Intro to Coding with Python– Strings and String Methods

Dr. Ab Mosca (they/them)

Slides based off slides courtesy of Jordan Crouser (https://jcrouser.github.io/)

Reminder

- Your first homework is out today!
- Start early, work with 1-2 other people
- There is a how-to for GitHub Desktop on the course website under "Demos"

Plan for Today

- operations on strings
- accessing individual letters
- handy methods

(RECAP) Core concept 3: strings

- In CS, a sequence of characters that isn't a number is called a **`string`**
- In Python, a string is declared using **quotation marks**
- Strings can contain letters, numbers, spaces, and special characters
- Example:

```
x = "Ab"
x = "Bass Hall"
```

Operations on strings

- **Concatenation**: join two strings together with +, e.g.

```
"Ab" + " " + "Mosca"
```

- **Repetition** (i.e. self-concatenation): use *, e.g.

```
3 * "hi"
```

Multi-line strings

- **Problem**: a string that looks ugly when you try to type it all on one line, e.g.

```
desc = "This course is an introduction to co
mputer science and computer programming. The
programming language Python (Version 3) is u
sed to introduce basic programming skills an
d techniques."
```

- We can use **triple quotes** to make a multi-line string, e.g.

```
desc = """This course is an introduction to
computer science and computer programming.
The programming language Python (Version 3)
is used to introduce basic programming
skills and techniques."""
```

Escaping quotes

- **Problem:** you have a statement that contains both an apostrophe and double quotes, e.g.

```
"I can't!" he said
```

- What's the **issue** here?
  - If we try to wrap it in single quotes, Python thinks the apostrophe should end the string:

```
s = '"I can't!" he said'
```

  - If we try to wrap it in double quotes, Python thinks the double quote at the beginning of the sentence should end the string

```
s = ""I can't!" he said"
```

Escaping quotes

- **Problem:** you have a statement that contains both an apostrophe and double quotes, e.g.

```
"I can't!" he said
```

- **Solution**: protect ("escape") special characters using a backslash, e.g.

```
s = '"I can\'t!" he said'
```

or

```
s = "\"I can't!\" he said"
```

Accessing individual letters

- One way to think about a **`string`** is as a list of letters

Accessing individual letters

- One way to think about a **`string`** is as a list of letters:

```
name = "Jordan"
     ≈ ['J', 'o', 'r', 'd', 'a', 'n']
        0    1    2    3    4    5
```

Accessing individual letters

- One way to think about a **`string`** is as a list of letters:

```
name = "Jordan"
     ≈ ['J', 'o', 'r', 'd', 'a', 'n']
        0    1    2    3    4    5
```

- To print out the 3rd letter (position 2)?

```
print(name[2])
```

Accessing individual letters

- One way to think about a **`string`** is as a list of letters:

```
name = "Jordan"
     ≈ ['J', 'o', 'r', 'd', 'a', 'n']
        0    1    2    3    4    5
```

- How would I print out the last letter?

Accessing individual letters

- One way to think about a **`string`** is as a list of letters:

```
name = "Jordan"
     ≈ ['J', 'o', 'r', 'd', 'a', 'n']
        0    1    2    3    4    5
```

- How would I print out the last letter?

```
print(name[5])
```

"Slicing" (getting a substring)

- What about the 2nd - 5th letters (positions 1-4)?

```
print(name[1:5])
```

- What happens if we do this?

up to, but **not including**

```
print(name[2:])
```

- What about this?

```
print(name[-2:])
```

15-minute exercise

- Given this string:

```
*Untitled*
s = "BATMAN AND ROBIN"
Ln: 1    Col: 0
```

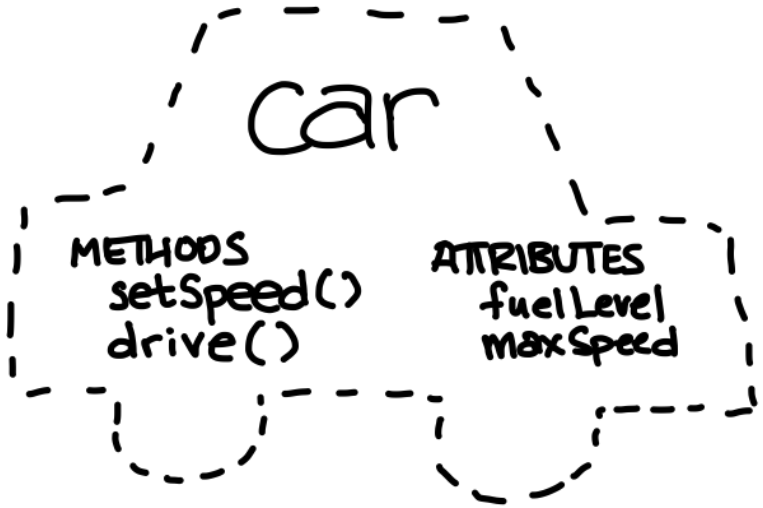
- Write a short program that uses **slicing** to produce:

```
Python 3.7.0 Shell
BATMAN
ROBIN
BATBIN
ROB ATM
Ln: 207    Col: 4
```

Discussion

What did you come up with?

Strings as objects



"object-oriented"

Useful methods for working with strings

- `s.lower()`: convert the string `s` to lowercase
- `s.upper()`: convert the string `s` to UPPERCASE
- `s.strip()`: remove whitespace from the start / end of `s`
- `s.replace('old', 'new')`: replace all occurrences of `'old'` in `s` by `'new'`
- `s.split(c)`: slice `s` into pieces using `c` as a delimiter
- `s.join(list)`: opposite of `split()`, join the elements in the list together using `s` as the delimiter, e.g.

```
'-'.join(['a', 'b', 'c']) # a-b-c
```

Fun fact

- **`strings`** in python are **immutable** (along with **`ints`**, **`floats`**, **`bools`**, and a few other built-in types)
- This means that when we call a method on them, the original isn't modified

15-minute exercise

- Work with 1 – 2 other people to write a short program that:
  - Takes as input from the user a string
  - Takes as input from the user a character (char1)
  - Takes as input from the user another character (char2)
  - Returns that the input string with all occurrences of char1 replaced with char2 and in all caps

```
Input a string: Ab Mosca
Input a character: a
Input another character: o
OB MOSCO
```