From flask import Flask ------------------------- importing flask and creating new object from Flask class

App = Flask(\_\_name\_\_)

@app.route(“route”) -------------------------------------------------- creating a new route for the app

Def hello\_world():

Return “hello world”

**Export settings:**

FLASK\_APP – containing the name of the server file example: export FLASK\_APP = app.py

FLASK\_ENV – define development/production environment example: FLASK\_ENV = development

DEBUG – enables debug mode

TESTING – enables testing mode

SECRET\_KEY – sign session cookie

SESSION\_COOKIE\_NAME – name of the session cookie

SERVER\_NAME – binds the hosts and port

JSONIFY\_MIMETYPE – defaults to application/json

**To run application:** flask run

Default port: 5000

**Alternative for setting environment variables you can pass in args to the flask exe:**

flask –app <app> --debug run

**return JSON option 1:** return {“message”: “Hello World”} (like a dictionary)

**return JSON option 2 jsonify() method:** import jsonify =>return jsonify(message = “Hello World”)

**Loading application configuration example:** app.config[‘SECRET\_KEY’] = “random-secret-key”

**Configure from an environment variable example:** app.config[“VARIABLE\_NAME”] => app.config.from\_prefixed\_env()

**Configure from a python file example:** app.config.from\_file(“pathtoconfigfile”)

**in big apps you rather store files in separates folders, for example:**

תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

התיאור נוצר באופן אוטומטי

**Custom routes:**

@app.route(“/path”) – defaults to the GET method

@app.route(“/health”)

**The following two methods are the same:**

Def health():

Return jsonify(dict(status=”OK”)), 200

@app.route(“/health”, methods=[“GET”])

Return jsonify(dict(status=”OK”)), 200

**Another option:**

@app.route(“/health”, methods=[“GET”, “POST”])

Def health():

If request.method == “GET”: return jsonify(status=”OK”, method=”GET”), 200

If request.method == “POST”: return jsonify(status=”OK”, method=”POST”), 200

**Request object - Attributes:**

* All HTTP calls to Flask contain the request object created from the Flask.Request class.
* Some common request attributes are:
  + Server – the address of the server as a tuple (host, port)
  + Headers – header of the request
    - Cache-Control – holds information on how to cache in browsers
    - Accept – informs the browser what kind of content type is understood by the client
    - Accept-encoding – indicates the code content
    - User-agent – identifies the client, application, OS or version
    - Accept-language – requests for a specific language and locale
    - Host – specifies the host and port number of the requested server
  + URL - URL that is the resource asked by request
  + Access\_route – list all the IP addresses for requests that are forwarded multiple times.
  + Full\_path – represents the complete path of the request, including any query string
  + Is\_secure – True if a client makes a request using HTTPS or WSS protocols
  + Is\_JSON – True if the request contains JSON data
  + Cookies – dict that contains cookies that are in the request

**Example for a message:**

****

**Request object - methods:**

* Get\_data – get POST data from the request as bytes (you are responsible to parse the data)
* Get\_JSON – parses POST data from the request as JSON

**Request object – parse request data:**

There are multiple ways to get parsed data depending on what is coming in with a request

* Args: MultiDict[str, str] – query parameters as a dict
* JSON: optional[any] – parse data into dict
* Files: immutableMultiDict[str, FileStorage] – provide users uploaded files
* Form: immutableMultiDict[str, str] – contains all values posted in a form submission
* Values: combinedMultiDict[str, str] – combine the args data with the form data

**Request object – access values:**

you can get values from MultiDict, immutableMultiDict, and CombinedMultiDict as you would from a python dict using indexing or the GET method.

**Extracting example using args:**

**תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**

**Response object – attributes:**

* When a client calls a URL, it expects a response back. Flask has an in-built response class you can leverage.

**Common response attributes:**

* Status code – indicates the success or failure of the request
* Headers – gives more info about the response
* Content\_Type – shows the media type of the requested resource
* Content\_length – size of the response message body
* Content\_encoding – indicates any incoding applied to the response, so the client know how to decode the data
* Mime-type sets the media type of the response
* Expires – contains the data or time after which the response is considered expired

**Common methods of response object:**

* Set\_cookie – sets a browser cookie on the client.
* Delete\_cookie – deletes a cookie on the client.

**Response object – usage:**

* Success response from @app.route method (status code, html mime-type)
* JSONify method – creates response object automatically
* Make\_response method – create a custom response
* Redirect response – return a 302 status-code and redirect the client to another URL
* Abort method – return a response with an error condition

**תמונה שמכילה טקסט, צילום מסך, גופן, דף אינטרנט

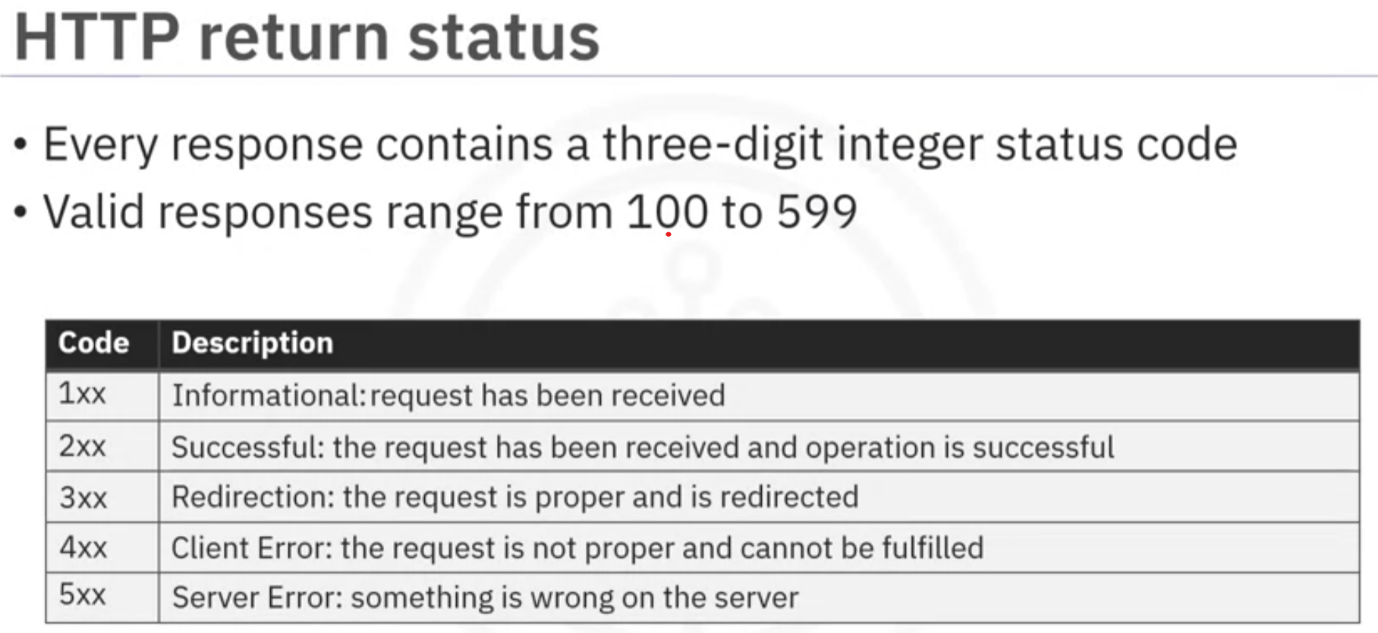
התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, גופן, מספר

התיאור נוצר באופן אוטומטי**

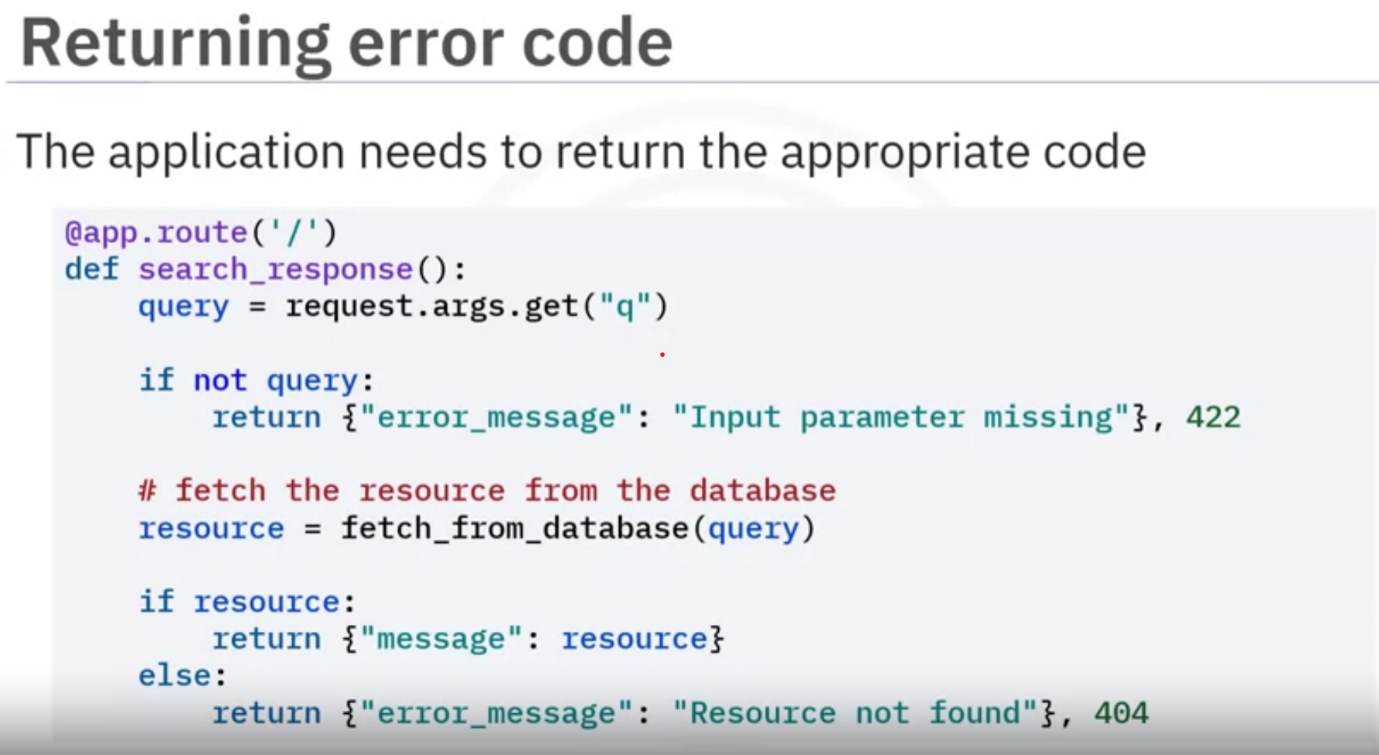
****

**תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**

**תמונה שמכילה טקסט, צילום מסך, גופן, מספר

התיאור נוצר באופן אוטומטי**

****

**תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**

**Accessing Form Data with flask.request.form**

You can use flask.request.form to access form data that a user has submitted via a POST request. For instance, this feature can be used if you have a login form with username and password fields.

In your HTML file, you might have a form like this:

1. 1
2. 2
3. 3
4. 4
5. 5
6. <form method="POST" action="/login">
7. <input type="text" name="username">
8. <input type="password" name="password">
9. <input type="submit" value="Submit">
10. </form>

Copied!

The Python code to access the username and password will be as follows:

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. from flask import request
9. @app.route('/login', methods=['POST'])
10. def login():
11. username = request.form['username']
12. password = request.form['password']
13. # process login here

Copied!

**Redirecting to a URL with flask.redirect**

Flask provides a function called flask.redirect to guide users to a different webpages (or endpoints). The flask.redirect function can be useful in several scenarios. For example, you can use the flask.redirect function to redirect a user to a **login** page when they try to access a restricted **admin** page.

Python code:

1. 1
2. 2
3. 3
4. 4
5. 5
6. from flask import redirect
7. @app.route('/admin')
8. def admin():
9. return redirect('/login')

Copied!

**Generating Dynamic URLs with flask.url\_for**

The flask.url\_for function dynamically generates URLs for a given endpoint. Dynamically generating URLs can be particularly useful when the URL for a route is altered. The flask.url\_for function automatically updates the URL throughout your templates or code, minimizing manual work. For example, consider the scenario where a user is trying to access the **admin** page and must be redirected to the **login** page. In this scenario, url\_for('login') will retrieve the URL for the **login** page from the existing routes.

Python code:

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. from flask import url\_for
11. @app.route('/admin')
12. def admin():
13. return redirect(url\_for('login'))
14. @app.route('/login')
15. def login():
16. return "<Login Page>"

Copied!

**Handling different HTTP request types**

Flask allows you to define routes to manage different types of HTTP requests. You can define the route with both the access methods, GET and POST, and in the function description, define the use cases for both methods.

Python code:

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. @app.route('/data', methods=['GET', 'POST'])
8. def data():
9. if request.method == 'POST':
10. # process POST request
11. if request.method == 'GET':
12. # process GET request

Copied!

In the HTML file, you will add a form that allows both GET and POST requests:

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. <!-- For POST -->
10. <form method="POST" action="/data">
11. <!-- Your input fields here -->
12. <input type="submit" value="Submit">
13. </form>
14. <!-- For GET -->
15. <a href="/data">Fetch data</a>

Copied!

In the last example, the /data route accepts both GET and POST requests. The type of the request can be checked using flask.request.method.

**CRUD operations**

The CRUD operations represent the four basic functions that you require to interact with any persistent storage, such as a database. In web development, the CRUD operations often correspond to HTTP methods.

**Create operation**

Creating data often involves presenting a form to the user to gather the information that you want to store in the database as a new record. In Flask, this data is accessed using flask.request.form.

HTML form for creating data:

1. 1
2. 2
3. 3
4. 4
5. <form method="POST" action="/create">
6. <input type="text" name="name">
7. <input type="submit" value="Create">
8. </form>

Copied!

Python code:

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. @app.route('/create', methods=['GET', 'POST'])
13. def create():
14. if request.method == 'POST':
15. # Access form data
16. name = request.form['name']
17. # Create a new record with the name
18. record = create\_new\_record(name) # Assuming you have this function defined
19. # Redirect user to the new record
20. return redirect(url\_for('read', id=record.id))
21. # Render the form for GET request
22. return render\_template('create.html')

Copied!

**Read operation**

Reading data involves accessing the data and presenting it to the user. To access specific entries, the request needs to go with specific IDs. Therefore, you will need to pass the ID as an argument to the function. The following example shows that the ID can be accessed from the route.

Python code:

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. @app.route('/read/<int:id>', methods=['GET'])
8. def read(id):
9. # Get the record by id
10. record = get\_record(id) # Assuming you have this function defined
11. # Render a template with the record
12. return render\_template('read.html', record=record)

Copied!

**Update operation**

Updating data requires the process of accessing specific entries, like the **Read** operation, and involves giving new data to the concerned parameter, like the **Create** operation. Therefore, the route should access the ID and contain both access methods.

Sample HTML form for updating data:

1. 1
2. 2
3. 3
4. 4
5. <form method="POST" action="/update/{{record.id}}">
6. <input type="text" name="name" value="{{record.name}}">
7. <input type="submit" value="Update">
8. </form>

Copied!

Python code:

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. @app.route('/update/<int:id>', methods=['GET', 'POST'])
15. def update(id):
16. if request.method == 'POST':
17. # Access form data
18. name = request.form['name']
19. # Update the record with the new name
20. update\_record(id, name) # Assuming you have this function defined
21. # Redirect user to the updated record
22. return redirect(url\_for('read', id=id))
24. # Render the form for GET request with current data
25. record = get\_record(id) # Assuming you have this function defined
26. return render\_template('update.html', record=record)

Copied!

**Delete operation**

Deleting data involves removing a record based on its ID. The Delete operation will typically require the ID to be passed, as reported by the HTML page, in the form of an argument to the function.

Sample HTML form for deleting data:

1. 1
2. 2
3. 3
4. <form method="POST" action="/delete/{{record.id}}">
5. <input type="submit" value="Delete">
6. </form>

Copied!

Python code:

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. @app.route('/delete/<int:id>', methods=['POST'])
8. def delete(id):
9. # Delete the record
10. delete\_record(id) # Assuming you have this function defined
11. # Redirect user to the homepage
12. return redirect(url\_for('home'))