Built-In Functions

In this lesson, we'll take a look at some of the built-in functions offered by Python.

We'll cover the following

Python boasts a huge library of built-in functions. As we explore the language further, we'll discover many more of these functions.

And trust us when we say that there's something for almost everyone.

Let's look at some of the simple built-in functions.

Strings

Functions that are properties of a particular entity are known as **methods**. These methods can be accessed using the `.` operator. The string data type has several methods associated with it. Let's look at some of them.

Search

An alternative for finding a substring using the `in` keyword is the `find()` method. It returns the first index at which a substring occurs in a string. If no instance of the substring is found, the method returns `-1`.

`-1` is a conventional value that represents a `None` or failure in case the output was supposed to be positive.

For a string called `a_string`, `find()` can be used in the following way:

```
a_string.find(substring, start, end)
```

- `substring` is what we are searching for.
- `start` is the index from which we start searching in `a_string`.
- `end` is the index where we stop our search in `a_string`.

`start` and `end` are optional.

```
random_string = "This is a string"
print(random_string.find("is"))  # First instance of 'is' occurs at index 2
print(random_string.find("is", 9, 13))  # No instance of 'is' in this range
```

Replace

The `replace()` method can be used to replace a part of a string with another string. Here's the template we must use:

```
a_string.replace(substring_to_be_replace, new_string)
```

The original string is not altered. Instead, a new string with the replaced substring is returned.

```
a_string = "Welcome to Educative!"
new_string = a_string.replace("Welcome to", "Greetings from")
print(a_string)
print(new_string)
```

Changing the Letter Case

In Python, the letter case of a string can be easily changed using the `upper()` and `lower()` methods.

Let's try them out:

```
print("UpperCase".upper())
print("LowerCase".lower())
```

Type Conversions

There may be times when we need to change data from one type to another. In Python, this is usually an easy process since the compiler can automatically convert data between different types in order to avoid errors.

However, there are built-in functions which allow us to perform explicit type conversions.

Let’s explore them in this lesson.

`int()`

To convert data into an integer, we can use the `int()` utility.

Keep in mind, a string can only be converted to an integer if it is made up of numbers.

```
print(int("12") * 10)  # String to integer
print(int(20.5))  # Float to integer
print(int(False))  # Bool to integer

# print (int("Hello")) # This wouldn't work!
```

`ord()`

This function can be used to convert a character to its Unicode value:

```
print(ord('a'))
print(ord('0'))
```

`float()`

The `float()` function translates data into a floating-point number:

```
print(float(24))
print(float('24.5'))

print(float(True))
```

`str()`

To convert data into a string, we must use `str()`:

```
print(str(12) + '.345')
print(str(False))
print(str(12.345) + ' is a string')
```

`bool()`

This function takes in data and gives us the corresponding Boolean value.

Strings are always converted to `True`. Floats and integers with a value of zero are considered to be `False` in Boolean terms:

```
print(bool(10))
print(bool(0.0))
print(bool("Hello"))
```

There are several other conversions as well including `complex()` and `hex()`, but we'll leave them for you to explore on your own.

In the world of Python's built-in functions, what we've seen so far is just the tip of the iceberg.

In the next lesson, we'll learn about a special type function called the **lambda**.