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

```python
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:

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

```python
from flask import Flask
app = Flask(__name__)
@app.route('/', methods=['GET'])
def index():
    return "index function!"

@app.route('/', methods=['POST'])
def post():
    return "post function!"

@app.route('/hello', methods=['GET'])
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 ____________________________!

Do we need additional abstractions?

---

Virtualization

Big Idea:

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

What is a “machine”?

- Language Virtualization:

- Process Virtualization:

- System Virtualization:

---

Language Virtualization: Example w/ a JVM

<table>
<thead>
<tr>
<th>Initial State ($S_i$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition ($S_i \Rightarrow S_2$):</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System #1</th>
<th>System #2</th>
<th>System #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY r1 1</td>
<td>COPY r1 x</td>
<td>COPY r1 x</td>
</tr>
<tr>
<td>SHIFTL x 2</td>
<td>SHIFTL x</td>
<td>ADD r1 x</td>
</tr>
<tr>
<td>ADD x r1</td>
<td>SHIFTL x</td>
<td>ADD r1 x</td>
</tr>
<tr>
<td>ADD r1 x</td>
<td>ADD r1 x</td>
<td>ADD r1 x</td>
</tr>
<tr>
<td>ADD r1 x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Final State ($S_2$):

---

Process Virtualization: Example w/ Rosetta and the M1 chip

<table>
<thead>
<tr>
<th>Initial State ($S_i$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition ($S_i \Rightarrow S_2$):</td>
</tr>
<tr>
<td>Final State ($S_2$):</td>
</tr>
</tbody>
</table>

---

System Virtualization: Your CS 240 Virtual Machine / EC2

- Type 1 Hypervisor:

- Type 2 Hypervisor:

Q: How has this changed the deployment of software?