Python Bootcamp 2

String objects - review

A **string** is an object kind of like text. It is contained in double `"` or single `'` quotes (you can choose which you want to use).

```
"Hello World"
```

```
'Hello World'
```

Let's use our first function! It can be used on almost all classes of objects:

```
print("Hello World")
```

```
Hello World
```

Strings can contain digits:

```
print("Hello World 2")
```

```
Hello World 2
```

Strings can contain **special characters**. The most commonly used are new lines `\n`:

```
print("Hello World\nHow are you?")
```

```
Hello World
How are you?
```

...and tabs `\t`:

```
print("Hello World\tHow are you?")
```

```
Hello World	How are you?
```

If you simply **run** the string, it will return the string how it looks to the computer:

```
"Hello World\nHow are you?"
```

```
'Hello World\nHow are you?'
```

...but if you use the `print()` function, it interprets the special characters and returns the string like this:

```
print("Hello World\nHow are you?\n123\n456\taaa")
```

```
Hello World
How are you?
123
456   aaa
```

What happens if you forget the quotes around a string?

```
print("Hello World")
```

```
Hello World
```

Our first error!! This is a common error - the `SyntaxError`. It often means that we forgot to type something.

```
pint("Hello World!")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 pint("Hello World!")

NameError: name 'pint' is not defined
```

A `NameError` is another common error - it often means we've mispelled a function or variable.

What error do you think we will get if we run this line of code?

```
print("Hello World)
```

```
  Cell In[10], line 1
    print("Hello World)
          ^
SyntaxError: unterminated string literal (detected at line 1)
```

Some arithmetic operators work with strings. To combine two strings:

```
print("Hello" + "World")
```

```
HelloWorld
```

```
print("Hello " + "World")
```

```
Hello World
```

```
print("Hello " + "World" + "!")
```

```
Hello World!
```

```
print("Hello World "*4)
```

```
Hello World Hello World Hello World Hello World
```

```
print("Hello World\n"*4)
```

```
Hello World
Hello World
Hello World
Hello World
```

String functions

```
greeting = "Hello World!"
```

```
print(greeting)
```

```
Hello World!
```

```
len(greeting)
```

```
12
```

```
greeting.replace("!", ".")
```

```
'Hello World.'
```

```
greeting.replace(" ", "")
```

```
'HelloWorld!'
```

`""` is referred to as an **empty string**.

```
print(greeting)
```

```
Hello World!
```

Even though we ran code to change our greeting, the greeting variable still **points** to the original greeting. `greeting` did not change when we used the method functions even though a changed version of `greeting` was returned.

To change the greeting, we would have to reassign the variable name:

```
greeting = greeting.replace("!", ".")
```

```
print(greeting)
```

```
Hello World.
```

```
big_greeting = greeting.upper()
```

```
print(big_greeting)
```

```
HELLO WORLD.
```

```
small_greeting = greeting.lower()
```

```
print(small_greeting)
```

```
hello world.
```

```
my_name = "Xyz"
```

```
greetMe = greeting.replace("World", my_name)
```

```
print(greetMe)
```

```
Hello Xyz.
```

```
greetMe2 = "Hello " + my_name + "."
print(greetMe2)
```

```
Hello Xyz.
```

```
aaa = greetMe2.replace('l','e')
print(aaa)
```

```
Heeeo Xyz.
```

You might have noticed that I included two lines of code in that code cell. Jupyter Noteboooks can run multiple lines of code in a single code cell.

Indexing strings

We can index strings to return a **substring**.

```
greeting = "Hello World!"
```

```
greeting[0:6]
```

```
'Hello '
```

```
hello = greeting[0:5]
print(hello)
```

```
Hello
```

```
hello[2]
```

```
'l'
```

Remember, in Python we start indexing at 0, so the third letter in our string is indexed as 2.

We can also start indexing from the right end of the string. The number in the farthest right position can be indexed as -1:

```
hello[-1]
'o'
hello[-2]
'l'
hello[-4]
'e'
hello[0:2]
'He'
hello[-1:5] # Pay attention to staet index and end index position.
'o'
hello[-1:4] # Pay attention to staet index and end index position.
''
hello[-1:9] # Pay attention to staet index and end index position.
'o'
```

Remember, our variable is `hello`, but the string it has stored is "Hello" with a capital "H".

If we don't give any number after the `:`, Python will continue to the end of the string:

```
hello[3:]
'lo'
```

If we don't give any number before the `:`, Python will start at the beginning of the string:

```
hello[:2]
'He'
```

We can also use negative numbers in indexing:

```
hello[2:-1]
'll'
hello[-3:-1]
```

```
'll'
```

```
"Hello"[-3:]
```

```
'llo'
```

BACK TO THE SLIDES

Writing functions

First we'll write a function that just does something whenever it's called. It takes no arguments.

```
def hello():
    # Prints Hello!
    print("Hello!") # pay attention to the indentation within a
function

print("hello".replace('l', "L"))
```

```
heLLo
```

Let's call the `hello()` function:

```
hello()
```

```
Hello!
```

We can add an argument. Whatever you call the arguments in your function definition must match exactly to how they are used inside the function definition, just like we saw with for loops and with/as statements:

```
namee = "abc"
def hello_you(): # 'Xyz' is the default value
    #Prints Hello Xyz! replacing Xyz with whatever string you give it.
    print("Hello " + namee + "!")
```

Function takes the default value if no argument is passed:

```
hello_you()
```

```
Hello abc!
```

Now we can pass it any string as an argument:

Next we'll write a function that creates a new object.

Let's write our own function to find the area of a rectangle.

The arguments our function will need are length and width.

```
def area(length, width):
    #This function takes a length and width of a rectangle and returns
the area.
    answer = length * width

area(10, 12)

print(answer) # see error below
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
Cell In[55], line 1
----> 1 print(answer) # see error below

NameError: name 'answer' is not defined
```

So we created `answer` inside our function definition, but it doesn't exist outside that definition. We need to include a **return** statement if we want our function to return the value of an object created inside the function.

```
def area(length, width):
    #This function takes a length and width of a rectangle and returns
the area.
    answer = length * width
    return answer
```

Still, `answer` exists only inside the `area` function:

```
print(answer) # see error below
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
Cell In[57], line 1
----> 1 print(answer) # see error below

NameError: name 'answer' is not defined
```

We must supply the argument(s) if the function does not specify default value(s):

```
area(5,2)
```

```
10
```

We can also assign the output of a function to a variable. Let's say my kitchen is 10 feet long and 12 feet wide:

```
temp_area = area(10, 12)

print(temp_area)
```

```
120
```

We can also pass variables to the function:

```
kitchen_l = 10
kitchen_w = 12

kitchen = area(kitchen_l, kitchen_w)
print(kitchen)
```

```
120
```

```
def area2(length = 5, width = 10):
    #This function takes a length and width of a rectangle and returns the area.
    answer = length * width
    return answer

area2()
```

```
50
```

```
area2(1, 20)
```

```
20
```

Example: Let's write a simple function to convert a volume in teaspoons to a volume in cups. There are 48 teaspoons in 1 cup.

```
def tspToCup(tsp):
    return tsp / 48

tspToCup(8)
```

```
0.16666666666666666
```

Let's improve it by rounding the answer:

```
def tspToCup(tsp):
    cup = round(tsp / 48, 2)
    return cup

tspToCup(8)
```

```
0.17
```

BACK TO THE SLIDES