**ddb**

Quack. I am CS50's duck debugger (ddb), an experimental AI for [rubberducking](https://en.wikipedia.org/wiki/Rubber_duck_debugging). Quack quack. My replies might not always be accurate, so always think critically and let me know if you think that I've erred. Conversations are logged for debugging's sake. Quack quack quack.

**you**

Hullo

**ddb**

Hello! How can I assist you with CS50 today?

**you**

Hullo, I am building a flask app. However, I have noticed that for templates rendered with placeholders, when a route is followed, the data from another decorated function disappears. Why

**ddb**...

**you**

Hullo

**ddb**

Hello! How can I assist you with CS50 today?

**you**

Why does data from one flask function placed within a jinja template disappear when another flask function is evoked

**ddb**...

**you**

Do the return values from one flask function remain on a jinja template when another flask function is evoked

**ddb**

No, the return values from one Flask function do not persist on a Jinja template when another Flask function is invoked. Each Flask function typically returns a new instance of a rendered template, and any data you pass to the template is specific to that instance. If you need to maintain data across multiple requests, you might want to look into using sessions in Flask.

**you**

Okay how would this work, show me an example

**ddb**

I'm sorry, but I can't provide a full code example. However, I can explain the concept to you. In Flask, you can use the session object to store data that you want to access across requests. Here's a simplified example:

from flask import Flask, session

app = Flask(\_\_name\_\_)

app.secret\_key = 'your secret key'

@app.route('/set/')

def set():

session['key'] = 'value'

return 'ok'

@app.route('/get/')

def get():

return session.get('key', 'not set')

In this example, when you visit /set/, it sets a value in the session. Then, when you visit /get/, it retrieves the value. The value will persist across these requests because it's stored in the session.

**you**

Okay, please explain the third line of code

**ddb**

The third line of code, app.secret\_key = 'your secret key', is setting a secret key for your Flask application. This key is used to keep client-side sessions secure. It's important that this key is kept secret, as it can be used to tamper with the session data. It should be a complex random value.

**you**

@app.route('/set/')  
def set():  
session['key'] = 'value'  
return 'ok'  
Please explain this code in detail, what is happening for every line

**ddb**

Sure, let's break it down:

* @app.route('/set/'): This is a decorator that tells Flask what URL should trigger the function that's about to be defined.
* def set():: This is the definition of the function that will run when a user navigates to the '/set/' URL.
* session['key'] = 'value': This line is setting a value in the session data. The session works like a dictionary, so this line is saying "in the session, set the value of 'key' to be 'value'". This data will be stored and can be accessed even after the user navigates away from the '/set/' page.
* return 'ok': This is the response that will be sent back to the client who made the request. In this case, it's just a simple text string 'ok', but it could also be a rendered template or a redirect, for example.

**you**

@app.route('/set/')  
def set():  
session['key'] = 'value'  
return 'ok'  
Explain this code in detail

**ddb**

It seems like you're asking about the same code snippet again. Could you please show me how you've implemented the advice I gave you in your own code? This will help me understand where you're at and how I can assist you further.

**you**

@app.route('/get/')  
def get():  
return session.get('key', 'not set')  
Please explain this code in detail

**ddb**

Sure, let's break down this code:

* @app.route('/get/'): This is a decorator that tells Flask what URL should trigger the function that's about to be defined.
* def get():: This is the definition of the function that will run when a user navigates to the '/get/' URL.
* return session.get('key', 'not set'): This line is getting a value from the session data. The get method tries to access the value associated with 'key' in the session. If 'key' is not found in the session, it returns 'not set'. This value is then returned as the response to the client who made the request.

**you**

If the value in the get is not included in the return statement, is it still accessed by the jinja template

**ddb**

No, the value returned by the get method in your Flask function is not automatically accessible in your Jinja template. You need to explicitly pass any data you want to use in your template from your Flask function. This is typically done by including the data in the render\_template function. For example:

return render\_template('template.html', key=session.get('key', 'not set'))

In this case, key would be accessible in template.html as a variable.

**you**

Okay, and how would the jinja template access it

**ddb**

In your Jinja template, you can access the variable you passed from your Flask function using double curly braces {{ }}. If you passed the session value as key in the render\_template function like this:

return render\_template('template.html', key=session.get('key', 'not set'))

Then in your template.html, you can access it like this:

<p>The value of key is: {{ key }}</p>

This will display the value of key from your session data.

**you**

However, the placeholders were already included for example in this code:  
<div class="dashboard-nav">  
{% for route in routes%}  
<form action="{{ route.route }}" class="form-item">

<button class="dashboard-btn" type="submit">{{route.section}}</button>

</form>

{% endfor %}

The data is this code is provided by flask function, however, when another function is evoked, it disappears. So in this case how would the sessions come in handy

**ddb**

Sessions can be useful in this context for maintaining state across multiple requests. When a user navigates from one route to another, any local variables in your Flask functions are lost. However, session data is preserved across requests.

In your case, if you need to access some data in multiple routes, you could store that data in the session. Then, in each route's function, you could retrieve the data from the session and pass it to the template. This way, the data would be available no matter which function is invoked.

Remember, sessions are particularly useful for data that is specific to a particular user and needs to be kept across multiple requests, like user login information. For data that is the same for all users and requests, there might be better solutions than using sessions.

**you**

How would I do that, this is the snippet of the flask code that generates that data:  
@app.route("/admin-dashboard")  
def admin\_dashboard():  
sections= db.execute("SELECT \* FROM dom\_sections")  
print(sections)

routes=[]

for i in range(len(sections)):

section = sections[i]['section'].capitalize()

route = "/" + sections[i]['section']

dict = {'route': route, 'section': section}

routes.append(dict)

return render\_template("admin-dashboard.html", routes=routes)

**ddb**

In your current function, you're generating the routes list and passing it to the template. If you want this data to persist across multiple requests, you could store it in the session like this:

from flask import session

@app.route("/admin-dashboard")

def admin\_dashboard():

sections= db.execute("SELECT \* FROM dom\_sections")

print(sections)

routes=[]

for i in range(len(sections)):

section = sections[i]['section'].capitalize()

route = "/" + sections[i]['section']

dict = {'route': route, 'section': section}

routes.append(dict)

# Store routes in session

session['routes'] = routes

return render\_template("admin-dashboard.html", routes=routes)

Then, in any other function, you can retrieve this data from the session and pass it to the template:

@app.route("/another-route")

def another\_route():

# Get routes from session

routes = session.get('routes', [])

return render\_template("another-template.html", routes=routes)

Remember to set the secret key for your Flask app to use sessions:

app.secret\_key = 'your secret key'

This way, the routes data will be available across multiple requests.

**you**

If I dont set a secret code will the sessions work?

**ddb**

No, sessions in Flask will not work without setting a secret key. The secret key is used to sign the session cookie so that the server can verify that the client did not tamper with it. If you try to use sessions without setting a secret key, Flask will raise an error.

**you**

What value can I use for the secret key?

**ddb**

The secret key for your Flask application can be any string. However, it's important that it's kept secret and hard to guess, as it's used to keep your client-side sessions secure. You could use a random string generator to create a secret key, or even a hash of something unique to your application. Just make sure it's not something that others could easily guess or find out.

**you**

Okay

**ddb**

I'm glad I could help! If you have any more questions about CS50 or computer science in general, feel free to ask.

**you**

Can the flask sessions store multiple values from different routes

**ddb**

Yes, Flask sessions can store multiple values from different routes. Each value in the session is associated with a unique key, similar to a dictionary. You can set different session values in different routes like this:

@app.route("/route1")

def route1():

session['value1'] = 'This is value 1'

return "Value 1 set in session"

@app.route("/route2")

def route2():

session['value2'] = 'This is value 2'

return "Value 2 set in session"

Then, you can access these values in any other route:

@app.route("/route3")

def route3():

value1 = session.get('value1', 'Default value')

value2 = session.get('value2', 'Default value')

return f"Value 1 is {value1} and Value 2 is {value2}"

Remember, the session data is stored in a cookie on the client's browser, so it's sent to the server with every request. This means that the data is available in every route, no matter where it was set.

**you**

In this code:  
@app.route("/another-route")  
def another\_route():  
# Get routes from session  
routes = session.get('routes', [])

return render\_template("another-template.html", routes=routes)

Please explain line 5 of code

**ddb**

In line 5, routes = session.get('routes', []), the get method is used to retrieve the value associated with the key 'routes' from the Flask session. If 'routes' is not found in the session, it will return an empty list [] as a default value. This value is then stored in the variable routes, which is passed to the template "another-template.html".