

Methods – on strings and other things

- Strings, revisited
- Objects and their methods
- Indexing and slicing
- Some commonly used string methods

Remember: What is a string?

- A string is a sequence of zero or more characters
- A string is delimited (begins and ends) by single or double quotes

```
poem = 'Ode to a Nightingale'
```

```
lyric = "Roll on, Columbia, roll on"
```

```
exclamation = "That makes me !#? "
```

- The empty string has zero characters (' ' or "")

Quote characters in strings

- You can include a single quote in a double quoted string or a double quote in a single quoted string

```
will = "All the world's a stage"  
ben = 'BF: "A penny saved is a penny earned"'
```

- To put a single quote in a single quoted string, precede it with the backslash ('\') or 'escape' character.

```
>>> will = 'All the world\'s a stage'  
>>> print(will)  
All the world's a stage
```

- The same goes for double quotes

```
>>> ben = "BF: \"A penny saved is a penny earned\""  
>>> print(ben)  
BF: "A penny saved is a penny earned"
```

Putting a format character in a string

- A format character is interpreted by the `print()` function to change the layout of text
- To put a format character in a string, precede it with the backslash (`'\'`)
- A newline is represented by `'\n'`

```
>>> juliette = 'Good night, good night\nParting is  
such sweet sorrow'  
>>> print(juliette)  
Good night, good night  
Parting is such sweet sorrow
```

- A tab is represented by `'\t'`

```
>>> tabs = 'col0\tcol1\tcol2'  
>>> print(tabs)  
col0   col1   col2
```

Index of string characters

- The first character of a string has index 0

```
>>> greeting = 'hello, world'
>>> greeting[0]
'h'
>>> 'hello, world'[0]
'h'
```

- You can also count back from the end of a string, beginning with -1

```
>>> greeting = 'hello, world'
>>> greeting[-1]
'd'
>>> 'hello, world'[-1]
'd'
```

Slicing a string

- You can use indexes to slice (extract a piece of) a string
- `aStr[i:j]` is the substring that begins with index `i` and ends with (but does not include) index `j`

```
>>> greeting = 'hello, world'
>>> greeting[1:3]
'el'
>>> greeting[-3:-1]
'rl'
```

- omit begin or end to mean 'as far as you can go'

```
>>> print(greeting[:4], greeting[7:])
hell world
```

- `aStr[i:j:k]` is the same, but takes only every `k`-th character

```
>>> greeting[3:10:2]
'l,wr'
```

Index/slice a string vs index/slice a list

How they're the same and how they're different

- SAME:
 - You can index a list or string by providing an integer index value, beginning with 0 from the left or -1 from the right [i].
 - You can slice a list or string by providing begin and end values ([i:j]) or begin, end and step values ([i:j:k])
 - You can omit begin or end ([i:] or [:j]) to mean 'as far as you can go'

List index vs string index (continued)

- DIFFERENT:
 - if you reference a single element of a list with the index operator ([i]), its type is the type of that element

```
>>> abc = ['a', 'b', 'c']
>>> abc[0]
'a'
>>> type(abc[0])
<class 'str'>
```
 - If you slice (extract a piece of) a list with begin and end ([i:j]) values, you get a sublist (type list)

```
>>> abc[0:2]
['a', 'b']
>>> type(abc[0:2])
<class 'list'>
```


String methods

- A method is a function that is bundled together with a particular type of object
- A string method is a function that works on a string
- This is the syntax of a method:

```
anObject.methodName(parameterList)
```

- For example,

```
>>> 'avocado'.index('a')  
0
```

returns the index of the first 'a' in 'avocado'

- You can also use a variable of type string

```
>>> fruit = 'avocado'  
>>> fruit.index('a')  
0
```

Method parameters

- Like any function, a method has zero or more parameters
- Even if the parameter list is empty, the method still works on the 'calling' object:

```
>>> 's'.isupper()  
False
```

- Here is a string method that takes two parameters:

```
>>> aStr = 'my cat is catatonic'  
>>> aStr.replace('cat', 'dog')  
'my dog is dogatonic'
```

Strings are immutable

- A string is immutable -- once created it can not be modified
- When a string method returns a string, it is a different object; the original string is not changed

```
>>> aStr = 'my cat is catatonic'
>>> newStr = aStr.replace('cat', 'dog')
>>> newStr
'my dog is dogatonic'
>>> aStr
'my cat is catatonic'
```

- However, you can associate the old string name with the new object

```
>>> aStr = 'my cat is catatonic'
>>> aStr = aStr.replace('cat', 'dog')
>>> aStr
'my dog is dogatonic'
```

Python string methods

- Python has many very useful string methods
- You should always look for and use an existing string method before coding it again for yourself. Here are some

```
s.capitalize()  
s.count()  
s.endswith() / s.startswith()  
s.find() / s.index()  
s.format()  
s.isalpha() / s.isdigit() / s.islower() / s.isspace()  
s.join()  
s.lower() / s.upper()  
s.replace()  
s.split()  
s.strip()
```

The split method

- The string method `split()` lets us separate a string into useful parts
- Common use: splitting a sentence into its words
- Splits by space characters by default, but you can give it a different 'separator' string

```
>>> s = "Captain, incoming transmission!"
>>> print(s.split())
['Captain,', 'incoming', 'transmission!']
```

```
>>> s = "a one, a two, a one two three four"
>>> print(s.split(', '))
['a one', 'a two', 'a one two three four']
```

The strip method

- The string method `strip()` “cleans” the edges of a string by removing the character(s) you specify (default: spaces)

```
>>> s = "(hello!)"
>>> print(s.strip("()!"))
hello
```

- The `string` module contains a useful variable for this, called `punctuation` (like how the `math` module has `pi`)

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

Using split() and strip() together

- The split method is useful for extracting words, and the strip method is useful for cleaning them up
- Remember that strip() is a string method, **not** a list method

```
>>> import string
```

```
>>> words = ['How', 'are', 'you,', 'sir?']
```

```
>>> print(s.strip(string.punctuation))
```

```
AttributeError: 'list' object has no attribute 'strip'
```

- So, how can we clean up every word in a sentence, once we've split it?

Using split() and strip() together

- The strip() method works on one “word” at a time
- So, take it one word at a time

```
>>> import string
    >>> words = ["It's", 'warm', 'today,', 'yeah?']
    >>> for item in words:
        print(item.strip(string.punctuation))

It's
warm
today
yeah
```

Side question: why can't we just use the replace() method to get rid of punctuation like this?

Python string method documentation

- You can find the meaning of each of these string methods in the Python documentation
- Some operations on strings also work with other sequence types, both mutable and immutable. For example:

```
x in s
x not in s
s + t
s*n / n*s
len(s)
min(s)
max(s)
```

Strings and the print() function

- The print() function always prints a string. The input() function always inputs a string.
- Every object in Python has a string representation. Therefore, every object can be printed.
- When you print a number, a list or a function it is the string representation of the object that is printed
- print() takes 0 or more arguments and prints their string representations, separated by spaces

```
>>> print('pi =', 3.14)
pi = 3.14
>>> def f():
    pass
>>> print(f)
<function f at 0x02C4BD20>
```

The print separator and end

- By default, `print()` separates multiple outputs with spaces
- You can change this to any string that you want, for example, a colon and a space (': ')

```
>>> print(1, 2, 3, sep=': ')  
1: 2: 3
```

- By default, `print()` ends its output with a newline ('\n')

```
>>> for i in range(3):  
    print(i)  
  
0  
1  
2
```

- You can change this, for example, to a hyphen

```
>>> for i in range(3):  
    print(i, end='-')  
  
0-1-2-
```

The string format method

- The string format() method allows you detailed control over what is printed and its arrangement (including alignment; width; your choice of date, time and number formats; and many other things).
- Here is an example of how s.format() can be used to control what is printed:

```
>>> print('{} is {}'.format('Big Bird', 'yellow'))  
Big Bird is yellow
```

```
>>> print('{} is {}'.format('Oscar', 'grumpy'))  
Oscar is grumpy
```