MAKE SCHOOL

Forms

WEB 1.1

Agenda

- Learning Outcomes
- Review: Route Variables
- Forms
- **BREAK**
- Action & Method
- Getting Form Results
- Lab

Learning Outcomes

By the end of today, you should be able to...

1. **Explain** how forms are used in many of the websites you regularly visit.
2. **Use** an HTML form to collect user data & process the results.
3. **Explain** the difference between the GET & POST methods.

Route variables, or: How to make Infinite Web Pages

Routes

Last time, we created some **route functions** to serve various web pages. We ended up with something like this:

```
# app.py
from flask import Flask
app = Flask(__name__)

@app.route('/profile/Rey')
def my_page():
    return "<h1>Hello, Rey!</h1>"

if __name__ == '__main__':
    app.run(debug=True)
```

Routes

BUT... that gives us **only one web page**! What if I need a profile page for every single person here?! Man, that'd take a while...

```
...

@app.route('/profile/Luke')
def luke_profile():
    return "Hello, Luke!"

@app.route('/profile/Chewbacca')
def chewbacca_profile():
    return "Hello, Chewbacca!"

@app.route('/profile/Padme')
def padme_profile():
    return "Hello, Padme!"
```

```
@app.route('/profile/Leia')
def leia_profile():
    return "Hello, Leia!"

@app.route('/profile/Lando')
def lando_profile():
    return "Hello, Lando!"

@app.route('/profile/Jar-Jar')
def jarjar_profile():
    return "Hello, Jar-Jar!"

...
```

Routes

It turns out, there is a way to do that! We can make 5, 10, 20 web pages all at once... or even *infinite* web pages! Check out this page for an example (try typing your name into the URL)!

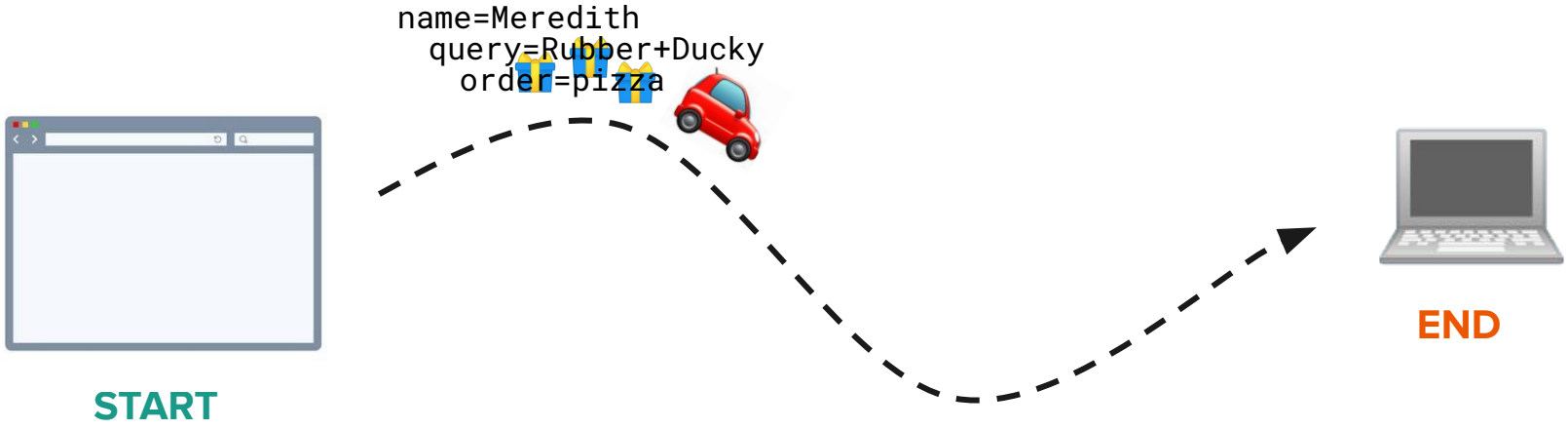
So... how do we do it?!?

Route Variables!!!

Routes

A **route variable** is an extra piece of data that is sent along the route by the **client** (your browser) to the **server** (in this case, your laptop).

Route variables are always sent as **key-value pairs**.



Route variables

Open up the `app.py` file from last class and add the following route:

```
@app.route('/profile/<users_name>')
def profile(users_name):
    return "Hello " + users_name
```

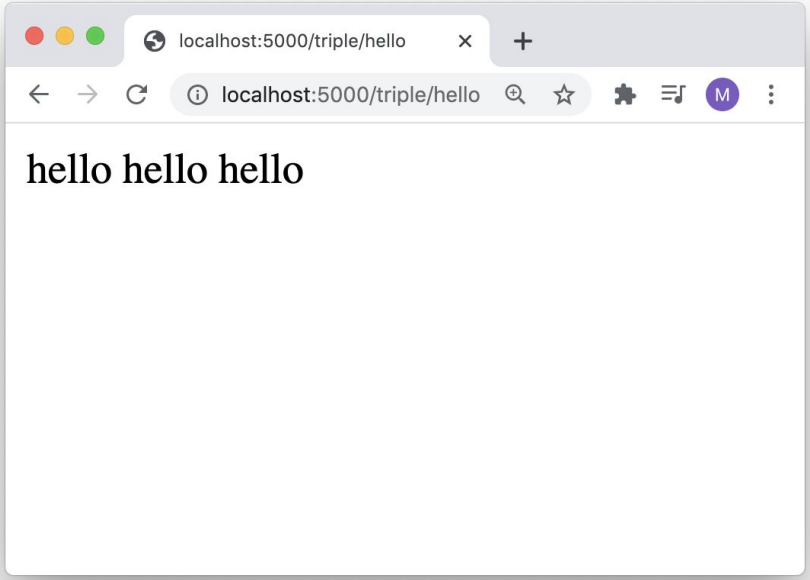
Whoa - we just made infinite URLs! With only 3 lines of code!!!!

🗣️ What are **3 things you notice** about the `users_name` variable? Type into the chat!

Review: Route Variables

Warm-Up (15 minutes)

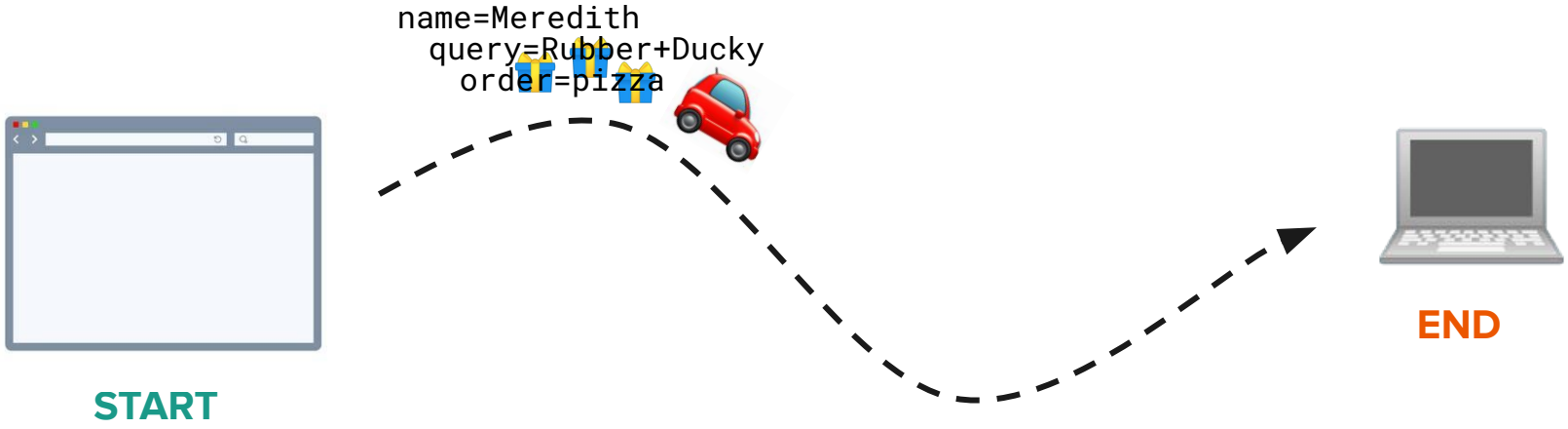
Open breakout rooms of pairs, have 1 person share their screen. Create a URL for **/triple/<word>** which, when you type in a word, will show you the word 3 times:



Route Variables

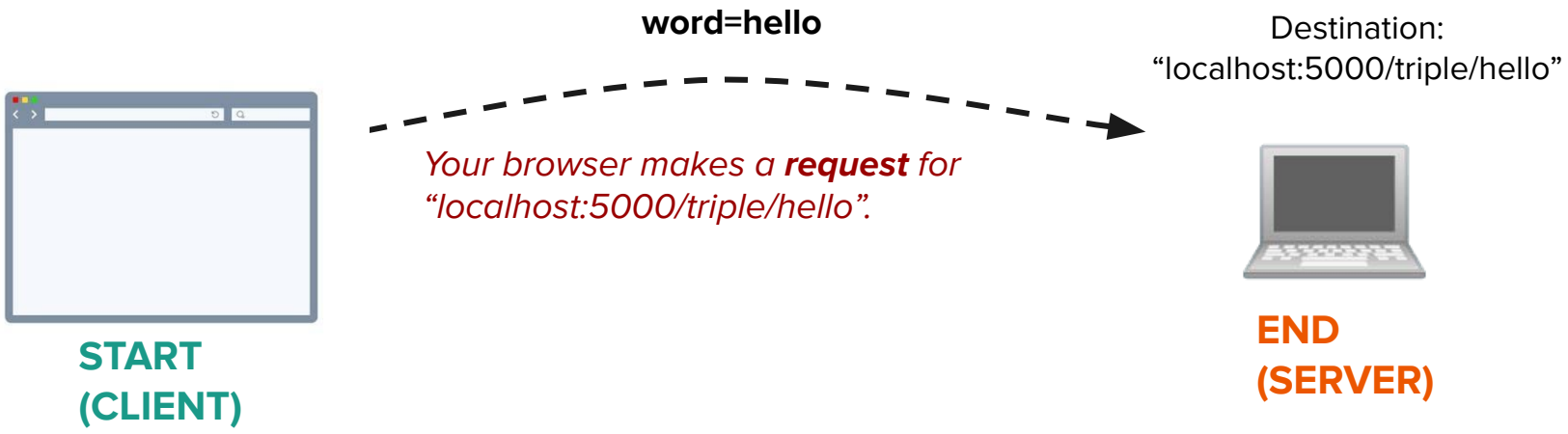
A **route variable** is an extra piece of data that is sent along the route by the **client** (your browser) to the **server** (in this case, your laptop).

Route variables are always sent as **key-value pairs**.



Route Variables

We are, in effect, **sending the route variable** as a **key-value pair** as part of our request.



Route Variables

Then, the server **uses that route variable to construct a response**, and **sends the response back to the client**.

Route Variables

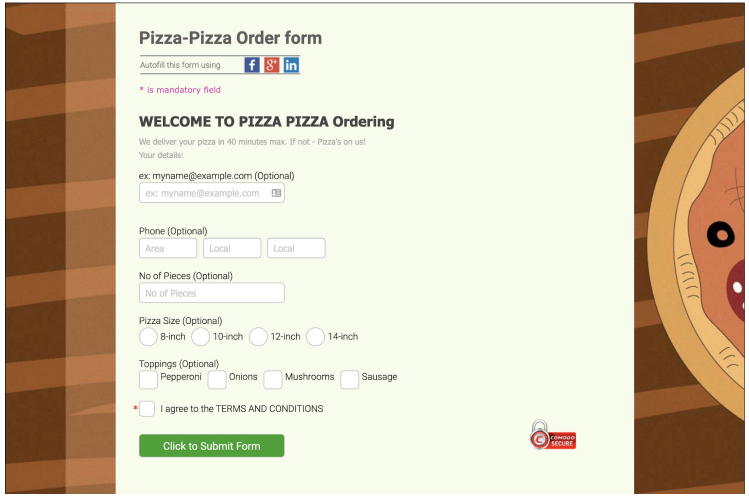
Route variables are one way to **collect data from the user**. (If the user clicks on `'/profile/Ducky'`, you've just collected data on which profile they want to see!)

But, it isn't the *only* way, or even (depending on what kind of data) the *best* way!

Introducing: Forms

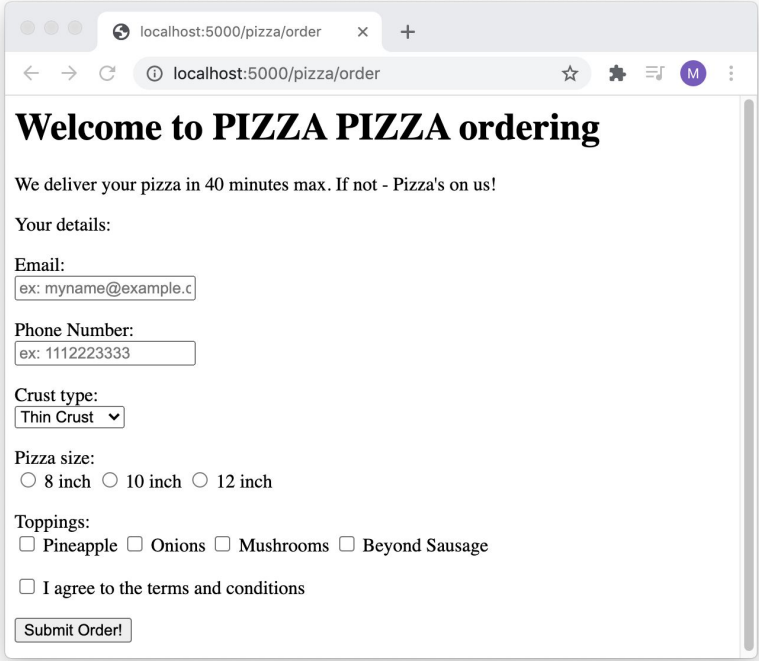
Forms

You are the owner of a pizza shop, and you'd like to move your business online. You'll need a way to collect **pizza orders** from your users! So, how do we do that?



Forms

Clone the starter code, run the server, visit localhost:5000 in your browser, and go to the "Complex Order Form".



* no cows were harmed in the making of this website

Activity (10 minutes)

MAKE SCHOOL

Take a look at the server code for the `/complex` route and, in groups of 3, answer the following questions:

1. What **input types** exist on the page (e.g. drop-down)? You should be able to list 5 different input types.
2. How does the **code** indicate **what type of data** is being collected?
3. What is the **key-value pair** that is being sent for each piece of data?

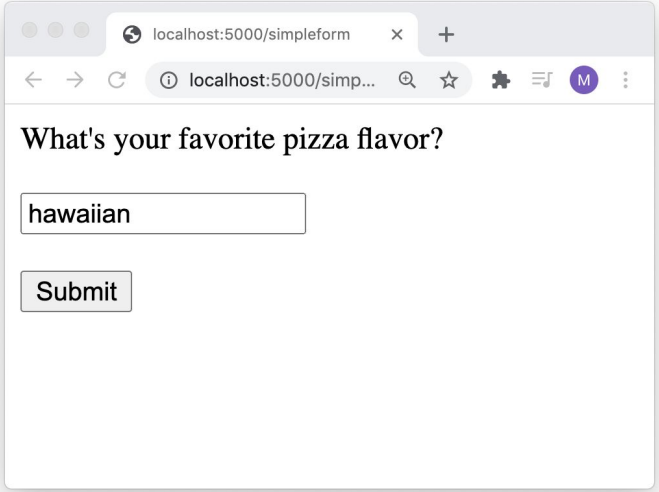
   HINT: Press the “Submit Order” button and look at the URL bar!

Let’s break it down...

Form Essential Parts

Any form on the Web has 3 essential parts: The **<form> tags**, at least one **input tag**, and a **submit button**.

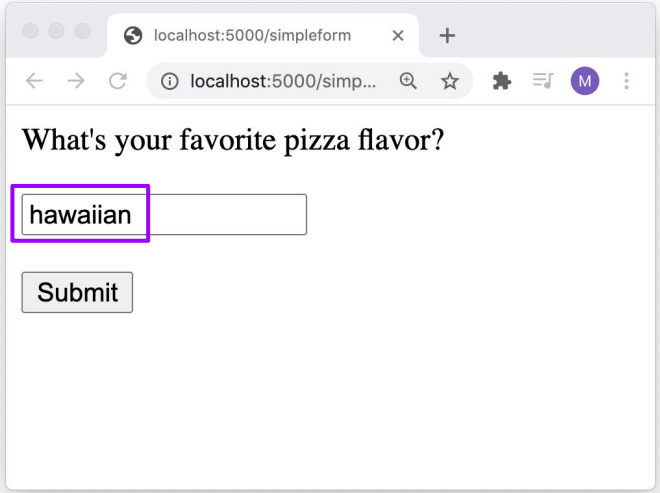
```
<form>
   What's your favorite pizza flavor?
   <input type="text" name="pizza_flavor">
   <input type="submit" value="Submit">
</form>
```



Form Essential Parts

The **name** field is the **key** of the **key-value pair** that is sent to the server. *If the name field is missing, no data will be sent!*

```
<form>
   What's your favorite pizza flavor?
   <input type="text" name="pizza_flavor">
   <input type="submit" value="Submit">
</form>
```



client → pizza_flavor=hawaiian → server

Form Essential Parts

Modify the "simple_pizza_order" route in the Pizza Ordering site. Give it the following code:

```
@app.route('/simple')
def simple_pizza_order():
    return """
    <form>
        What's your favorite pizza flavor? <br><br>
        <input type="text" name="pizza_flavor"> <br><br>
        <input type="submit" value="Submit">
    </form>
    """
```

Now, try it out. **What happens** when you submit the form? What do we ***want*** to happen?

Action & Method

Form Essential Parts

There are 2 form attributes that are usually included: the **action** and the **method**.

Where are these results going?

How are they being sent there?

```
<form action="/results" method="GET">
   What's your favorite pizza flavor?
   <input type="text" name="pizza_flavor">
   <input type="submit" value="Submit">
</form>
```

The **action** says *which URL* to send the results to.

The **method** says *how to send* the results. It can only be **GET** or **POST** (GET is the default). We'll talk about the differences between these later.

Form Essential Parts

Update your previous code for the '`/simple`' route with the following changes:

1. Send the form results to the '`/simple_results`' URL. (HINT: use the 'action' attribute.)
2. Change the form method to be '`POST`'.

Then, add another route:

```
@app.route('/simple_results', methods=['GET', POST'])
def simple_pizza_esults():
    return "Your order has been received!"
```

Try out your code again. **What happens** when you press the submit button?

What's different now?

GET vs. POST

When to use the **GET** method vs. the **POST** method?

- Use **GET** to request data or information.
  - Example: sending a search query in a search form
- Use **POST** to send data to update a resource, or otherwise change the state of the system.
  - Example: sending username/password in a login form
  - Example: creating a new database object

Check for understanding

Which method (**GET** or **POST**) would you use for the following scenarios?

- A “Contact Me” form
- A “Subscribe to this Newsletter” form
- A “Search for Your Pet” form

Use **POST** whenever we are changing the state of the system.

Break - 10 min

Getting Form Results

Form Results

OK, so this is all well and good, but... we haven't **actually done anything** with the results of the form yet.

How do I, the pizza shop owner, get my pizza orders??

Introducing... `request.args` & `request.form`!

Form Results

`request.args` is like a **suitcase** that holds all of the **key-value pairs** that were typed in by the user. It only works for **GET** routes.

```
email=me@me.com
phone=3334445555
crust=thick
size=10
toppings=pineapple
toppings=onions
accepted_terms=yes
```

request.args

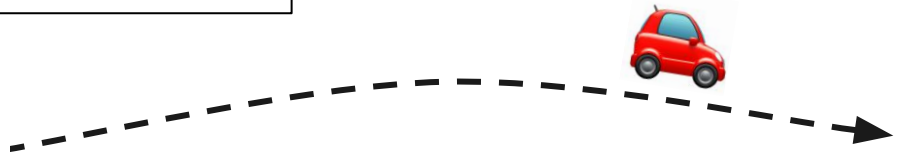
<form method="GET">

Form Results

`request.form` works exactly the same way, but only works for **POST** routes.

```
email=me@me.com
phone=3334445555
crust=thick
size=10
toppings=pineapple
toppings=onions
accepted_terms=yes
```

request.form



<form method="POST">

Form Results

Let's practice using our form results in code!

Modify the '`/simple_results`' route, as follows:

```
@app.route('/simple_results')
def simple_pizza_results():
    print(request.args)
    return 'Your order has been received!'
```

Now, run your server again and submit the form. *What do you see in the console?*

Type your answer into the chat!

Form Results

You should have seen something like this:

```
 * Detected change in '/Users/meredithmurphy/course-work/web1.1/names_app/app.py'
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 246-619-731
ImmutableMultiDict([('pizza_flavor', 'hawaiian')])
```

So... What is an `ImmutableMultiDict` ???

- It's like a dictionary
- It can hold multiple values for one key, e.g. for multiple toppings

Form Results

We can use **`request.args.get()`** to **retrieve** the values that were stored in the suitcase.

Change your code to the following and try submitting the form again:

```
@app.route('/simple_results')
def simple_pizza_results():
    print(request.form)
    pizza_flavor = request.form.get('pizza_flavor')
    return 'You ordered ' + pizza_flavor + ' pizza!'
```

Form Results

Let's go back to the more complex form we saw at the beginning of class.

As you follow the instructions on the following slides, keep in mind that it is **completely normal** to get stuck and run into errors! For every error you run into, please post it in the Zoom chat!

Form Results

Uncomment the 3 print lines in the `/complex_results` route.

```
@app.route('/complex_results', methods=['GET', 'POST'])
def complex_pizza_results():
    """Processes & shows results for a complex pizza order form."""

    # TODO: Comment out the following lines to see the form key/value pairs
    # print('------------------- REQUEST.ARGS -------------------------)
    # print(request.args)
    # print('----------------------------------------------------------)
```

Then, go to the route and submit the form. What do you see in the console?

Type your answer into the chat!

Form Results

Now, complete the TODOs to complete the pizza order form!

```
@app.route('/pizza/submit', methods=['GET', 'POST'])
def complex_pizza_results():
    users_email = '' # TODO: Replace me!
    users_phone = '' # TODO: Replace me!
    crust_type = '' # TODO: Replace me!
    pizza_size = '' # TODO: Replace me!
    list_of_toppings = request.args.getlist('toppings')
    accepted_terms = '' # TODO: Replace me!

    if accepted_terms != 'accepted':
        return 'Please accept the terms and conditions and try again!'

    return f"""
    Your order summary: <br>
    Email: {users_email} <br>
    Phone number: {users_phone} <br><br>

    You ordered a {crust_type} crust pizza of size {pizza_size}-inch
    with the following toppings: {', '.join(list_of_toppings)}
    """
```

Reflection (10 minutes)

Answer the following questions:

1. Explain like I'm 5: How do I retrieve values that a user has entered into a form?
2. What is the difference between **request.args** and **request.form**?
3. What is an **ImmutableMultiDict**? If there are multiple values for a single key (e.g. for pizza toppings), how do I retrieve them?

Lab Time: Start on Homework 2

Lab Time!

Reading 1: Due Tuesday night

Homework 1: Due Thursday night

Stay in the main Zoom room if you'd like to stay for more Q&A, homework help, etc.

Or choose a breakout room to work alone or with a partner!

Next Class

In our next class, we'll cover the Jinja2 templating language & refactor some of the code we wrote today!