Sending HTTP Requests:
In Python, the `requests` library provides us the ability to make HTTP requests to external APIs:

```
import requests
r = requests.get("https://www.colr.org/json/color/random")
print(f"Status Code: {r.status_code}")
print(f"Character Encoding: {r.encoding}")
```

- `requests.get(...)` sends a GET request,
- `requests.post(...)` sends a POST request,
- `requests.put(...)` sends a PUT request,
- ...etc...

The `requests` library is just a wrapper around the request and response from any HTTP web service:

```
print("== Headers ==")
for header in r.headers:
    print(header + " : " + r.headers[header])
print("== Payload (text) ==")
print(r.text)
print("== Payload (json) ==")
data = r.json()
data["colors"][0]["hex"]
```

Note that:

- `r.text` returns the response as a string (at attribute).
- `r.json()` parses it for us into a dictionary for us to index into quickly (it’s a function, requires the parameters)!

Receiving HTTP Requests:
The `flask` library allows us to receive HTTP requests:

```
from flask import Flask
app = Flask(__name__)
@app.route('/', methods=["GET"])
def index():
    return "index function!"
@app.route('/', methods=["POST"], methods=["GET"])
def post():
    return "post function!"
@app.route('/hello', methods=["GET", "POST"])
def hello():
    return "hello function!"
@app.route('/hello/<id>')
def with_id(id):
    return f"with_id function: {id}"
@app.route('/hello')
def mystery():
    return "mystery function!"
```

What happens with the following requests:

1. GET /
2. POST /
3. PUT /
4. GET /hello/
5. GET /hello
6. POST /hello
7. PUT /hello
8. GET /hello/42
9. GET /hello/world
Operating Systems: A Great Illusionist
Throughout this entire course, we have discussed how the operating system abstracts away the complexity of real systems:

- As a process, it appears that we have ________________.
- ...and has ________________.

Virtualization

Q: What is a “machine”?

Big Idea:
Map a ________________ onto a ________________.

- All states $S_x$ can be represented on a host system $H(S_x)$.
- For all sequences of transitions between $S_1 \Rightarrow S_2$, there is a sequence of transitions between $H(S_1) \Rightarrow H(S_2)$.

- Language Virtualization:
- Process Virtualization:
- System Virtualization:

Language Virtualization: Example w/ a JVM

<table>
<thead>
<tr>
<th>Initial State ($S_1$):</th>
<th>Transition ($S_1 \Rightarrow S_2$):</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System #1</strong>&lt;br&gt;COPY r1 1&lt;br&gt;SHIFTL x 2&lt;br&gt;ADD x r1</td>
<td><strong>System #2</strong>&lt;br&gt;COPY r1 x&lt;br&gt;SHIFTL x&lt;br&gt;SHIFTL x&lt;br&gt;ADD x r1</td>
</tr>
</tbody>
</table>

Final State ($S_2$):

System Virtualization: Containers

A commonly deployed form of visualization is ____________.

- As a developer of a Docker, you build a Dockerfile that specifies the snapshot of the system you want to provide and then build that snapshot into a ____________.
- Create a Dockerfile to specify how to build the image:

```
FROM gcc:latest
COPY ./docker/entrypoint.sh /
RUN chmod +x entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
```

To build it:

```
$ docker build --tag mp3-docker .
```

- As a user of a container, you specify the name of the docker image that you want to use to launch that image:

```
$ docker run -it --rm -v "pwd":/mp3  mp3-docker "make"
$ docker run --rm -it -p 27017:27017 mongo
```

System Virtualization: Hypervisor

...has this changed our industry??

Your CS 340 Virtual Machine:
The CS department has a “private cloud”, containing cloud services for us to use!

- As part of being in CS 340, you have your very own VM!
- This machine is a “private cloud” solution to “Infrastructure as a Service” and is effectively identical to AWS EC2 or other compute services.