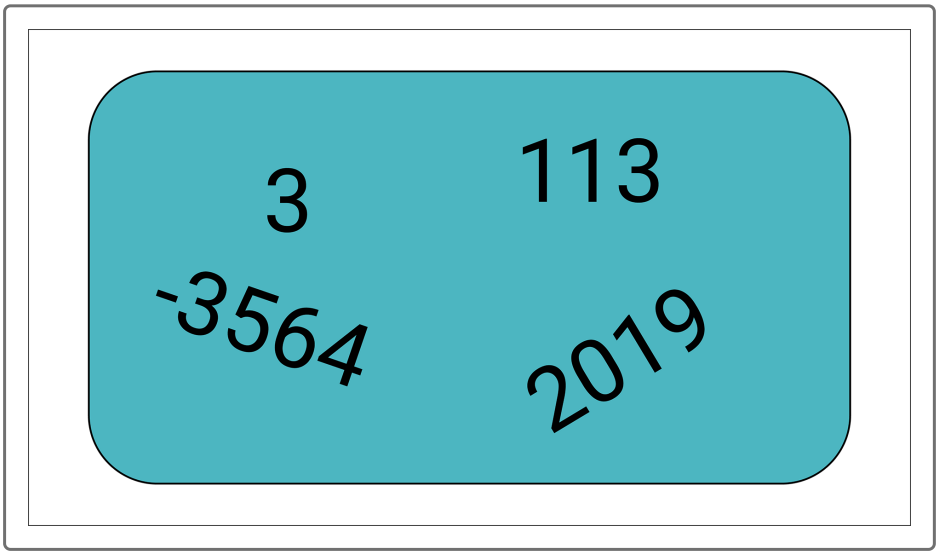
3.2.3 Data Types

Now that you are familiar with opening the Python interpreter, Tom is going to walk you through some of the basics of the Python language. This way, you'll be up-to-speed on the skills needed to perform the election audit. The first thing Tom will do is go over the data types you're likely to encounter. Knowing how to identify the data types you are working with is key, because only certain data types can be used for calculations. In the election analysis, you will be performing mathematical calculations as well as getting non-mathematical data from a file.

There are a variety of data types in Python, some of which you may encounter in the election dataset. Because you need to be prepared to handle different types of data, let's discuss a few key data types: integers, floating-point decimal numbers, strings, and Boolean values.

Integers



Integers are used to perform mathematical calculations. The length of an integer is constrained by the amount of RAM memory on your computer. For example, a computer with 8 GB of RAM will have 8,000,000,000 bytes. The integer 10,000,000,000,000,000,000, or 10 quintillion, will only take up 36 bytes of memory.

REWIND

An integer is a positive or negative whole number. Whole numbers between −32,768 and 32,767 are stored in 16 bits, and whole numbers between −2,147,483,648 and 2,147,483,647 are stored in 32 bits.

Let's determine the data type of a given integer using the command line. To get started, open the command line and activate the Python interpreter.

NOTE

> To use Python in the command line:
>
> - If you're on a Mac, type `python3`.
> - If you're using Windows, open the Python 3.7 (64-bit) Command Prompt.

The `type()` function is used to determine data type. Inside this function, add the integer or other value to determine the data type. For example, let's say we want to find the data type of the integer 3. Type `type(3)` after the Python interpreter prompt, as shown, and then press Enter.

```
>>> type(3)
```

The output shows that the data type for the number 3 is an integer, as denoted by the `'int'`.

```
>>> type(3)
<class 'int'>
```

Note that when typing integer values greater than 999, a thousands separator, or comma, should **not** be used. Inserting commas can make the whole number you thought you were typing something different. For example, the variable "ballots" is equal to 1,341.

```
>>> ballots = 1,341
>>> ballots
(1, 341)
>>>
```

When we type "ballots" in the next line, the output is not 1,341, but 1 and 341. This is known as the Python data type called a **tuple** (which we'll encounter later), containing two numbers.

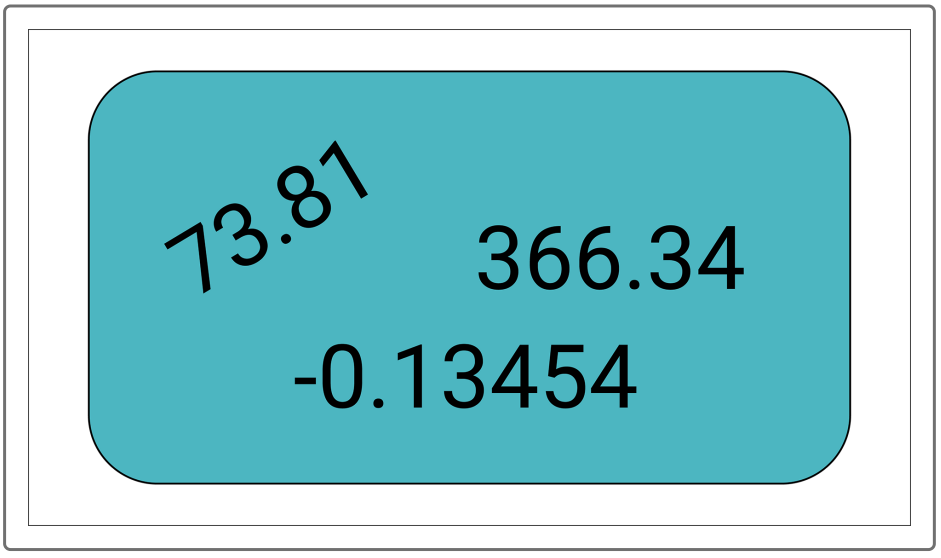
We can confirm that "ballots" is a tuple by determining its data type.

```
>>> type(ballots)
<class 'tuple'>
>>>
```

IMPORTANT

> When typing integers greater than 999 in Python, do not use a thousands separator (comma).

Floating-Point Decimal Numbers



Like integers, **floating-point decimal numbers** are used to perform mathematical calculations. Floating-point decimal numbers specify numbers that have a decimal point, like 73.81.

Retake

When we type `type(73.81)` and press Enter, the output shows that this is a floating-point decimal number, as indicated by "float."

```
>>> type(73.81)
<class 'float'>
```

Strings

String variables can be text or numbers wrapped by either single or double quotes, or delimiters. A **delimiter** is any nonalphabetical characters used to specify the boundary between plain text or other numbers, like the single or double quotes, or opening and closing parentheses. All characters between the quotes are part of the string; for example, `'Hello World'` or `"Hello World"`.

To determine the data type of `"Hello World"`, type the following:

`type("Hello World")`

Press Enter. The output shows that this is a string, `'str'`.

```
>>> type("Hello World")
<class 'str'>
```

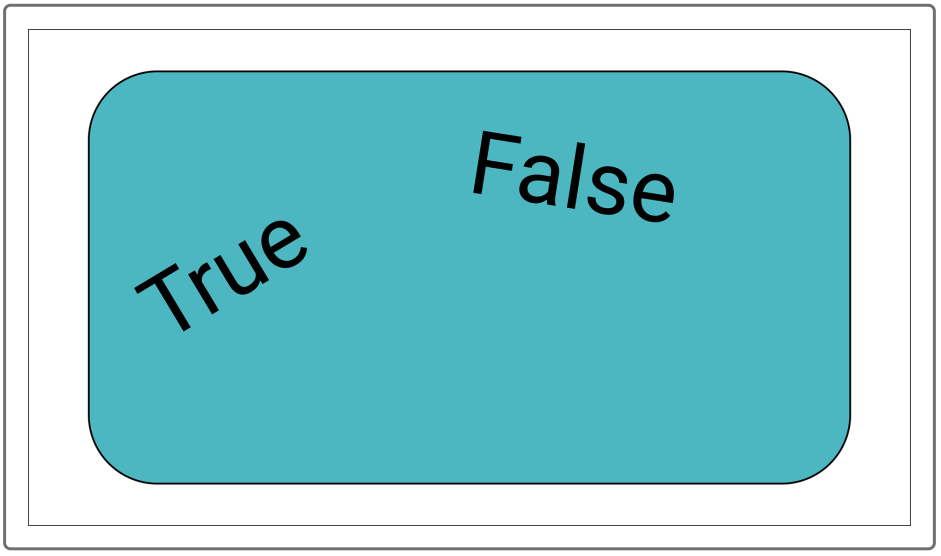
Sometimes, you may come across **empty strings**, which contain no text or numbers between the delimiters. A string that is empty is written as `''`, or `""`. Because there are opening and closing single or double quotes, the data type is still a string.

SKILL DRILL

Determine the data type of an empty string.

Unlike integers and floating-point decimal numbers, strings cannot be used to perform mathematical calculations. Also, it's important to note that if an integer or a floating-point decimal number is wrapped in quotes, it's considered to be a string, not an integer or decimal number. Later in this module, we'll go over how to change a data type from a string to an integer or floating-point decimal number.

Boolean Values



Boolean data types can have one of two values: true or false. If we pass `True` in the `type()` function, we'll get the data type `'bool'`.

IMPORTANT

> To determine the data type of Boolean values, "True" or "False" must be capitalized when written in code.

```
>>> type(True)
<class 'bool'>
```

Retake

To summarize, here are four data types that we encountered in this module.

Data Type	Python Classification
Integers	`<class 'int'>`
Float point numbers	`<class 'float'>`
Strings	`<class 'str'>`
Boolean	`<class 'bool'>`

NOTE

> For more information, read the **documentation on Python data types (https://docs.python.org/3.7/library/stdtypes.html#numeric-types-int-float-complex)** .

Creating Variables in Python

Variables are essentially a way to store different data types to be used later. A **variable** is a name that represents a value that is stored in the computer's memory. A variable can be an integer, floating-point decimal number, string, or Boolean value. In Python, you can declare a variable by typing a variable name followed by an equals sign.

REWIND

After a variable is declared, it is assigned a value. This is done by referencing the variable by name and setting its value with an equals sign.

Here are a few examples of declared variables in Python.

```
num_candidates = 3
winning_percentage = 73.81
candidate = "Diane"
won_election = True
```

Naming Conventions for Python Variables

Variable names in Python can be any length and can consist of the following:

- Uppercase letters (A–Z)
- Lowercase letters (a–z)
- Digits (0–9)
- Underscores (_).

When considering a variable name, keep the following in mind:

- The first character of a variable name cannot be a digit.
- Choose a variable name that reflects what variable will reference and can be understood by someone reading your code. For example, using `x = "Diane"` for a candidate's name is not a good choice; instead, use `candidate_name = "Diane"`.
- Variables are case-sensitive.
- Shorter names are better.
- Variables must be in **snake case,** which is when words are separated by underscores. For example: `candidate_name`.

One final consideration: the Python language has a small set of **keywords**, or reserved words that have special language functionality. No variable can have the same name as a keyword. These reserved words in Python are listed below.

False	**def**	**if**	**raise**
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with

assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

NOTE

> You can also get this list if you type `help("keywords")` in the command line.

When you follow these rules, you can create variables with any name that you like. Remember, it's recommended that you choose short, simple names that accurately reflect what the variable represents.