2.6 Type Conversion Functions

There is another useful set of built-in functions that we have not yet discussed: functions that allow us to convert values between different data types. For example, given a string `'10'`, can we convert it into the integer `10`? Or given a list `[1, 2, 3]`, can we convert it into a set `{1, 2, 3}`?

The answer to these questions is yes, and the way to do so in Python is quite elegant. Each data type that we have learned about so far, from `int` to `dict`, is also a function that takes an argument and attempts to convert it to a value of that data type.

Here are some examples. Some of these are more "obvious" than others. Don't worry about the exact rules for conversions between types, as you won't be expected to memorize them. Instead, we just want you to know that these conversions are possible using data types as functions.

```
>>> int('10')
10
>>> float('10')
10.0
>>> bool(1000)
True
>>> bool(0)
False
>>> list({1, 2, 3})
[1, 2, 3]
>>> set([1, 2, 3])
{1, 2, 3}
>>> set()  # Giving set no arguments results in the empty set
set()
>>> dict([('a', 1), ('b', 2), ('c', 3)])
{'a': 1, 'b': 2, 'c': 3}
```

In particular, `str` is the most versatile of these data types. *Every* value of the data types we've studied so far has a string represention which corresponds directly to how you would write the value as a Python literal.

```
>>> str(10)
'10'
>>> str(-5.5)
'-5.5'
>>> str(True)
'True'
>>> str({1, 2, 3})
'{1, 2, 3}'
>>> str([1, 2, 3])
'[1, 2, 3]'
>>> str({'a': 1, 'b': 2})
"{'a': 1, 'b': 2}"
```

Warning: conversion errors

Type conversion functions in Python are quite powerful, but also have their limits. You often have to be careful when attempting to convert between different data types, as not all values of one type can be converted into another. Attempting to convert an "invalid" value often results in a Python exception to be raised:[1]

[1] These exceptions typically have type `ValueError` or `TypeError`.

```
>>> int('David')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'David'
>>> list(1000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

Preview: creating values of arbitrary data types

The ability to create values of a given type by calling the data type as a function is not unique to the built-in data types in this section. We've actually seen two examples of doing this so far in the course!

`range` *revisited*

Earlier, we saw that we could call `range` to create a sequence of numbers. But if you just try calling `range` by itself in the Python console, you see something kind of funny:

```
>>> range(5, 10)
range(5, 10)
```

Whereas you might have expected to see a list (`[5, 6, 7, 8, 9]`), in the Python console output it looks like nothing happened at all! This is because `range` is actually a type conversion function: Python also has a `range` data type that is distinct from lists (or other collection data types).

```
>>> five_to_nine = range(5, 10)
>>> type(five_to_nine)
<class 'range'>
>>> five_to_nine == [5, 6, 7, 8, 9]
False
```

However, we can convert a `range` value into a `list` with, you guessed it, a type conversion:

```
>>> list(five_to_nine)
[5, 6, 7, 8, 9]
```

`datetime.date` *revisited*

Recall an example from the last section in Section 2.5 Importing Modules:

```
>>> import datetime
>>> canada_day = datetime.date(1867, 7, 1)  # Create a new
>>> type(canada_day)
<class 'datetime.date'>
```

In this case, the data type is `datetime.date`, and it is called on three arguments instead of one. In this context, `datetime.date` is called to *create* a new `date` value given three arguments (the year, month, and day).[2] And of course, this behaviour isn't unique to `datetime.date` either. As we'll see a bit later in this course, you'll be able to take *any* data type—even ones you define yourself—and create values of that type by calling the data type as a function.

[2] This is a more general form of type "conversion", which created a data type of a new value given a single argument.