1 Strings

Python represents text as “strings”, which are sequences of characters.

1.1 Basics

We define strings using either single or double quotes:

'csc120' or "csc120"

unless a string needs to span more than one line and then we use triple quotes:

```python
s = '''This is a very, very long sentence that needs to span more than one line.'''
```

Q. What do you think the result of 'csc' + '120' is?

A.

Q. What other use does “+” have?

A.

We say that “+” is overloaded.

Q. How do you think Python knows what to do?

A.

Q. Why would "csc" + 120 result in an error?

If we want to use “+” between a variable of type string and of type int we need to convert one:

```python
>>> 'csc' + str(120)
'csc120'
```

Q. What do you think "hi" * 3 will do?

Q. How about len("Mississippi")?

Q. How about "ou" in "colour"?
2 Indexing and Slicing Strings

Since strings are sequences of characters, we sometimes want to be able to retrieve portions of the string. To do so, we need to have a way to refer the \(i^{th}\) character. By convention, the first character has position or \textit{index} 0. After that, each character has an index one greater than the index of the character before it. The index -1 refers to the last character of the string; -2 is the second-to-last; etc. Another way to specify the index of the last character of a string \(s\) is \(\text{len}(s) - 1\).

\textit{Examples}, with \(s = "sliceofspam"\)

<table>
<thead>
<tr>
<th>Index/Slice</th>
<th>Expression’s Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s[0])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[3])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[-2])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[2:5])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[3:])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[:8])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[:])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[3:-2])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[-5:])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s[5:2])</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.1 Comparing Strings

Strings are \textit{comparable} in that lowercase letters increase in alphabetic order, as do uppercase letters. This makes it easy to compare words. For example:

\[ 'a' < 'b' \text{ ; } 'ant' <= 'apple' \text{ ; } 'First' < 'Second' \]

Q. What about \(a' == 'A' \text{ or } ', ' < '!\) or '! < 'a'\)?

A.

3 Methods

We have been using some string operators such as:

\[ 'csc' + '120' \# concatenation \]
\[ "hi" * 3 \# repetition \]
\[ "ou" in "colour" \# substring \]

There are many more! — so many, in fact, that there aren’t enough operator symbols for all of them. Python defines these operators using a special kind of function called a \textit{method}.

The words “methods” and “functions” are sometimes used interchangeably, as they are the same in some programming languages. But in Python they are different.

- You call a function on a string like so: \texttt{function\_name(x)}.
- But you call a method on a string like so: \texttt{x.method\_name()}.
Using methods:

- These particular methods \textit{belong} to the string type.
- We call methods using the same notation as calling a function in a module:

  ```python
  'u of toronto'.capitalize()
campus = 'st george'
campus.capitalize()
  ```

- Because you connect the method call to the string using dot notation, you don’t need to pass it in as a parameter: `campus.capitalize(campus)` would be redundant (and wrong).

- Some methods \textit{require} parameters — for example,
  ```python
campus.startswith('ut')
  ```

Why?

- You need to know when you are calling a \textit{method} vs calling a \textit{function}, since the notation is different. For example:

  ```python
  s = "Hi, CSC120 students!"
  len(s)  # len is a built-in function. s is the argument passed in.
  s.len()  # Doesn't work. len is not a string method.
  s.lower()  # lower is a str method: it operates on s
  lower(s)  # Doesn't work. lower() must be called on an object
  ```

### 3.1 String Methods

Q. How can we get a list of all string methods with their descriptions?

A.

For the methods listed below, assume that the string we are using is: \texttt{s = "This is CSC120"}.

<table>
<thead>
<tr>
<th>Method</th>
<th>Expression’s Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.lower()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.replace(&quot;20&quot;, &quot;48&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.count(&quot;i&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;yabababa daba doo!&quot;.count(&quot;abab&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.find(&quot;is&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.rfind(&quot;is&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.find(&quot;is&quot;, 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.find(&quot;is&quot;, 2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.find(&quot;the&quot;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Define a new \texttt{s = " A quite open line "}.

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.lstrip()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.rstrip()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s.strip()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.2 Strings are Immutable

Strings are **immutable**: their state cannot be changed once they are defined.

**Q.** After the following two lines, is \( s \) now ‘Ho, Jim’?

\[
s = 'Hi, Jim' \\
s[1] = 'o'
\]

3.3 Escape Sequences

**Q.** How can we put a newline or quote inside a string?

**A.** We use special escape sequences.

For each of the following escape sequences, guess what they do:

```
\n \'
" \t \\n```

**Example:** What does this print?

```
print("Anna's and Jen's half marathon times:
  1:54:26.1 \t 1:50:09.8")
```

4 Loops

Sometimes we need to walk or iterate through each character in a string. This is called **looping** or **iterating** over the string.

```
The form for looping over a string:

```for char in s:
    <do something involving char>

```for, in are Python keywords.
`char` is a variable that refers to the value of the current character.
`s` is a string or a variable of type string

**Example**

```python
def print_vertical(s):
    for char in s:
        print(char)
```

**Q.** How many times does the loop execute?