CS50 - SECTION 7

10/30/18

POST-MORTEM ON PSET 6

QUICK REMINDERS

1. What’s the difference between a for loop and a for-each loop?

QUICK REMINDERS

1. What's the difference between a for loop and a for-each loop?

The for loop lets us iterate a predetermined number of times.

C

```c
for(int x = 0; x < 5; x++) {
    printf("%i \n", x);
}
```

Python

```python
for x in range(5):
    print(x)
```

QUICK REMINDERS

1. What's the difference between a for loop and a for-each loop?

The for-each loop iterates once per each item in a data structure.

C

```c
int arr[5] = {0, 1, 2, 3, 4};

int arr_len =
sizeof(arr)/sizeof(arr[0]);

for(int x = 0; x < arr_len; x++)
{
   printf("%i \n", arr[x]);
}
```

Python

```python
arr = range(5)

for x in arr:
   print(x)
```

QUICK REMINDERS

2. What’s the difference between a method and a function?

QUICK REMINDERS

2. What's the difference between a method and a function?

A function is something that can be invoked on its own, whereas a method is a function that belongs to an object.

QUICK REMINDERS

2. What's the difference between a method and a function?

```
# An object class and method within it
class Student():
   def set_gpa(self, x):
       self.gpa = x

# A standalone function
def calculuate_gpa(classes):
   # Def goes here
```

QUICK REMINDERS

2. What's the difference between a method and a function?

You invoke methods all the time in Python!

`str.`**`islower`**`()`

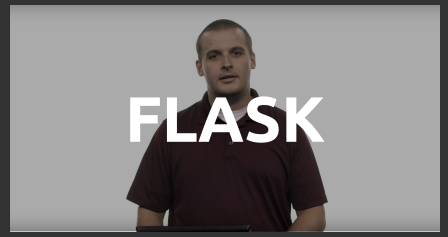
Return true if all cased characters [4] in the string are lowercase and there is at least one cased character, false otherwise.

QUESTIONS?

Any questions about Python before we move on?

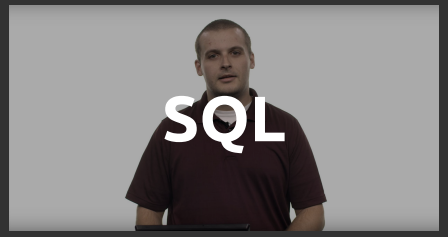
CONCEPTS DEEP-DIVE

“SHORTS” FOR THE WEEK



https://youtu.be/jOKx1JkRlho

https://youtu.be/xgyc_wOQt2Y



https://youtu.be/nfGiGSCEYRI

DECORATORS IN PYTHON

- Python allows us to work with a variety of different data types such as numbers, strings, objects, *and* functions
  - Yes, functions are themselves a data type!
  - The ability to use functions as a data type and to pass them as arguments indicates they are **first-class objects** in Python

DECORATORS IN PYTHON

```
# y = mx + b
def make_linear(slope, y_intercept):
    def linear(x):
        return (slope * x) + y_intercept

    return linear

# y = 2x + 4
my_linear = make_linear(2, 4)
print(my_linear(3))
```

DECORATORS IN PYTHON

- A **decorator** is a function that modifies the behavior of other functions, typically used to add extra functionality to them
  - They take advantage of what we just showed: that we can take a function as input, modify it, return it, etc.

DECORATORS IN PYTHON

What will this print?

```
def override(func):
    def incr():
        return func() + 1
    return incr

@override
def one():
    return 1

print(one())
```

DECORATORS IN PYTHON

What will this print?

```
def override(func):
    def incr():
        return func() + 1
    return incr

@override
def one():
    return 1

print(one())
```

This prints 2.

DECORATORS IN PYTHON

What will this print?

```
def override(func):
    def incr():
        return func() + 1
    return incr

@override
def one():
    return 1

print(one())
```

This prints 2.

Decorators are "syntactic sugar" for this:

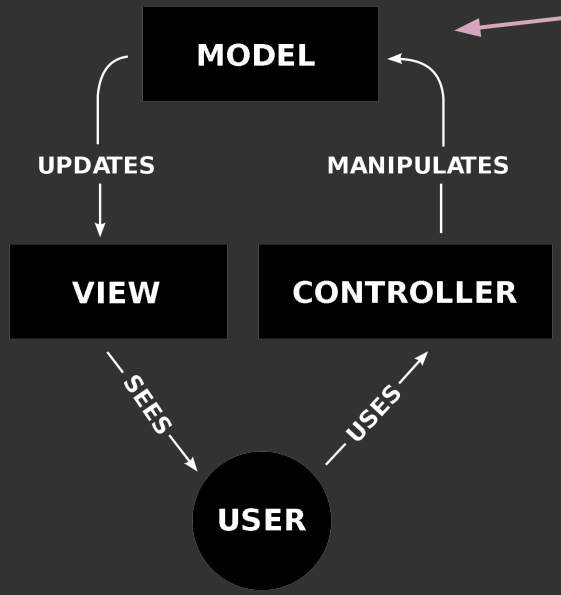
```
def override(func):
    def incr():
        return func() + 1
    return incr

def one():
    return 1

dec_one = override(one)

print(dec_one())
```

DECORATORS IN PYTHON

- You will often use the `@app.route()` decorator with Flask
- <u>For PS7:</u> You might also need the `@login-required` director to ensure certain functions are only called if the user is logged in

MVC PARADIGM

- Data Structures
- MySQL & Relational DBs



- HTML/CSS documents
- Jinja

- Flask Routes
- JavaScript

⚠ HOW DO I SERVE MY WEB APPS?

- We want to serve our web apps to the user, but the built-in server for Python is clunky and tedious to work with

⚠ HOW DO I SERVE MY WEB APPS?

```
from http.server import BaseHTTPRequestHandler, HTTPServer

class HTTPServer_RequestHandler(BaseHTTPRequestHandler):
   def do_GET(self):
       self.send_response(200)

       self.send_header("Content-type", "text/html")
       self.end_headers()

       self.wfile.write(b"<!DOCTYPE html>")
       self.wfile.write(b"<html lang='en'>")
       self.wfile.write(b"<head>")
       self.wfile.write(b"<title>hello, title</title>")
       self.wfile.write(b"</head>")
       self.wfile.write(b"<body>")
       self.wfile.write(b"hello, body")
       self.wfile.write(b"</body>")
       self.wfile.write(b"</html>")

port = 8080
server_address = ("0.0.0.0", port)
httpd = HTTPServer(server_address, HTTPServer_RequestHandler)

httpd.serve_forever()
```

⚠ HOW DO I SERVE MY WEB APPS?

```
from http.server import BaseHTTPRequestHandler, HTTPServer

class HTTPServer_RequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)

        self.send_header("Content-type", "text/html")
        self.end_headers()

        self.wfile.write(b"<!DOCTYPE html>")
        self.wfile.write(b"<html lang='en'>")
        self.wfile.write(b"<head>")
        self.wfile.write(b"<title>hello, title</title>")
        self.wfile.write(b"</head>")
        self.wfile.write(b"<body>")
        self.wfile.write(b"hello, body")
        self.wfile.write(b"</body>")
        self.wfile.write(b"</html>")

port = 8080
server_address = ("0.0.0.0", port)
httpd = HTTPServer(server_address, HTTPServer_RequestHandler)

httpd.serve_forever()
```

We have to manually specify:

- **HTTP Response Status Codes**
- **Packet headers**
- **HTML documents**
- **Server backend information**

✓ HOW DO I SERVE MY WEB APPS?

- We can use **Flask** to help streamline serving our web apps
- It is a *microframework* that vastly simplifies the process of setting up a server, identifying routes for our webpage, etc.

FLASK [application.py]

```
from flask import Flask


app = Flask(__name__)


@app.route("/")
def index():
    return "You are at the index!"
```

This is called a route.
Notice it's a decorator!

This corresponds to "`http://localhost/`".

FLASK [application.py]

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "You are at the index!"
```

You would run this application using `flask run` in the directory with `application.py` in your terminal

FLASK [application.py]

```
from flask import Flask


app = Flask(__name__)


@app.route("/")
def index():
    return "You are at the index!"
```

Flask rebuilds your apps on the fly—meaning as you edit source files, the server updates automatically!

DEMO: FLASK

How might we write a Flask web app to return the current time?

Hint: Check out the **`datetime`** library and the **`timezone`** (from **`pytz`**) libraries

DEMO: FLASK

```
from flask import Flask
from datetime import datetime
from pytz import timezone

app = Flask(__name__)

@app.route("/")
def time():
    now = datetime.now(timezone('America/New_York'))
    return f"The current date and time is {now}."
```

⚠ HOW DO I ADD MULTIPLE PAGES TO MY SITE?

- Most websites don’t just have one route, (“/”), that you visit
  - Websites like Facebook have literally *thousands* of routes
- Imagine how tedious it would be to keep track of all of the common HTML/CSS code between these routes?
  - You’d also have no abstraction—If Facebook wanted to change its logo or update its sidebar, it’d have to change these details on every single page—and there are thousands of pages...

✓ HOW DO I ADD MULTIPLE PAGES TO MY SITE?

- We can use **Jinja**, a templating engine, to resolve this issue
- Jinja allows us to create templates to abstract out common aspects of our HTML/CSS
- Jinja is also dynamic—It interfaces with Python well to allow us to easily iterate over data structures, render them in HTML, etc.

JINJA

- The most basic feature of Jinja is the <u>block</u>:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>My Webpage</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

JINJA

- The most basic feature of Jinja is the <u>block</u>:

This might be what our basic layout file looks like:

`layout.html`

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>My Webpage</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

JINJA

- The most basic feature of Jinja is the <u>block</u>:

This might be what our basic layout file looks like:

`layout.html`

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>My Webpage</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

This block is the dynamic content that will inserted later

JINJA

Here is what an actual instance of a sub-page might look like:

```
{% extends "layout.html" %}


{% block body %}
   <h1>First Page</h1>>
   <p>Welcome to the first page of my webpage!</p>
{% endblock %}
```

JINJA

Jinja also has control structures to interface with data structures from Python:

```
<h1>Members</h1>
<ul>
{% for user in users %}
 <li>{{ user.username }}</li>
{% endfor %}
</ul>
```

JINJA

```
{% if users %}
<ul>
{% for user in users %}
   <li>{{ user.username }}</li>
{% endfor %}
</ul>
{% endif %}
```

JINJA

- There are a lot more features to Jinja, far too many to discuss here
- Check out the documentation!
  - http://jinja.pocoo.org/docs/2.10/templates/

CLIENT-SIDE VS SERVER-SIDE

- When we're discussing web application development, we often refer to the distinction of the client-side vs. the server-side

CLIENT-SIDE VS SERVER-SIDE

- When we're discussing web application development, we often refer to the distinction of the client-side vs. the server-side
  - The **server-side** is what we've been focusing on today thus far—it is all the code that runs on the server and sends completed webpages to the user (e.g. Flask)
  - The **client-side** is code that runs *once* a user has downloaded a webpage (i.e. locally in their browser)

⚠ HOW DO I RUN CODE ON THE CLIENT-SIDE?

- What if you want to add dynamic interactivity to your applications once the user has already downloaded the page?
  - You could redirect them to another page as the content changes, but this would put a lot of strain on your server and is unfeasible for certain types of applications (e.g. Google Maps)

✓ HOW DO I RUN CODE ON THE CLIENT-SIDE?

- We can use **JavaScript**, which we saw last week, to achieve this!
- JavaScript runs locally on the client-side in a user's browser, allowing for interactivity after a page has already been loaded
- Note: Some browsers don't have JS enabled, but 99% do (StackOverflow), so most web developers rely on it anyways

DEMO: JAVASCRIPT FORM VALIDATION

- A common use of JavaScript is form validation—checking that a user has filled out a form correctly before allowing them to submit it to the server
- Let's implement a basic form validator:

http://bit.ly/2ze3gzo

DEMO: JAVASCRIPT FORM VALIDATION

SOLUTION:

http://bit.ly/2Sur9M1

⚠ HOW DO I ACHIEVE DATA PERSISTENCE?

- In the example from lecture, Prof. Malan created a webpage to register a new student and their freshman dorm
  - Initially, the web app just stored the values in a list
- What problems does this pose?

⚠ HOW DO I ACHIEVE DATA PERSISTENCE?

- In the example from lecture, Prof. Malan created a webpage to register a new student and their freshman dorm
  - Initially, the web app just stored the values in a list
- What problems does this pose?
  - Slow and limited to the memory of the computer running the Python program
  - No easy way to search/traverse data efficiently
  - No **persistence**—once the program ends, the data is lost

✓ HOW DO I ACHIEVE DATA PERSISTENCE?

- We can store data in an offline file and read/write to that in our application (AKA flat file)
- Or better yet, we can use a **database**, which is optimally designed to solve this problem!
  - More on this next week...

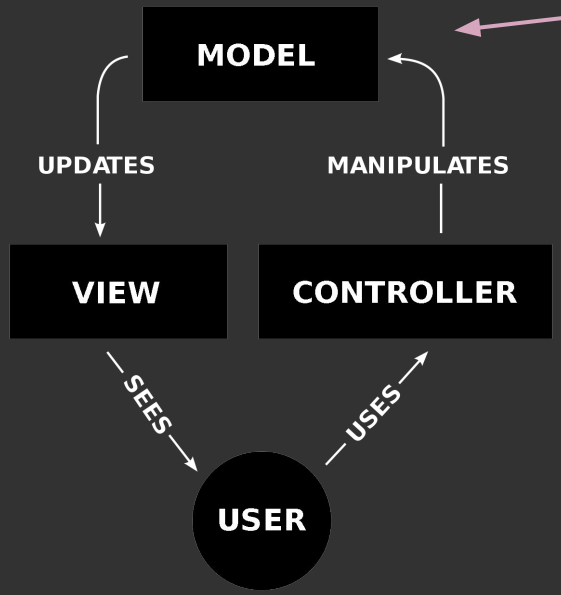
A REVIEW

PROBLEM	SOLUTION
HOW DO I SERVE MY WEB APP?	FLASK
HOW DO I ADD MULTIPLE PAGES TO MY SITE?	JINJA
HOW DO I RUN CODE ON THE CLIENT-SIDE?	JAVASCRIPT
HOW DO I ACHIEVE DATA PERSISTENCE?	FILES/DB's

Web development involves thousands of "problems" and infinitely many more solutions.

REVISITING THE MVC PARADIGM

- Data Structures
- MySQL & Relational DBs



- HTML/CSS documents
- Jinja

- Flask Routes
- JavaScript

PROBLEM SET 7 PREVIEW

PROBLEM SET 7 PREVIEW

Implement the following:

- `similarities`
- `Survey`

https://cs50.harvard.edu/2018/fall/psets/7/

FINAL PROJECT

THE FINAL PROJECT

- The purpose of the **final project** is to give you an opportunity to develop your own piece of software based off of what you have learned in CS50
- It is very *open-ended*—The topic, language/environment you program in, etc. are all up to you
- Specific guidelines for the final project can be found in the CS50 Syllabus

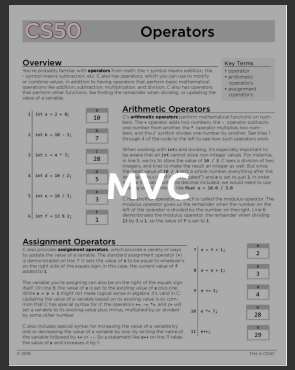
THE FINAL PROJECT

MILESTONE	DATE
Pre-Proposal	Tue 11/6, 11:59pm
Proposal	Tue 11/13, 11:59pm
Status Report	Tue 11/27, 11:59pm
CS50 Hackathon	Thu 11/29, 7pm – Fri 11/30, 7am
Implementation	Thu 12/6, 11:59pm
CS50 Fair	Fri 12/7, 12pm – 4pm

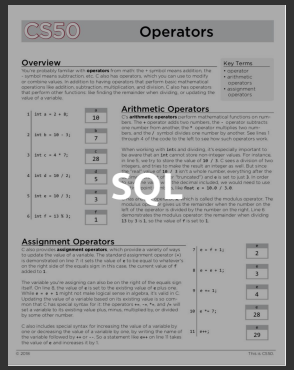
THE FINAL PROJECT

MILESTONE	DATE
Pre-Proposal	**Tue 11/6, 11:59pm**
Proposal	Tue 11/13, 11:59pm
Status Report	Tue 11/27, 11:59pm
CS50 Hackathon	Thu 11/29, 7pm – Fri 11/30, 7am
Implementation	Thu 12/6, 11:59pm
CS50 Fair	Fri 12/7, 12pm – 4pm

REFERENCE SHEETS

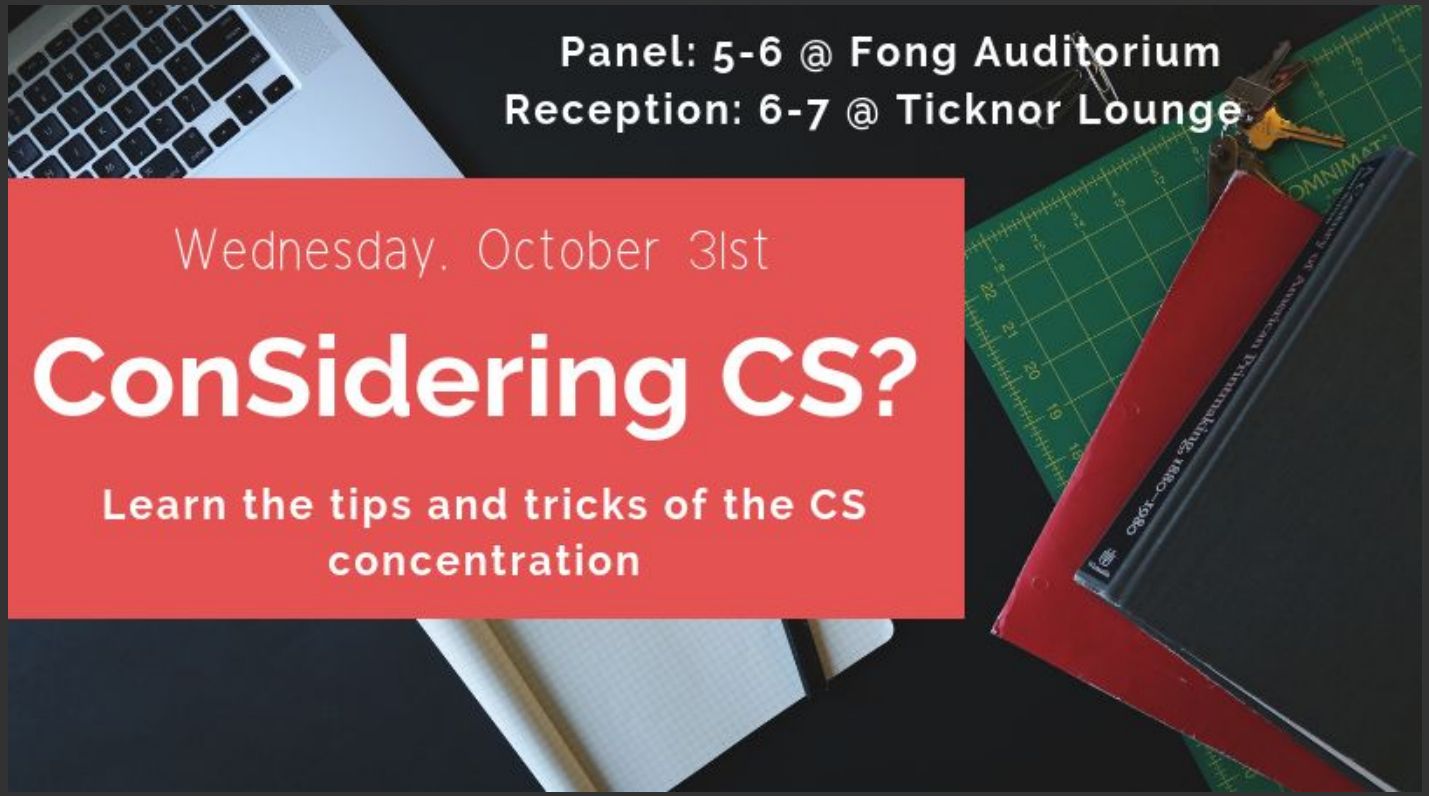


https://www.dropbox.com/sh/5y662ey1hc4sde4/AACcfWq8pwZ_PkfN_AekbkXa/MVC.pdf?dl=0



https://www.dropbox.com/sh/5y662ey1hc4sde4/AACjqHN3NtSKk4ShsRDFd_Sja?dl=0&m=&preview=SQL.pdf

CONSIDERING CS?



Seminar - An Introduction to jQuery

Time: Thursday, October 8, 4:30-5:30pm
Location: 67 Mt. Auburn St., the HSA building on the 4th floor

jQuery is helpful to solve the problem of writing client-side JavaScript code for web applications. It simplifies cumbersome JS, adds a variety of powerful dynamic features (e.g. animations), and is an industry-standard tool.

jQuery