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.

A new point of view
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.
Methods

Hard to appreciate the reasons for this coupling between data and methods so early in the course.

For now, we settle on getting used to the special notation that is associated with the use of methods.

We will get into this topic using strings.
Three String Methods

**count**  How many times does string $t$ occur in a string $s$?

**find**  Where is the first occurrence of string $t$ in a string $s$?

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

There will be others later in the course.
Designing count as a Function

**count**

How many times does string \( y \) occur in a string \( x \)?

```
'ITH-JFK-ITH'

'ItH'
```

\[ n = \text{count}(y, x) \]
Designing count as a Method

Suppose

\[
x = \text{'ITH-JFK-ITH'}
\]

\[
y = \text{'ITH'}
\]

Instead of the usual function-call syntax

\[
n = \text{count}(y, x)
\]

we will write

\[
n = x.\text{count}(y)
\]
Methods: The Notation

Here is the syntax associated with using a string method:

```
name of string . name of method (arg1, arg2, ...)
```

Once again, the 'dot" notation
String Methods: count

```python
>>> s = 'ITH-JFK-ITH'
>>> m = s.count('ITH')
```

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

  m  -->  2
```

`s1.count(s2)` the number of occurrences of string `s2` in string `s1`
String Methods: count

```python
>>> s = 'ITH-JFK-ITH'
>>> m = s.count('LGA')
```

```
s --&gt; ITH - JFK - ITH
   0 1 2 3 4 5 6 7 8 9 10
m --&gt; 0
```

`s1.count(s2)` the number of occurrences of string s2 in string s1
count
The Formal Definition

If $s_1$ and $s_2$ are strings, then

$$s_1.count(s_2)$$

returns an int value that is the number of occurrences of string $s_2$ in string $s_1$.

Note, in general $s_1.count(s_2)$ is not the same as $s_2.count(s_1)$.
Using count: An Example

# 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

Illegal: n = A.count('a' or 'e' or 'I' or 'o' or 'u')
Designing `find` as a Function

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

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

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

<table>
<thead>
<tr>
<th>n1</th>
<th>n2</th>
<th>n3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
If \( s1 \) and \( s2 \) are strings, then

\[ s1 \text{ in } s2 \]

is a boolean-valued expression.

True if there is an instance of \( s1 \) in \( s2 \).
False if there is NOT an instance of \( s1 \) in \( s2 \).
These are equivalent:

```python
x = s1 in s2

x = s2.find(s1) >= 0
```
Designing replace as a Function

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

It would then be used like this: \( \text{sNew} = \text{replace}(s, s1, s2) \)
Designing replace as a Method

\[ s = 'one hundred and one' \]
\[ t = s.replace(' ',','+') \]

\[ s \rightarrow 'one hundred and one' \]
\[ t \rightarrow 'one-hundred-and-one' \]
The replace Method

```python
s = 'one hundred and one'
t = s.replace(' ', '')
```

- `s` -> `one hundred and one`
- `t` -> `onehundredandone`

*The null string has length 0.*

Replacing each blank with the "null string"
The replace Method

```
s = 'one hundred and one'
t = s.replace('x', '-')
```

```
s -> 'one hundred and one'
t -> 'one hundred and one'
```

No change if the character to be replaced is missing
The replace Method

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

\[
\begin{align*}
  s &\rightarrow \text{‘one hundred and one’} \\
  t &\rightarrow \text{‘seven hundred and seven’}
\end{align*}
\]
The replace Method

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

\[
s \rightarrow \ 'one hundred and one'
t \rightarrow \ 'one hundred and one'
\]

No change if the designated substring is missing
replace
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

s = 'xxx'
t1 = s.replace('x', 'o')
t2 = s.replace('xx', 'o')
t3 = s.replace('xx', 'oo')

t1 -> 'ooo'
t2 -> 'ox'
t3 -> 'oox'
replace does Not Replace

s.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 the its “updated” copy:

s = s.replace(s1, s2)
Strings are **immutable**. They cannot be changed.

Have to ```live with``` the replace function, slicing, and concatenation.

```python
s = 'abcdefgh'
s[5] = 'x'
```

```python
s = 'abcdefgh'
s = s[:5]+'x'+s[6:]`
Quickly Review Some Other String Methods
The upper and lower Methods

s = ‘A2sh?’
t1 = s.upper()
t2 = s.lower()
Some Boolean-Valued Methods

These methods return either True or False:

- islower()
- isupper()
- isalnum()
- isalpha()
- isdigit()
## Boolean-Valued Methods

<table>
<thead>
<tr>
<th>s</th>
<th>s.islower()</th>
<th>s.isupper()</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ab3?</code></td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td><code>AbcD</code></td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td><code>AB3</code></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><code>s.isalnum()</code></td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td><code>s.isalpha()</code></td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td><code>s.isdigit()</code></td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Useful String Constants

\[
\text{alpha} = \text{string.letters}
\]

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
Useful String Constants

specialChar = string.punctuation

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

TheDigits = string.digits

1234567890
The “Dot” Notation--Again

We have seen it with modules and import

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

```
pi=3.1416
```

The “folder metaphor.

The “dot” means “go inside and get this”
string is a “Special” Module

```
digits = '01234567890'
letters = 'abcdef etc'
punctuation = `!"#$ etc`
count isupper isalnum
find islower isalpha
replace isdigit
```

The “folder” metaphor.

The “dot” means “go inside and get this”

string is actually a “class”. More in a few lectures.