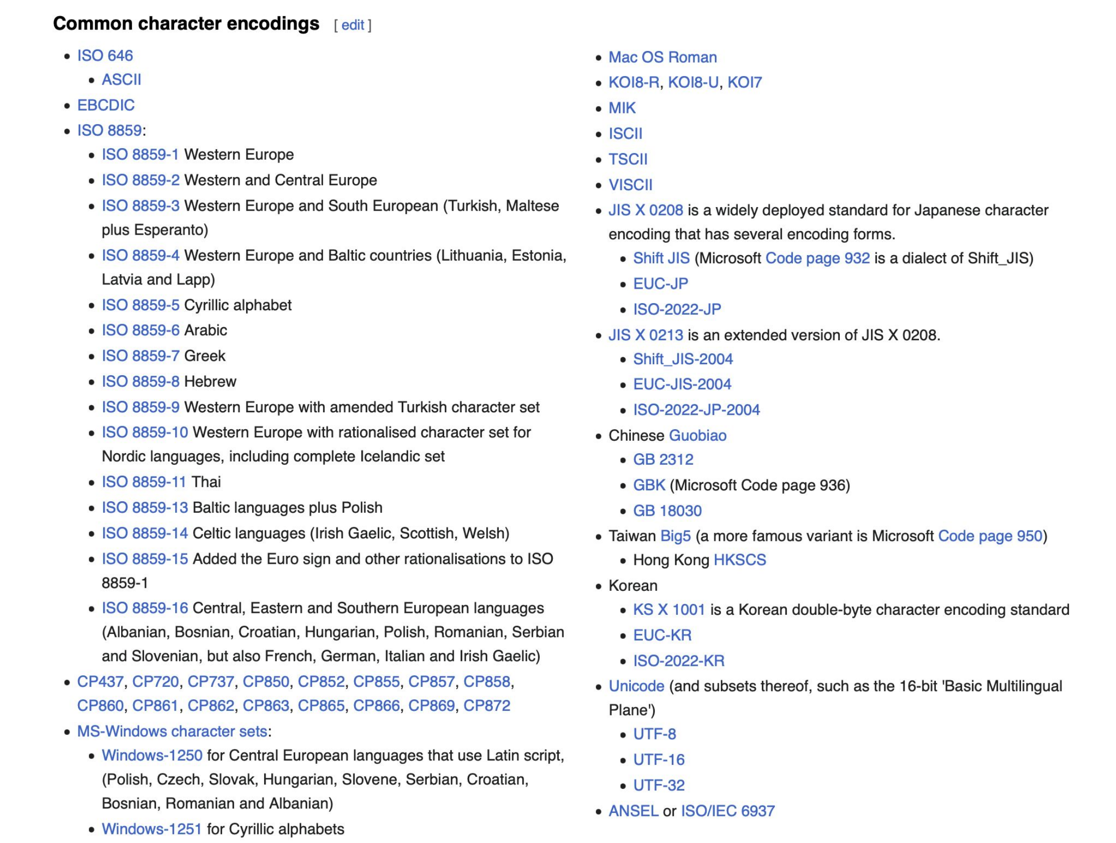
Check this article…

The Chaos that is Character Encodings

https://medium.com/@zwork101/the-chaos-that-is-character-encodings-d79111a45e1d

String Encoding

List of some encodings...



Common character encodings [edit]

- ISO 646
  - ASCII
- EBCDIC
- ISO 8859:
  - ISO 8859-1 Western Europe
  - ISO 8859-2 Western and Central Europe
  - ISO 8859-3 Western Europe and South European (Turkish, Maltese plus Esperanto)
  - ISO 8859-4 Western Europe and Baltic countries (Lithuania, Estonia, Latvia and Lapp)
  - ISO 8859-5 Cyrillic alphabet
  - ISO 8859-6 Arabic
  - ISO 8859-7 Greek
  - ISO 8859-8 Hebrew
  - ISO 8859-9 Western Europe with amended Turkish character set
  - ISO 8859-10 Western Europe with rationalised character set for Nordic languages, including complete Icelandic set
  - ISO 8859-11 Thai
  - ISO 8859-13 Baltic languages plus Polish
  - ISO 8859-14 Celtic languages (Irish Gaelic, Scottish, Welsh)
  - ISO 8859-15 Added the Euro sign and other rationalisations to ISO 8859-1
  - ISO 8859-16 Central, Eastern and Southern European languages (Albanian, Bosnian, Croatian, Hungarian, Polish, Romanian, Serbian and Slovenian, but also French, German, Italian and Irish Gaelic)
- CP437, CP720, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872
- MS-Windows character sets:
  - Windows-1250 for Central European languages that use Latin script, (Polish, Czech, Slovak, Hungarian, Slovene, Serbian, Croatian, Bosnian, Romanian and Albanian)
  - Windows-1251 for Cyrillic alphabets
- Mac OS Roman
- KOI8-R, KOI8-U, KOI7
- MIK
- ISCII
- TSCII
- VISCII
- JIS X 0208 is a widely deployed standard for Japanese character encoding that has several encoding forms.
  - Shift JIS (Microsoft Code page 932 is a dialect of Shift_JIS)
  - EUC-JP
  - ISO-2022-JP
- JIS X 0213 is an extended version of JIS X 0208.
  - Shift_JIS-2004
  - EUC-JIS-2004
  - ISO-2022-JP-2004
- Chinese Guobiao
  - GB 2312
  - GBK (Microsoft Code page 936)
  - GB 18030
- Taiwan Big5 (a more famous variant is Microsoft Code page 950)
  - Hong Kong HKSCS
- Korean
  - KS X 1001 is a Korean double-byte character encoding standard
  - EUC-KR
  - ISO-2022-KR
- Unicode (and subsets thereof, such as the 16-bit 'Basic Multilingual Plane')
  - UTF-8
  - UTF-16
  - UTF-32
- ANSEL or ISO/IEC 6937

String Encoding - ASCII

Why so many?

- Languages can have different characters.
- **ASCII standard**, one of the most widely used encoding systems in the past, only supported english latin alphabet and general symbols.
- What about japanese, chinese, arabic?

Stop for a while, think, how would you do that?

ASCII Encoding

Digital Career Institute

String Encoding - ASCII

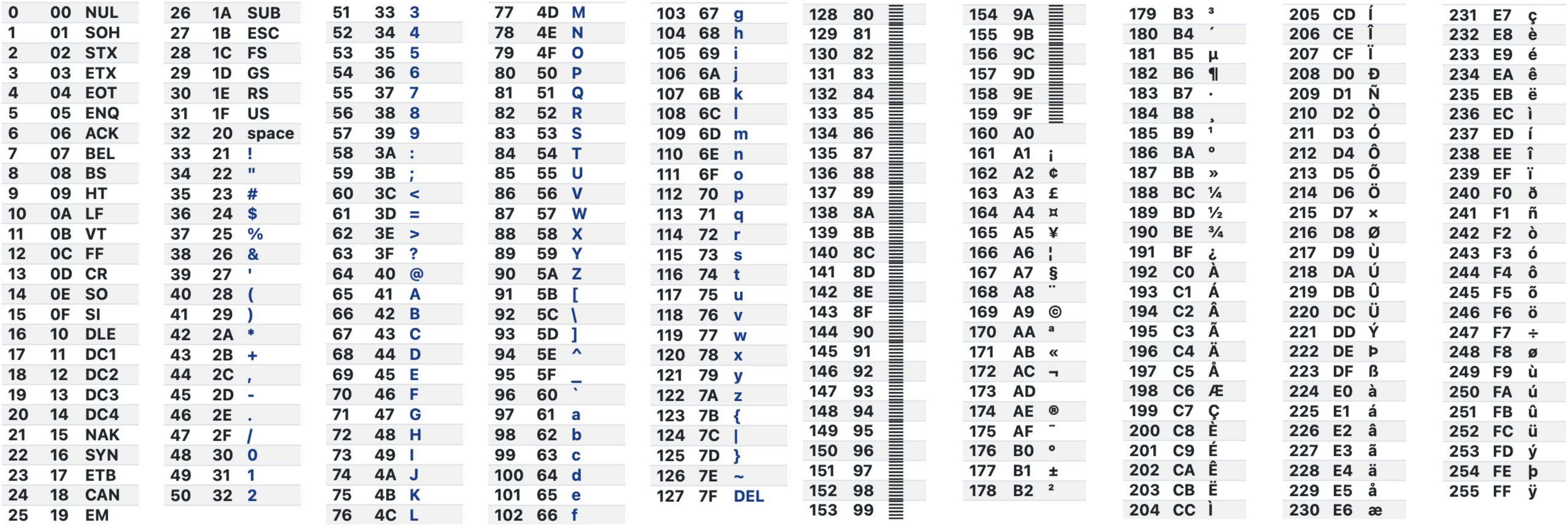
The ASCII Standard

- ASCII stands for American Standard Code for Information Interchange
- It was created by the American National Standards Institute
- It covers the Latin Alphabet and a set of useful symbols,
- It is one byte long, which means that we can have a maximum of 256 characters.
- ASCII was designed to improve performance of some operations when handling characters, ex. Converting Uppercase to Lowercase and vice-versa.
    - Let's take a look at the table and this property...

String Encoding - ASCII

Digital Career Institute

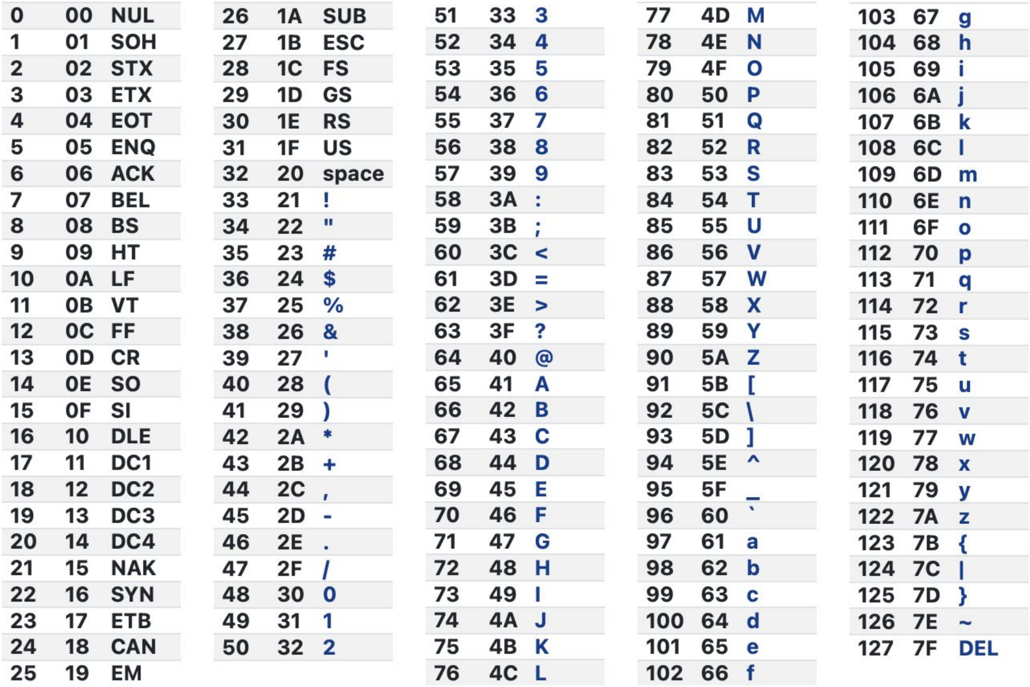
The ASCII Standard Table...



0	00	NUL
1	01	SOH
2	02	STX
3	03	ETX
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	BEL
8	08	BS
9	09	HT
10	0A	LF
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL
128	80	
129	81	
130	82	
131	83	
132	84	
133	85	
134	86	
135	87	
136	88	
137	89	
138	8A	
139	8B	
140	8C	
141	8D	
142	8E	
143	8F	
144	90	
145	91	
146	92	
147	93	
148	94	
149	95	
150	96	
151	97	
152	98	
153	99	
154	9A	
155	9B	
156	9C	
157	9D	
158	9E	
159	9F	
160	A0	
161	A1	¡
162	A2	¢
163	A3	£
164	A4	¤
165	A5	¥
166	A6	¦
167	A7	§
168	A8	¨
169	A9	©
170	AA	ª
171	AB	«
172	AC	¬
173	AD	
174	AE	®
175	AF	¯
176	B0	°
177	B1	±
178	B2	²
179	B3	³
180	B4	´
181	B5	µ
182	B6	¶
183	B7	·
184	B8	¸
185	B9	¹
186	BA	º
187	BB	»
188	BC	¼
189	BD	½
190	BE	¾
191	BF	¿
192	C0	À
193	C1	Á
194	C2	Â
195	C3	Ã
196	C4	Ä
197	C5	Å
198	C6	Æ
199	C7	Ç
200	C8	È
201	C9	É
202	CA	Ê
203	CB	Ë
204	CC	Ì
205	CD	Í
206	CE	Î
207	CF	Ï
208	D0	Ð
209	D1	Ñ
210	D2	Ò
211	D3	Ó
212	D4	Ô
213	D5	Õ
214	D6	Ö
215	D7	×
216	D8	Ø
217	D9	Ù
218	DA	Ú
219	DB	Û
220	DC	Ü
221	DD	Ý
222	DE	Þ
223	DF	ß
224	E0	à
225	E1	á
226	E2	â
227	E3	ã
228	E4	ä
229	E5	å
230	E6	æ
231	E7	ç
232	E8	è
233	E9	é
234	EA	ê
235	EB	ë
236	EC	ì
237	ED	í
238	EE	î
239	EF	ï
240	F0	ð
241	F1	ñ
242	F2	ò
243	F3	ó
244	F4	ô
245	F5	õ
246	F6	ö
247	F7	÷
248	F8	ø
249	F9	ù
250	FA	ú
251	FB	û
252	FC	ü
253	FD	ý
254	FE	þ
255	FF	ÿ

String Encoding - ASCII

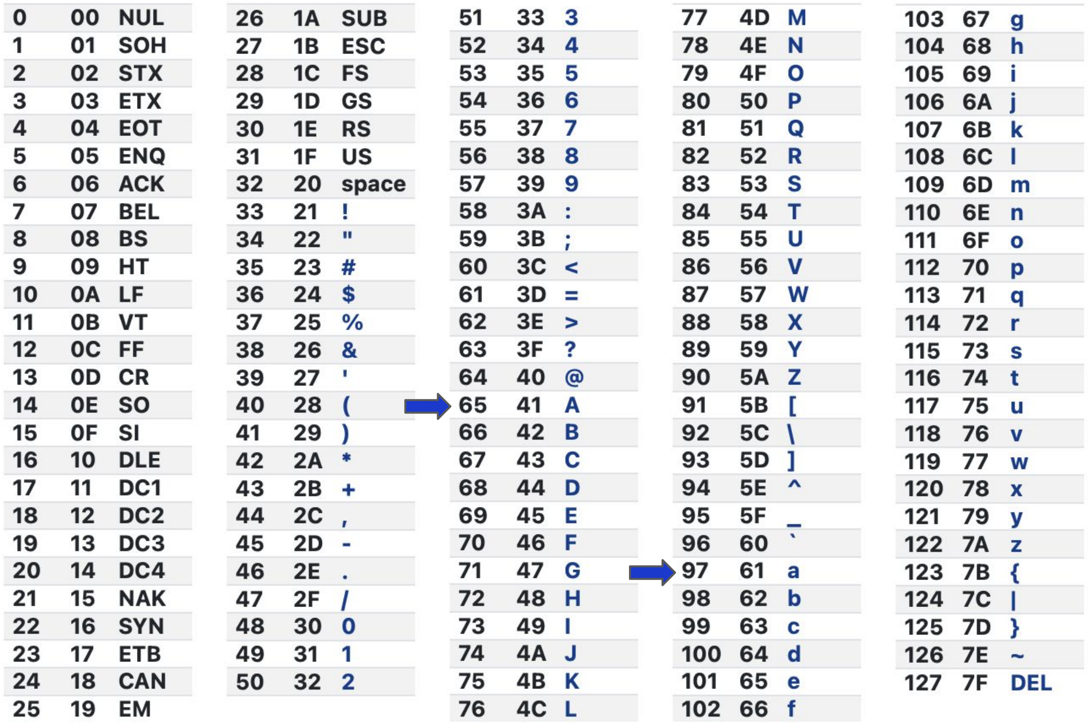
Now... Look at this part of table, try to figure out a pattern between uppercase and lowercase...



0	00	NUL
1	01	SOH
2	02	STX
3	03	ETX
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	BEL
8	08	BS
9	09	HT
10	0A	LF
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

String Encoding - ASCII

So...



0	00	NUL
1	01	SOH
2	02	STX
3	03	ETX
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	BEL
8	08	BS
9	09	HT
10	0A	LF
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

The **difference** between uppercase and lowercase letters **is always 32.**

We can **achieve a conversion** very quickly by doing a binary shift operation, or simple **by adding and removing 32.**

What is a binary shift operation? You just need to know now, that it is fast!

String Encoding - ASCII

If you feel curious and wants to check the engineering behind ASCII in details check the Wiki Article...

https://en.wikipedia.org/wiki/ASCII

String Encoding - ASCII

The ASCII Standard...

- It became really popular despite not supporting many languages, this, due to the lead of the USA in the computer race.
- Multiple extensions of the ASCII were created to support specific sets of characters.
- All these extensions and different encodings made the developer's life really complicated when having to handle multiple encodings.

UNICODE TO THE RESCUE...

Unicode Encoding

String Encoding - UNICODE

The Unicode Standard...

- Unicode changes the size from a 8-bit (1 byte) code to a 16-bit encoding
    - It can also use a 8-bit encoding version...
- Unicode can represent all the characters of all the major languages in the world.
    - Neat! Unicode encodes even the Emojis!
- It also provides an extension mechanism to allow the addition of more characters.
- Unicode provides some level of compatibility with ASCII, the idea was to make it easy to migrate from ASCII to UNICODE.

String Encoding - UNICODE

Unicode Transformation Format - UTF

"A Unicode transformation format (UTF) is an algorithmic mapping from every Unicode code point (except surrogate code points) to a unique byte sequence. The ISO/IEC 10646 standard uses the term “UCS transformation format” for UTF; the two terms are merely synonyms for the same concept."

Source: Unicode.org - https://unicode.org/faq/utf_bom.html

String Encoding - UNICODE

Unicode Transformation Format - UTF

- The most common UTFs are the UTF-8, UTF-16, UTF-32, being the first the most widely used o the Web and in Python.
- The Unicode.org states:
  - *"UTF-8 is most common on the web. UTF-16 is used by Java and Windows (.Net). UTF-8 and UTF-32 are used by Linux and various Unix systems. The conversions between all of them are algorithmically based, fast and lossless. This makes it easy to support data input or output in multiple formats, while using a particular UTF for internal storage or processing."*

Self Study

Investigate the different UTFs and their differences, pros and cons.

- UTF-8
- UTF-16
- UTF-32

Back to Strings… How to create a String?

String characteristics

A string is a series of characters.
To create a String we will need to use single or double quotes

Strings are immutable data, this means that once created we cannot change it, but we can reinitialize it!

When a string reference is reinitialized with a new value, it is creating a new object rather than overwriting the previous value.

Python String Methods

Method	Description
`string.capitalize()`	This method is used to capitalize a text (convert the first character to uppercase)
`string.upper()`	This method is used to transform a text into an **upper** (uppercase) text
`string.lower()`	This method is used to transform a text into a **lower** (lowercase) text

⤇ The methods do not accept any arguments, but they can be used using the dot operator

Python String Methods

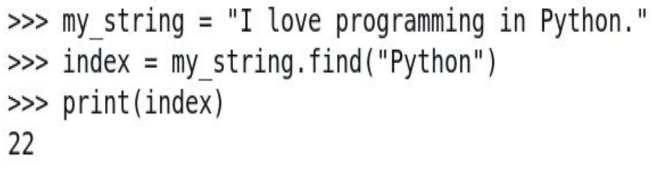
Method	Description
`string.isalpha()`	This method is used to check if all the characters in the text are letters
`string.isdecimal()`	This method is used to check if all the characters in the text are decimals
`string.isnumeric()`	This method is used to check if all the characters in the text are numbers

- ⤇ The methods do not accept any arguments, but they can be used using the dot operator

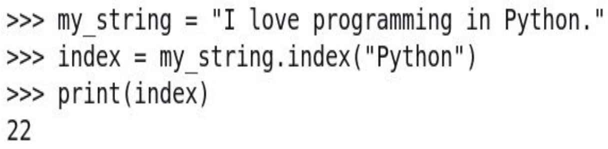
Python String Methods

Method	Description
`string.find(value,start,end)`	This method is used to find a text inside the String. This is ideal when we have a phrase. The `value` is required, while the `start`/`end` are optional and denote the index of the String positions.
`string.index(value,start,end)`	This method is used to finds the first occurrence of the specified value and return its index. This works similar to find, but it returns an exception if the value is not found, so we need to handle it with a catch statement.

Python String Methods



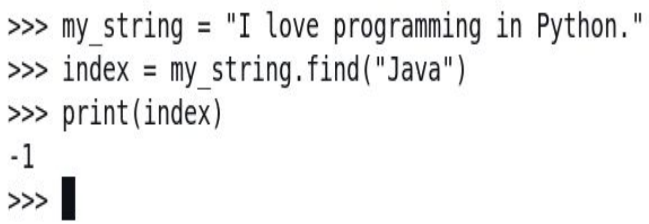
```
>>> my_string = "I love programming in Python."
>>> index = my_string.find("Python")
>>> print(index)
22
```



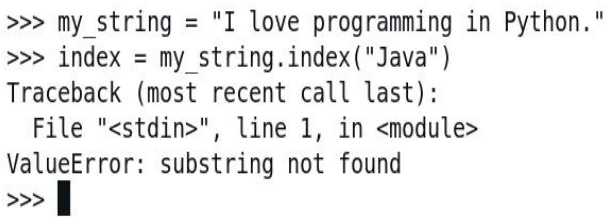
```
>>> my_string = "I love programming in Python."
>>> index = my_string.index("Python")
>>> print(index)
22
```

- The string methods `find()` and `index()` are returning the first occurrence of the substring **“Python”** in **my_string**

Python String Method



```
>>> my_string = "I love programming in Python."
>>> index = my_string.find("Java")
>>> print(index)
-1
>>> 
```



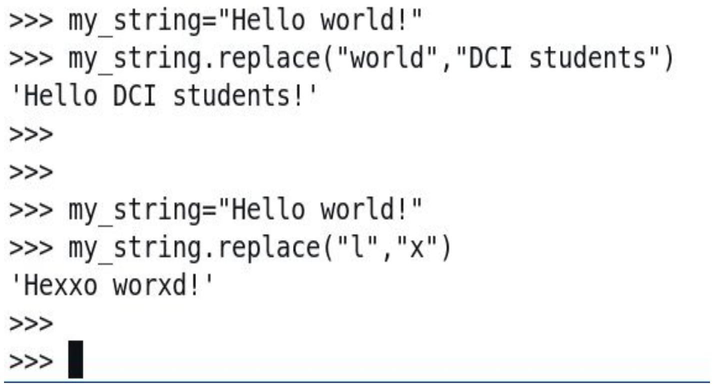
```
>>> my_string = "I love programming in Python."
>>> index = my_string.index("Java")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> 
```

- When the substring is not found `find()` will return **-1**. However `index()` will raise the error **ValueError** with the message **“substring not found”**

Python String Methods

Method	Description
`string.split(separator, maxsplit)`	This method is used to split a string in a list of words. The `separator` defines the base String to split with (e.g. dash), the whitespace is default. The `maxsplit` refers to how many splits to do and it is optional.
`string.replace(oldvalue, newvalue, count)`	This method is used to replace a given String with a another String. The `oldvalue` and `newvalue` are required. The `count` is optional and specifies how many occurrences of the given value we want to replace.

Python String Methods

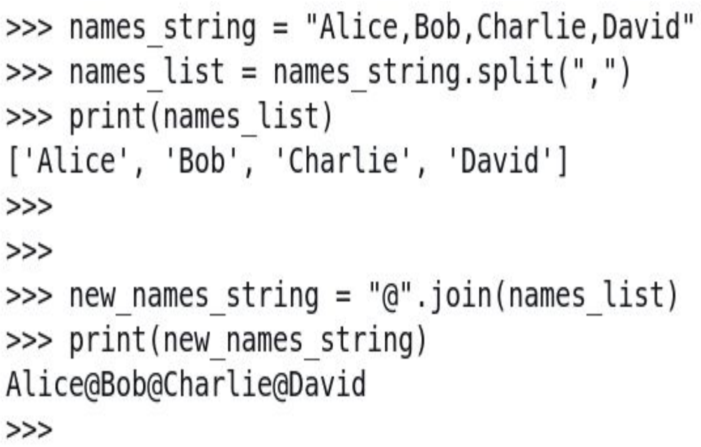


```
>>> my_string="Hello world!"
>>> my_string.replace("world","DCI students")
'Hello DCI students!'
>>>
>>>
>>> my_string="Hello world!"
>>> my_string.replace("l","x")
'Hexxo worxd!'
>>>
>>> 
```

- In **my_string** we replaced The substring **“world”** with the substring “**DCI students”.**

- In **my_string** we replaced every **“l”** in **“Hello world!”** with **“x”.**

Python String Methods



```
>>> names_string = "Alice,Bob,Charlie,David"
>>> names_list = names_string.split(",")
>>> print(names_list)
['Alice', 'Bob', 'Charlie', 'David']
>>>
>>>
>>> new_names_string = "@".join(names_list)
>>> print(new_names_string)
Alice@Bob@Charlie@David
>>>
```

- The `split()` method here splitted the string **“names_string”** and the delimiter used here is **“,”**. The output is stored in a list called **“names_list”** .

- The `join()` method joined all the names in **“names_list”** into a single string **“new_names_string”** separated by **“@”**

At the core of the lesson

Lessons Learned:

- We know what is String and why it is immutable
- We know how to create a String and what is the difference between Strings and Characters in Python
- We know how to use different String manipulation methods including:
  - capitalize()
  - upper()
  - lower()
  - isalpha()
  - isdecimal()
  - isnumeric()
  - find()
  - index()
  - split()
  - replace()