M3 – Self preparation

In this Module we will explore string methods and operations.

Try the following in the interaction window:

```
s = "abc"
m = "12345"
s + m
"" + "cs 116"
```

Write down the purpose of + used with two strings: (Note that strings values are presented in the interaction window with single quotation marks.

Repeat the same exercise while using single quotation marks instead of double quotation marks.

Why and when we would need to use both?

Try the following in the interaction window:

```
"$" * 3
5 * "wow"
"cs" * "116"
0 * "anc"
-5 * "yes"
```

Write down the purpose of * :

Note: * usually used with a string and a natural number.

Try the following in the interaction window:

```
"abc" in "myabcstring"
"abc" in "myab-cstring"
"ab*" in "ab*"
"" in "abc!"
"  " in "abc"
```

Write down the purpose of `in` :

Try the following in the interaction window:

```
len("")
len("CS116!")
len("cs 116")
```

Write down the purpose of `len`:

Try the following in the definition window:

```
s= "abcde"
s[2:4]
s[0:3]
s[1:10]
s[3:3]
s[4:1]
s[:3]
s[2:]
s[8:]
```

Write down the purpose of `s[i:j]` :

Try the following in the interaction window:

```
s = "abcdefghjklmn"
s[1:8:2]
s[8:1:-2]
s[5::3]
s[:9:4]
s[::-1]
```

Write down the purpose of `s[i:j:k]`:

Try the following in the interaction window:

```
s = "abcde"
s[0]
s[8]
s[3]
s[-1]
s[-2]
s[-7]
```

Write down the purpose of `s[i]`:

Try the following in the interaction window:

```
s = ""
```

```
s[0]
s[0:1]
```

You will take advantage of this while solving problems so remember this ☺

Try the following in the interaction window:

```
s = "abcde"
```

```
s[3] = "*"
```

```
s = s[:3] + "*" + s[4:]
```

```
s
```

Can we change the individual characters in a string?

Note that in the example about we had to create a **new** string with the change

---

Methods in Python.

Review M3 slide 7 before you move to the next page.

Try each of the following expressions in the interaction window and write down its purpose:

```
s = ”aBcd345!”
s.upper()
s.lower()
s.islower()
s[2:4].islower()
s[2:6].islower()
"AB##".isupper()
"".isupper()
"abcad".count("a")
s.count("a")
s.startswith("aB")
s.endswith("345!")
s.endswith("")
s="abcde 1 2   3 ab "
s.find("a")
s.find("a",1)
s.split()
s.split("a")
s.rfind("a")
s.strip()
```

Purposes:

upper():

lower():

islower():

isupper():

count():

startswith():

endswith():

find():

split()

rfind():

strip():

Note1: Make sure, for each string method, you try it with different strings and different length (including empty strings).

Note2: compare your definition for each function with the formal definition. For example: after you write your definition for `count`, write in the interaction window `help(str.count)` and compare.

Note3: write `dir(str)` in the interaction window, skip all the names in the list that start with _, what is left are valid string methods that you can explore by yourself.