CSC110 Lecture 31: Extending Our Food Delivery System

David Liu, Department of Computer Science

*Navigation tip for web slides: press **?** to see keyboard navigation controls.*

Announcements and Today's Plan

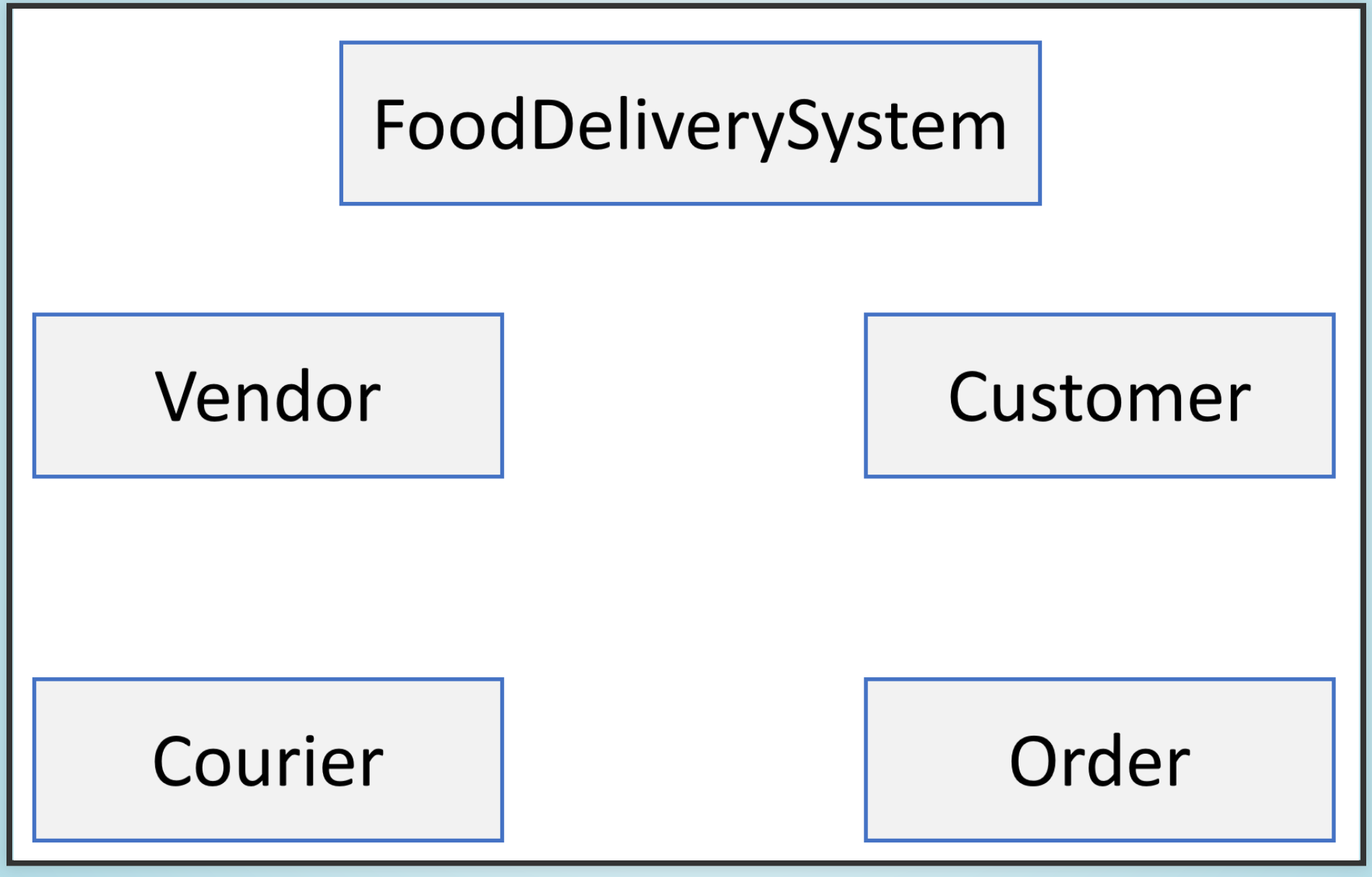
Announcements

- Please complete the PythonTA Survey 2
  - Due December 8
- Last lecture tomorrow!
- Final exam info will be posted after class today!

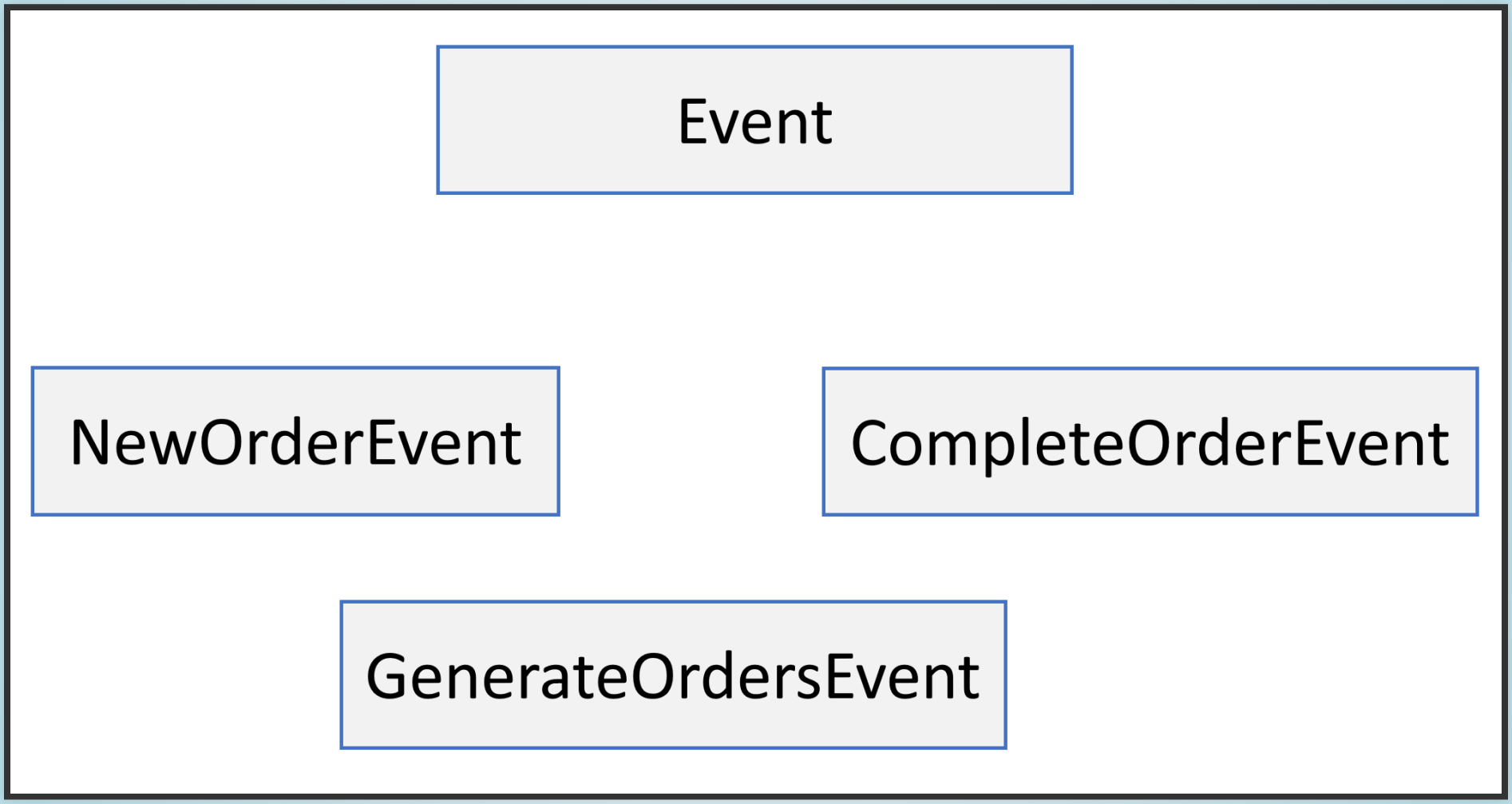
Story so far: problem domain

Seeing the proliferation of various food delivery apps, you have decided to create a food and grocery delivery app that focuses on students. Your app will allow student users to order groceries and meals from local grocery stores and restaurants. The deliveries will be made by couriers to deliver these groceries and meals—and you'll need to pay the couriers, of course!

Story so far: entities and a "manager" class



Story so far: events



Story so far: the discrete-event simulation loop

```
def run_simulation(initial_events: list[Event],
                   system: FoodDeliverySystem) -> None:
    events = EventQueueList()
    for event in initial_events:
        events.enqueue(event)

    while not events.is_empty():
        event = events.dequeue()
        new_events = event.handle_event(system)
        for new_event in new_events:
            events.enqueue(new_event)
```

Today, you'll learn to...

1. Explain the purpose of application logs and create logs using the built-in `logging` module.
2. Export data from an application to perform data analysis, using the `pandas` library.
3. Describe how to connect our food delivery system to a web application, using the `flask` library.

Application logging

Our `FoodDeliverySystem` class provides mutating methods to update the state of the system (e.g. `add_customer`, `place_order`).

But how do we **keep track of changes**?

Logging

In software, **logging** is the act of recording events in an application. We use the term **application log(s)** to refer to the records.

Question: What does a “record” of an event look like?

Many different possibilities!

- Text
- Custom data type (e.g., `EventRecord`)
- Entry in a csv file or database

Python's `logging` module

The Python `logging` module "defines functions and classes which implement a flexible event logging system for applications and libraries."

`logging` uses text to represent events.

```
>>> import logging
>>> logging.basicConfig(level='INFO')   # Set the "log level" to I
>>> logging.info('David is cool')       # Log a text record
INFO:root:David is cool
```

Demo: logging `FoodDeliverySystem` method calls.

Logging vs. printing (1)

Students often use `print` as a form of text-based logging. Why would we use `logging` instead?

Reason 1: saving logs to a file

```
logging.basicConfig(level='INFO',
                    filename='my_food_delivery_log.txt')
```

Demo!

Logging vs. printing (2)

Students often use `print` as a form of text-based logging. Why would we use `logging` instead?

Reason 2: differentiate between different types, or **levels**, of events.

Typical log levels

Log level	Description	`logging` function
`FATAL`/ `CRITICAL`	An error occurred that caused the entire application to stop working.	`logging.critical`
`ERROR`	An error occurred in the application.	`logging.error`
`WARNING`	An unexpected event occurred, but an error was not caused. This may or may not indicate a problem.	`logging.warning`
`INFO`	An (expected) event occurred. Useful for keeping track application events.	`logging.info`
`DEBUG`	Information recorded for debugging purposes. Often used for events tied to the (private) implementation/code.	`logging.debug`

Model Accuracy and Data Analysis

Our food delivery simulation represented a model of real-world events (adding new customers, placing orders, etc.)

We made many design decisions in our representation:

- How many customers/vendors do we have, and where are they located?
- How often do customers place orders? How do they pick where to order food?
- How are couriers actually assigned to deliveries?
- How long do deliveries take? What factors impact delivery time?

In general, the difficulty of creating a good computational model of a real-world system depends on the complexity of the system.

What do we mean by good computational model?

Unlike the functions we've studied throughout this course, "correctness" isn't the goal.

Key idea: we care about how closely a model can predict real-world data.

Suppose we run our food delivery simulation with Toronto data for November 2022.

We can then compare resulting event data against real-world data from, e.g., DoorDash or UberEats (assuming we have access to that data!).

- Number of orders placed
- Average time taken for deliveries
- Average cost per order
- Number of new customers
- Number of existing customers who didn't use the app at all

Recall: working with tabular data

ID	Civic Centre	Marriage Licenses Issued	Time Period
1657	ET	80	January 2011
1658	NY	136	January 2011
1659	SC	159	January 2011
1660	TO	367	January 2011
1661	ET	109	February 2011
1662	NY	150	February 2011
1663	SC	154	February 2011
1664	TO	383	February 2011

Orders as tabular data

Customer	Vendor	Courier	Start Time	End Time
David	Kinton Ramen	Mario	2022/12/05 11:30am	2022/12/05 12:13pm
Tom	The Cube	Mario	2022/12/05 2:30pm	2022/12/05 2:35pm
...	...	...	...	...

Enter `pandas`

`pandas` is a Python library for doing data analysis and manipulation.

The key data type is `pandas.DataFrame`, which represents tabular data.

Demo!

From System to Web Application

So far, all of our code for representing a food delivery system has run on a single computer.

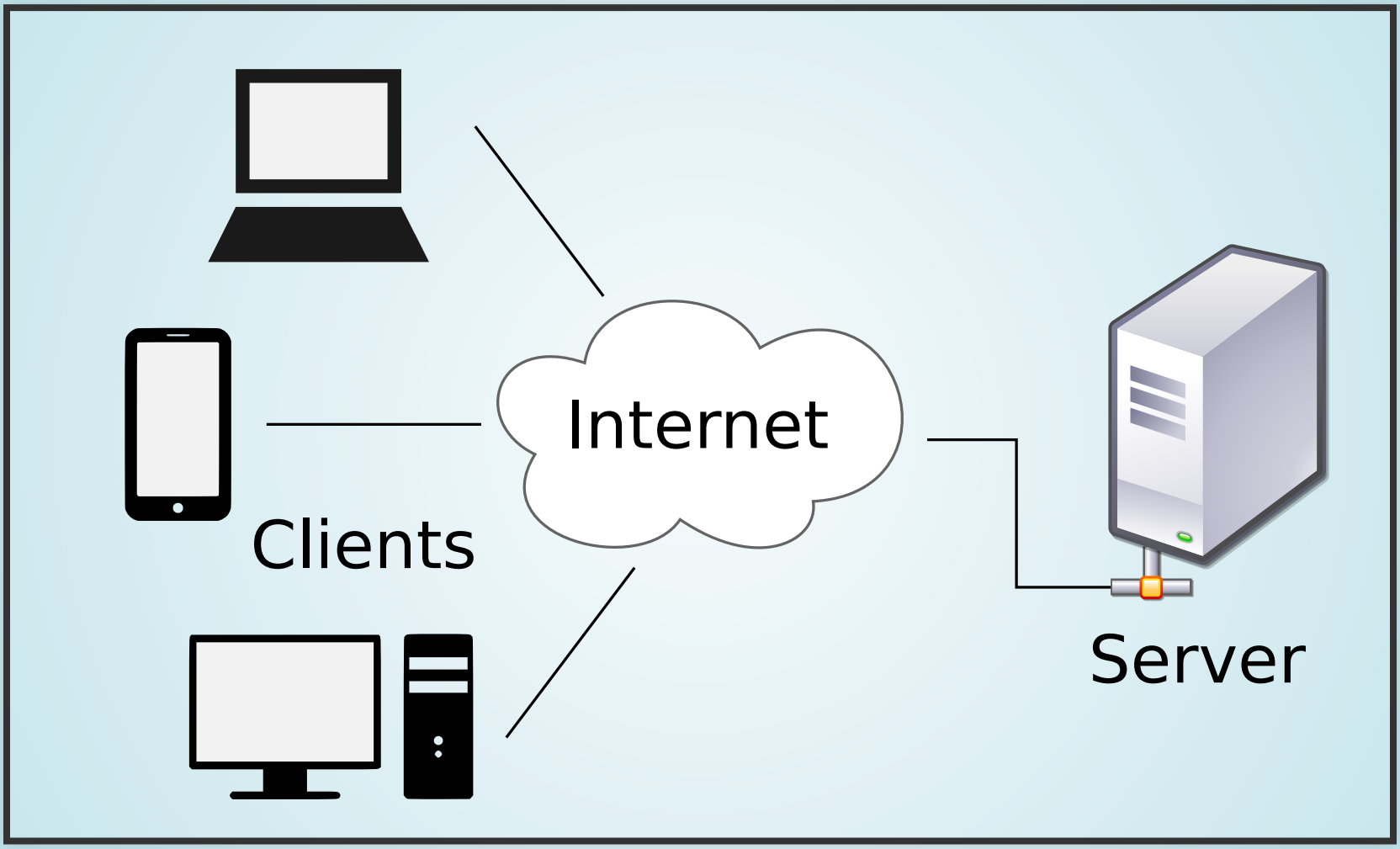
To turn our code into a “real” application, we need to let other people interact with it somehow.

Client-server model

In the **client-server model** of computing, multiple computers (or multiple programs) communicate with each other. Each program has one of two roles:

- A **server** is a computer that provides a service (e.g., a food delivery system)
- A **client** is a computer that requests a service (e.g., someone who wants to order food).

Typically, one server provides services for multiple clients.


Internet
Clients
Server

Requests and responses

Typically, a server does not initiate communication with a client.

Instead:

1. When the client wants a service, it sends a **request** to the server, specifying what service is being requested and any data required for the service.
2. The server determines the type of request being made, and then sends a **response** back to the client.

Requests and responses are usually encrypted, using ideas from earlier in this course!

A **web application** has code consisting of two parts:

1. The **back-end**: code that runs on the server. Responsible for:
    - Receiving requests and generating responses
    - Managing all application data
    - Keeping track of users; authentication and authorization
    - Mediating communications between clients
2. The **front-end**: code that runs on the client (e.g. web browser or mobile phone). Responsible for:
    - Initiating requests to the server
    - Displaying a user interface to help the user make requests
    - Interacting with other software on the client computer

Enter `flask`

`flask` is a Python library for creating web applications.

We'll use it today to focus on writing back-end code for our food delivery system server.

Web applications, URLs, and requests

When a user visits our website, they trigger a request depending what URL they visit. The exact URL is used to determine the type of request. Example:

- https://www.instagram.com
- https://www.instagram.com/uoftcssu
- https://www.instagram.com/explore/tags/cuteanimals

Routing is the (back-end) process of taking a URL* and determining what type of request is being made.

*and some other request metadata

```python
@app.route('/')
def home() -> str | Response:
    """Display the home page."""


@app.route('/vendors')
def vendors() -> str | Response:
    """Display all vendors."""


@app.route('/orders')
def orders() -> str | Response:
    """Display all orders."""


@app.route('/place_order')
def place_order() -> str | Response:
    """Handle users placing orders."""
```

HTML responses

The most common type of response when viewing a website is plain text written in a language called **HyperText Markup Language (HTML)**. This language is used to specify the structure and contents of a website.

Demo!

Another language, **Cascading Style Sheets (CSS)**, is used to describe the presentation (or "style") of a website's HTML content.

Placing orders with forms (1)

To allow a customer to place an order, the data for the order needs to be part of a request.

A `<form>` is an HTML element represent a place where a user can input data to send as part of a request.

A `<form>` contains `<input>` elements. Each `<input>` element has a `type` attribute specifying the type of input.

Examples:

- `<input type="text">`
- `<input type="number">`
- `<input type="date">`
- `<input type="file">`

Placing orders with forms (2)

The server access the request's **form data** to place an order:

```
vendor_name = request.form['vendor-name']
vendor = SYSTEM.get_vendor(vendor_name)
customer_name = request.form['customer-name']
customer = SYSTEM.get_customer(customer_name)

food_items = {}
for food in vendor.menu:
    quantity = int(request.form[f'{food}-quantity'])
    if quantity > 0:
        food_items[food] = quantity

new_order = Order(customer=customer, vendor=vendor,
                  food_items=food_items,
                  start_time=datetime.datetime.now())

SYSTEM.place_order(new_order)
```

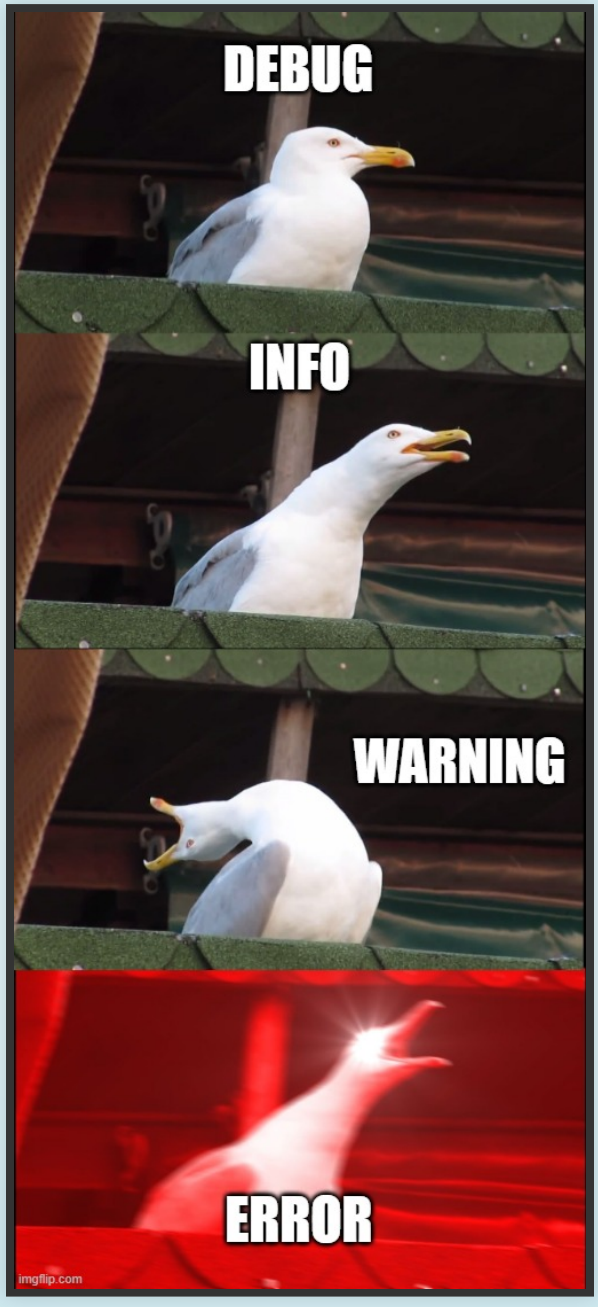
Summary

Today, you learned to...

1. Explain the purpose of application logs and create logs using the built-in `logging` module.
2. Export data from an application to perform data analysis, using the `pandas` library.
3. Describe how to connect our food delivery system to a web application, using the `flask` library.

Homework

- Review today's posted code
- Please complete the PythonTA Survey 2
  - Due December 8
- **Last lecture** tomorrow (say bye to your instructor 😢)
- Check out information about the final exam
  - to be posted after lecture


DEBUG
INFO
WARNING
ERROR
imgflip.com