Set 3: Literals

Skill 3.1: Interpret integer literals

Skill 3.1 Concepts

A literal is data whose values are determined by the literal itself. As this is a difficult concept to understand, a good example may be helpful. Take a look at the following set of digits,

```
123
```

Can you guess what value it represents? Of course you can – it's one hundred twenty three.

But what about this,

```
c
```

Does it represent any value? Maybe. It can be the symbol of the speed of light, for example. It can also be the constant of integration. Or even the length of a hypotenuse in the sense of a Pythagorean theorem. There are many possibilities.

You cannot choose the right one without some additional knowledge. This means that *123* is a literal, and *c* is not.

Let's look at another example,

Code	Output
`print("2")` `print(2)`	2 2

The first line looks familiar. The second seems to be erroneous due to the visible lack of quotes. The example above illustrates two different types of literals: A String and an integer.

The print() function presents them in exactly the same way – this example is obvious, as their human-readable representation is also the same. Internally, in the computer's memory, these two values are stored in completely different ways – the string exists as just a string – a series of letters.

The number is converted into machine representation (a set of bits). The print() function can show them both in a form readable to humans.

An integer is one of two types of numbers that we will encounter, the other type is a floating-point, or just float. For now, we will focus on integers.

If you were asked to write the number eleven million one hundred eleven thousand one hundred eleven, you would write the number like this: 11,111,111, or like this: 11.111.111, or even like this: 11 111 111. While it's clear that this provision makes it easier to read, especially when the number consists of many digits, in Python it is prohibited. What Python does allow, though, is the use of underscores in numeric literals,

Code	Output
`print(11111111)` `print(11_111_111)`	11111111 11111111

Negative numbers in Python are defined as usual, by adding a minus. You can write: -11111111, or -11_111_111, for example.

Code	Output
`print(-11111111)` `print(-11_111_111)`	-11111111 -11111111

Skill 3.1 Exercise 1

Skill 3.2: Interpret float literals

Skill 3.2 Concepts

A float literal is any number that has (or may have) a fractional part after the decimal point. Whenever we use a term like *two and a half or minus zero point four*, we think of numbers that the computer considers floating-point numbers,

```
2.5 - 0.4
```

In the example, 0.4 could have been written as .4, and 4.0 could be written as 4.,

```
.4
4.
```

This will change neither its type nor its value. In both cases above, the decimal is used to indicate a floating-point number.

Skill 3.2 Exercise 1

Skill 3.3: Write numbers in scientific notation

Skill 3.3 Concepts

A decimal is not the only way to indicate a float. We can also use the letter *e*.

When you want to use any numbers that are very large or very small, you can use scientific notation. Take, for example, the speed of light, expressed in meters per second. Written directly it would look like this: 300000000.

To avoid writing out so many zeros, physics textbooks use an abbreviated form, which you have probably already seen: 3 x 10^8. It reads: three times ten to the power of eight.

In Python, the same effect is achieved in a slightly different way – take a look:

```
3.8E8
```

The letter E (you can also use the lower-case letter e – it comes from the word exponent) is a concise record of the phrase times ten to the power of.

When using scientific notation,

- the exponent (the value after the E) has to be an integer;
- the base (the value in front of the E) may be either an integer or a float.

We can also use scientific notation to type very small numbers, for example, Plank's constant, 6.62607 x 10^{-34}.

```
6.62607E-34
```

One final note is that just because you chose to type a number a certain way in Python, what you type may not be what is printed. Python instead will print the most economical display. Consider the following examples,

Code	Output
`print(0.00000000000000000001)`	1e-22

Skill 3.3 Exercise 1

Skill 3.4: Interpret String literals

Skill 3.4 Concepts

Strings are used when you need to process text (like names of all kinds, addresses, novels, etc.), not numbers. You already know a bit about them, e.g., that strings need quotes the way floats need points,

```
"I am a String"
```

What if we want to encode a quote inside a string that is already delimited by quotes? The following, for example, would produce an error,

Code	Output
`print("I like "Monty Python"")`	Syntax error

There are two solutions to printing the above without an error. The first solution is based on a concept we already know about – the escape character (\). The backslash can escape quotes too. A quote preceded by a backslash changes its meaning – it's not a delimiter, but just a quote.

Code	Output
`print("I like \"Monty Python\"")`	I like "Monty Python"

The second method is to use apostrophes around the string we want to print,

Code	Output
`print('I like "Monty Python"')`	I like "Monty Python"

We can also embed an apostrophe in a string that is defined in apostrophes using the escape character,

Code	Output
`print('I\'m Monty Python.')`	I'm Monty Python.

Skill 3.4 Exercise

Skill 3.5: Interpret boolean literals

Skill 3.5 Concepts

The name Boolean comes from George Boole (1815-1864), the author of the fundamental work, The Laws of Thought, which contains the definition of Boolean algebra – a part of algebra that makes use of only two distinct values: True and False, denoted as 1 and 0.

In Python True and False are denoted as follows,

```
True
False
```

The example below illustrates that True and False are evaluated as 1 and 0, respectively in Python.

Code	Output
`print(True > False)` `print(True < False)`	True False

Skill 3.6: Interpret the none literal

Skill 3.6 Concepts

There are infinitely* many integers, floating point numbers, and strings. However, for the NoneType data type, there is only one value: None.

- Cells will **not** output expressions that evaluate to **None**.
- **None can** be displayed (i.e., printed).
- **None** is also referred to as the "null value."

As it turns out**, print()** returns **None**, therefore when evaluated as the last line in a cell:

- Print **displays** the value of the evaluated argument
- But the cell **does not output** anything!

To better understand this concepts lets explore how to print the datatype of a literal below.

Skill 3.7: Apply the type function to determine the datatype of a literal

Skill 3.7 Concepts

In this lesson we explored the different types of literals associated with Python. The type function is a useful for determining the datatype of a literal. Below is an example of how we can use the type function to print out the datatype of the literals we've learned about so far.

Code	Output
`print(type(2.))`	<class 'float'>
`print(type(True))`	<class 'bool'>
`print(type("Python"))`	<class 'str'>
`print(type(2 - 5))`	<class 'int'>
`print(type(1E5))`	<class 'float'>
`print(type(None))`	<class 'NoneType'>

Skill 3.7 Exercise 1