7. String Methods

Topics:
- Methods and Data
- More on Strings
- Functions and Methods
- The String Class

Data + Functions Together

"The square root of nine is three."
The tone of this comment is that the square root function can be applied to numbers like nine.

"Three is nine's square root."
The tone of this comment is that the number nine (like all numbers) comes equipped with a `sqrt` function.

Methods

A special kind of function that is very important to object-oriented programming is called a method.

In this style of programming, there is a tight coupling between structured data and the methods that work with that data.

Three String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>How many times does string t occur in a string s?</td>
</tr>
<tr>
<td>find</td>
<td>Where is the first occurrence of string t in a string s?</td>
</tr>
<tr>
<td>replace</td>
<td>In a string s replace all occurrences of a string s1 with a string s2.</td>
</tr>
</tbody>
</table>

There will be others later in the course.

Designing count as a Function

```
count 'ITH-JFK-ITH'  # count
'ITH'  # => 2
```

It would then be used like this: `n = count(y, x)`
Designing \textit{count} as a Method

Suppose
\begin{align*}
x &= 'ITH-JFK-ITH' \\
y &= 'ITH'
\end{align*}

Instead of the usual function-call syntax
\begin{align*}
n &= \text{count}(y, x)
\end{align*}
we will write
\begin{align*}
n &= x.\text{count}(y)
\end{align*}

Methods: The Notation

Here is the syntax associated with using a string method:

\begin{align*}
\text{name of string} \cdot \text{name of method} (\text{arg1, arg2, ...})
\end{align*}

Once again, the 'dot' notation

String Methods: \textit{count}

\begin{verbatim}
>>> s = 'ITH-JFK-ITH'
>>> m = s.count('ITH')
\end{verbatim}

\begin{verbatim}
s --> IT H - J F K - I T H
m --> 2
\end{verbatim}

\begin{verbatim}
s1.count(s2) the number of occurrences of string s2 in string s1
\end{verbatim}

String Methods: \textit{count}

\begin{verbatim}
>>> s = 'ITH-JFK-ITH'
>>> m = s.count('LGA')
\end{verbatim}

\begin{verbatim}
s --> IT H - J F K - I T H
m --> 0
\end{verbatim}

\begin{verbatim}
s2.count(s1) the number of occurrences of string s2 in string s1
\end{verbatim}

\textbf{count}

\textbf{The Formal Definition}

If \textit{s1} and \textit{s2} are strings, then
\begin{align*}
\text{s1.count(s2)}
\end{align*}
returns an \textit{int} value that is the number of occurrences of string \textit{s2} in string \textit{s1}.

\begin{verbatim}
print n
\end{verbatim}

\begin{verbatim}
Illegal: n = $A$.count('a' or 'e' or 'i' or 'o' or 'u')
\end{verbatim}

Using \textit{count}: An Example

\begin{verbatim}
# Count the number of vowels...
A = 'auric goldfinger'
n = 0
n = n + A.count('a')
n = n + A.count('e')
n = n + A.count('i')
n = n + A.count('o')
n = n + A.count('u')
print n
\end{verbatim}

Note, in general $A$.count(s2) is not the same as $s2$.count(s1)
Designing `find` as a Function

**find** 
Where is the first occurrence of string `y` in a string `x`?

`'ITH-JFK-ITH'` <--- `find` <--- `3` 

It would then be used like this: `n = find(y, x)`

Designing `find` as a Method

```python
>>> s = 'ITH-JFK-ITH'
>>> idx = s.find('JFK')
```

```
s --> I T H - J F K - I T H
     0 1 2 3 4 5 6 7 8 9 10
idx --> 4
```

`s1.index(s2)` the index of the first occurrence of string `s2` in string `s1`

String Methods: `find`

```python
>>> s = 'ITH-JFK-ITH'
>>> idx = s.find('RFK')
```

```
s --> I T H - J F K - I T H
     0 1 2 3 4 5 6 7 8 9 10
idx --> -1
```

`s1.index(s2)` evaluates to `-1` if there is no occurrence of `s2` in `s1`

Find: The Formal Definition

If `s1` and `s2` are strings, then

```
s1.find(s2)
```

returns an int value that is the index of the first occurrence of string `s2` in string `s1`.

If there is no such occurrence, then the value `-1` is returned.

Using `find`: Some Examples

```
s = 'nine one one'
n1 = s.find('one')
n2 = s.find('two')
n3 = s.find('nine')
```

```
n1 --> 5   n2 --> -1   n3 --> -1
```

In: A Handy Boolean Device

If `s1` and `s2` are strings, then

```
s1 in s2
```

is a boolean-valued expression.

```python
True  if there is an instance of `s1` in `s2`.
False if there is NOT an instance of `s1` in `s2`.
```
**in versus find**

These are equivalent:

\[ x = s1 \text{ in } s2 \]
\[ x = s2.\text{find}(s1) \geq 0 \]

---

**Designing replace as a Function**

replace  In a string s replace all occurrences of a string s1 with a string s2.

\[ \text{replace} \quad \text{replace} \quad \text{replace} \]
\[ \text{'ITH-JFK-ITH'} \quad \text{'ITH'} \quad \text{replace} \quad \text{'??-JFK-??'} \]

If would then be used like this: \( \text{new} = \text{replace}(s, s1, s2) \)

---

**Designing replace as a Method**

\[
s = \text{'one hundred and one'}
\]
\[
t = s.\text{replace}(' ', '-')
\]

\[ s \rightarrow \text{'one hundred and one'} \]
\[ t \rightarrow \text{'one-hundred-and-one'} \]

---

**The replace Method**

\[
s = \text{'one hundred and one'}
\]
\[
t = s.\text{replace}(' ', '')
\]

\[ s \rightarrow \text{'one hundred and one'} \]
\[ t \rightarrow \text{'onehundredandone'} \]

Replacing each blank with the "null string".

---

**The replace Method**

\[
s = \text{'one hundred and one'}
\]
\[
t = s.\text{replace}('x', '-')
\]

\[ s \rightarrow \text{'one hundred and one'} \]
\[ t \rightarrow \text{'one hundred and one'} \]

Replacing one character with another.

---

**The replace Method**

\[
s = \text{'one hundred and one'}
\]
\[
t = s.\text{replace}('one', 'seven')
\]

\[ s \rightarrow \text{'one hundred and one'} \]
\[ t \rightarrow \text{'seven hundred and seven'} \]

Replacing one substring with another.

---
The replace Method

\[
\begin{align*}
    s &= \text{'one hundred and one'} \\
    t &= s.\text{replace('two','seven')} \\
    s &\rightarrow \text{'one hundred and one'} \\
    t &\rightarrow \text{'one hundred and one'}
\end{align*}
\]

No change if the designated substring is missing.

The Formal Definition

If \( s, s1 \) and \( s2 \) are strings, then

\[
s.\text{replace}(s1,s2)
\]

returns a copy of the string \( s \) in which every non-overlapping occurrence of the string \( s1 \) is replaced by the string \( s2 \).

If \( s1 \) is not a substring of \( s \), then the returned string is just a copy of \( s \).

Using replace: Some Examples

\[
\begin{align*}
    s &= \text{'xxx'} \\
    t1 &= s.\text{replace('x','o')} \\
    t2 &= s.\text{replace('xx','o')} \\
    t3 &= s.\text{replace('xx','oo')}
\end{align*}
\]

\[
\begin{align*}
    t1 &\rightarrow \text{'ooo'} \\
    t2 &\rightarrow \text{'ox'} \\
    t3 &\rightarrow \text{'oox'}
\end{align*}
\]

replace does Not Replace

\( s.\text{replace}(s1,s2) \) does not change the value of \( s \).

It produces a copy of \( s \) with the specified replacements.

You are allowed to overwrite the "original" \( s \) with its "updated" copy:

\[
s = s.\text{replace}(s1,s2)
\]

Illegal!

\[
\begin{align*}
    s &= \text{'abcdefg'} \\
    s[5] &= \text{'x'}
\end{align*}
\]

Strings are immutable. They cannot be changed.

Have to "live with" the replace function, slicing, and concatenation

\[
\begin{align*}
    s &= \text{'abcdefg'} \\
    s &= s[:5]+\text{'x'}+s[6:]
\end{align*}
\]

Quickly Review Some Other String Methods
The upper and lower Methods

```python
s = 'A2sh?'
t1 = s.upper()
t2 = s.lower()
s -> 'A2sh?'
t1 -> 'A2SH?'
t2 -> 'a2sh?'
```

Some Boolean-Valued Methods

These methods return either True or False:

- `islower()`
- `isupper()`
- `isalnum()`
- `isalpha()`
- `isdigit()`

Boolean-Valued Methods

<table>
<thead>
<tr>
<th></th>
<th>s='ab3?'</th>
<th>s='AbcD'</th>
<th>s='AB3'</th>
</tr>
</thead>
<tbody>
<tr>
<td>islower</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>isupper</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

Boolean-Valued Methods

<table>
<thead>
<tr>
<th></th>
<th>'23'</th>
<th>'5a7'</th>
<th>'ab'</th>
<th>'-2.3'</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>isalpha</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>isdigit</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Useful String Constants

```
alpha = string.ascii_lowercase
```

Useful String Constants

```
specialChar = string.punctuation
```

!"#$%&'()*+,./:<=>?@[\]^_`{|}~
Useful String Constants

```
TheDigits = string.digits
```

```
1234567890
```

The "Dot" Notation--Again

```
math.sqrt
math.pi
```

```
pi=3.1416
```

```
math.py
```

The "folder" metaphor.
The "dot" means "go inside and get this".

```
string is a "Special" Module
```

```
"string.py"
```

```
digits = '01234567890'
letters = 'abcdef etc
punctuation = '!"#$ etc
```

The "folder" metaphor.
The "dot" means "go inside and get this".

```
string is actually a "class". More in a few lectures.
```