Module 2; Notebook 2 - String concatenation & type casting

January 4, 2022

1 Module 2; Notebook 2 - String concatenation & type casting

In this notebook we'll now look at combining a few of the features we looked at in the previous notebook so that we can handle numbers correctly. We'll also explore concatenating strings together using the `+` operator.

1.1 String concatenation

Building on the foundations of the last notebook, let's look at how we can use are inputs and concatenate them with other strings. If you remember from the start of the notebook, we briefly brushed over the `+` operator which will allow us to do this.

[ ]:
```
name = 'Ethan'
print("My name is " + name)
```

The example above shows how we can conacatenate strings together to create a different output. Now let's see how we can do it with a string and an integer…

[ ]:
```
print("My age is " + 20)
```

Looking at the output, can you see what has happened here…

We can only concatentate strings together - but how do we add an integer into the sentence?

[ ]:
```
age = 20
age_str = str(age)
type(age_str)
```

[ ]:
```
print("My age is " + age_str)
```

To concatentate we've casted the integer 20 into a string and then concatenated it to the string "My age is"!

What is casting - let's take a further look into the process and use cases of type casting.

1.2 Type casting & the `input()` function in more detail

Type casting is the process of changing the data type of a value for example from a integer to a float. It is done using a casting function depending on the type you want to cast to:

[ ]:
```
example_1 = type(str(1)) # int -> string
example_2 = type(int("1")) #string -> int
example_3 = type(bool("True")) #string -> boolean
example_4 = type(float(1)) #int -> float

print(example_1, example_2, example_3, example_4)
```

As we can see from the printed classes and the code snippet, we have access to numerous casting functions to change the types of variables - if they're compatible.

By default, the `input()` function returns a string. For the cell below, enter in the value `6` for the sake of this example!

[ ]:
```
random_number = input()
```

[ ]:
```
type(random_number)
```

This means that when we pass in an integer as an input we'll have to cast it to the correct data type; into an integer. Casting is the process of change compatable values into antoher data type e.g. 3 as an integer can be casted to a float.

[ ]:
```
random_number2 = 42
type(random_number2)
```

[ ]:
```
casted_random_number = float(random_number2)
type(casted_random_number)
```

[ ]:
```
print(casted_random_number)
```

As you can see casting `42` into a float yields `42.0`. Do note that casting between some data types won't work for example casting a integer to a string:

[ ]:
```
test_string = "Hello!"
casted_test_string = int(test_string)
```

As you can see from the error when running the above cell, we can't cast a the string 'Hello' into an integer as it isn't a whole, numerical value.

That brings us to the end of the second module. For the last module we'll look at numerical operators and conidtional statements!