Advanced Strings

String objects have a variety of methods we can use to save time and add functionality. Let's explore some of them in this lecture:

In [1]:

```
s = 'hello world'
```

Changing case

We can use methods to capitalize the first word of a string, or change the case of the entire string.

In [2]:

```
# Capitalize first word in string
s.capitalize()
```

Out[2]:

```
'Hello world'
```

In [3]:

```
s.upper()
```

Out[3]:

```
'HELLO WORLD'
```

In [4]:

```
s.lower()
```

Out[4]:

```
'hello world'
```

Remember, strings are immutable. None of the above methods change the string in place, they only return modified copies of the original string.

In [5]:

```
s
```

Out[5]:

```
'hello world'
```

To change a string requires reassignment:

In [6]:

```
s = s.upper()
s
```

Out[6]:

```
'HELLO WORLD'
```

In [7]:

```
s = s.lower()
s
```

Out[7]:

```
'hello world'
```

Location and Counting

In [9]:

```
s.count('o') # returns the number of occurrences, without overlap
```

Out[9]:

```
2
```

In [10]:

```
s.find('o') # returns the starting index position of the first occurence
```

Out[10]:

```
4
```

Formatting

The `center()` method allows you to place your string 'centered' between a provided string with a certain length. Personally, I've never actually used this in code as it seems pretty esoteric...

In [11]:

```
s.center(20,'z')
```

Out[11]:

```
'zzzzhello worldzzzzz'
```

The `expandtabs()` method will expand tab notations `\t` into spaces:

In [12]:

```
'hello\thi'.expandtabs()
```

Out[12]:

```
'hello   hi'
```

is check methods

These various methods below check if the string is some case. Let's explore them:

In [13]:

```
s = 'hello'
```

`isalnum()` will return True if all characters in **s** are alphanumeric

In [14]:

```
s.isalnum()
```

Out[14]:

```
True
```

`isalpha()` will return True if all characters in **s** are alphabetic

In [15]:

```
s.isalpha()
```

Out[15]:

```
True
```

`islower()` will return True if all cased characters in s** **are lowercase and there is at least one cased character in** **s, False otherwise.

In [16]:

```
s.islower()
```

Out[16]:

```
True
```

`isspace()` will return True if all characters in **s** are whitespace.

In [17]:

```
s.isspace()
```

Out[17]:

```
False
```

`istitle()` will return True if **s** is a title cased string and there is at least one character in **s**, i.e. uppercase characters may only follow uncased characters and lowercase characters only cased ones. It returns False otherwise.

In [18]:

```
s.istitle()
```

Out[18]:

```
False
```

`isupper()` will return True if all cased characters in **s** are uppercase and there is at least one cased character in **s**, False otherwise.

In [19]:

```
s.isupper()
```

Out[19]:

```
False
```

Another method is `endswith()` which is essentially the same as a boolean check on `s[-1]`

In [20]:

```
s.endswith('o')
```

Out[20]:

```
True
```

Built-in Reg. Expressions

Strings have some built-in methods that can resemble regular expression operations. We can use `split()` to split the string at a certain element and return a list of the results. We can use `partition()` to return a tuple that includes the first occurrence of the separator sandwiched between the first half and the end half.

In [21]:

```
s.split('e')
```

Out[21]:

```
['h', 'llo']
```

In [22]:

```
s.partition('l')
```

Out[22]:

```
('he', 'l', 'lo')
```

Great! You should now feel comfortable using the variety of methods that are built-in string objects!