String indexing and slicing

As you know, strings are an important class of data because they represent text. Data professionals encounter strings all the time, so it's important to become familiar with different ways of manipulating and working with them. This reading will review the string formatting techniques you've learned and also introduce you to regular expressions.

String formatting

Indexing refers to accessing a single element of a sequence by its position. *In Python, the first element of any sequence has an* **index of zero.** This means Python uses zero-based indexing. Numerous other programming languages also use zero-based indexing, but not all of them do. Some languages use one-based indexing, such as R, Julia, and SAS.

Use square brackets to perform indexing. Here are some examples:

```
my_string = 'Mississippi half-step'
print(my_string[0])

my_list = [1, 'unladen', 'swallow']
print(my_list[1])

print(my_list[-1])
```

RunReset

```
M
unladen
swallow
```

In these examples, there are two sequence variables: a string and a list. Indexing is used to access the character at index zero of the string, which is its first character—M. The list is selected at index one, which contains the word "unladen." The list is also selected at its final position using negative indexing.

Note: If you try to select an index that is out of range of what the object contains, you'll get an IndexError.

```
my_list = [1, 'unladen', 'swallow']
my_list[3]
```

RunReset

```
Error on line 2:
```

```
    my_list[3]
IndexError: list index out of range
```

Slicing

Slicing refers to accessing a range of elements from a sequence. Use square brackets containing two indices separated by a colon.

Here are some examples:

```
new_string = 'pining for the fjords'
print(new_string[0:3])
print(new_string[:3])
```

RunReset

```
pin
pin
```

These two examples, each with slightly different syntax, are being used to produce the same result. Notice two things: (1) the resulting slice includes the starting index and excludes the ending index; (2) when the starting index is omitted it's implied to be zero, as shown in the second print line.

The process follows the same logic when the ending index is omitted:

```
new_string = 'pining for the fjords'
print(new_string[6:21])
print(new_string[6:])
print(len(new_string))
```

RunReset

```
 for the fjords
 for the fjords
21
```

Again, there are two statements that are syntactically different but still produce the same substring. When the ending index is omitted, its implied value is the length of the sequence.

Finally, the code will throw an IndexError if you try to index a sequence at an index number outside the scope of the elements; this is not the case for slicing.

For example:

```
new_string = 'pining for the fjords'
print(new_string[6:100])
```

RunReset

```
 for the fjords
```

Although the ending index was 100—far beyond the scope of the indices in the string—the computer returned a substring that ended with the string's final element.

Key takeaways

Indexing and slicing are powerful tools in Python that allow you to access specific elements or parts of a sequence. Both indexing and slicing use square brackets. Remember that in a slice the starting index is inclusive and the stopping index is exclusive, and that negative indices count from the end of the sequence. With these tools, you can manipulate strings and other iterable sequences to perform a wide variety of operations, making you a more proficient data professional.