Topic	Flask API	
Class Description	**Students curate all the data they have in a list of dictionaries and create a Flask API for it.**	
Class	**C136**	
Class time	**45 mins**	
Goal	• Curate all the data • Create flask API for getting data for all the planets or an individual planet	

Resources Required	• Teacher Resources
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	• Student Resources
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen	

Class structure	**Warm Up** **Teacher-led Activity** **Student-led Activity** **Wrap up**	**5 mins** **15 min** **15 min** **5 min**

CONTEXT

- **Review the concepts learned in the earlier classes**

Class Steps	Teacher Action	Student Action
Step 1: Warm Up (5 mins)	Hi <Student Name> We have finally validated our output, and we know that we have the right output. We debugged our code in the last class! Can you tell me what is used for error handling in Python?	**ESR:** - Try Except Statements!

	That's right! Now in this class, we will curate all the data into a single list of dictionaries and create a couple of APIs using Flask! Are you excited?	**ESR:** "Yes!"
	Let's start!	
Teacher Initiates Screen Share		
CHALLENGE • **Curate all the data into a list of dictionary**		
Step 2: Teacher-led Activity (15 min)	*(Before beginning the class, make sure to use the same colab that you used in the last class. This is the continuation of that.)*	
	Let's start by thinking about it! We have analyzed for 4 specifications of a planet. What are they?	**ESR:** ~ Gravity ~ Planet Type ~ Goldilock Zone ~ Speed
	Okay! Now that we know this, we already have planet type in the CSV we started with, and we also know the orbital radius, which means that we know if the planet would be in the Goldilock Zone or not! Apart from that, we calculated the Gravity and the speed of the planet! Hence, it only makes sense to add	

these into our data and display this in the mobile app as well!

To achieve this, we will have to revisit the code where we were creating the dictionary of all the planets with their specifications!

Here, after calculating Gravity, we want to append the gravity into the **planet_data** and after calculating the Speed, we also want to append speed into it!

Let's do that!

```
try:
   if gravity < 100:

features_list.append("gravity")
   planet_data.append(gravity)
 except:
planet_data.append("Unknown")
```

```
try:
   try:
      distance = 2 * 3.14 *
(float(planet_data[8].split("
")[0]) * 1.496e+9)
   except:
      try:
        distance = 2 * 3.14 *
(float(planet_data[8]) *
1.496e+9)
      except: pass
   try:
```

```
      time, unit =
planet_data[9].split(" ")[0],
planet_data[9].split(" ")[1]
      if unit.lower() == "days":
        time = float(time)
      else:
        time = float(time) * 365
    except:
      time = planet_data[9]
    time = time * 86400
    speed = distance / time
    if speed < 200:

features_list.append("speed")
    planet_data.append(speed)
  except:
planet_data.append("Unknown")
```

Here, let's make a note that we handled the unknown values as well inside the Except clause. This means that if for some reason we were not able to calculate the speed or gravity of a planet, we are still having **Unknown** value there to have consistency in data.

Now here, this code is exactly the same as the earlier one. The only changes we made is that we have 2 extra lines.

```
planet_data.append(gravity)
```

```
planet_data.append(speed)
```

These are to add the gravity and speed to our planet data!

We are also adding

```
planet_data.append("Unknown")
```

in the Except clause so that it is well handled and we have consistency!

Make these changes and re-run the cell.

```
gravity = (float(planet_data[3])*5.972e+24) / (float(planet_data[7])*float(planet_data[7])*6371000*6371000) * 6.674e-11
try:
  if gravity < 100:
    features_list.append("gravity")
  planet_data.append(gravity)
except: planet_data.append("Unknown")
```

```
try:
  try:
    distance = 2 * 3.14 * (float(planet_data[8].split(" ")[0]) * 1.496e+9)
  except:
    try:
      distance = 2 * 3.14 * (float(planet_data[8]) * 1.496e+9)
    except: pass
  try:
    time, unit = planet_data[9].split(" ")[0], planet_data[9].split(" ")[1]
    if unit.lower() == "days":
      time = float(time)
    else:
      time = float(time) * 365
  except:
    time = planet_data[9]
  time = time * 86400
  speed = distance / time
  if speed < 200:
    features_list.append("speed")
  planet_data.append(speed)
except: planet_data.append("Unknown")
```

	Here, if we notice, we have not added the append() function in the if statement. What would have gone wrong if we did?	**ESR:** There might be planets whose gravity or speed got calculated so their values will not be **Unknown** but if they did not satisfy the if condition, their values would not be added.

	Awesome! Now, let's create a list of dictionaries containing specific data points for all the planets.
	
	We want it to include:
	• **name**
	• **distance from earth**
	• **planet mass**
	• **planet radius**
	• **planet type**
	• **distance from their sun (orbital radius)**
	• **orbital period**
	• **gravity**
	• **orbital speed**
	• **specifications/features** that we have in our **final_dict**
	
	Let's code it!
	
	`final_planet_list = []`
	
	`for planet_data in planet_data_rows:`
	`temp_dict = {`
	`"name": planet_data[1],`	

```
"distance_from_earth":
planet_data[2],
                    "planet_mass":
planet_data[3],
                    "planet_type":
planet_data[6],

"planet_radius": planet_data[7],

"distance_from_their_sun":
planet_data[8],

"orbital_period":
planet_data[9],
                    "gravity":
planet_data[20],

"orbital_speed": planet_data[21]
                }
 temp_dict["specifications"] =
final_dict[planet_data[1]]

final_planet_list.append(temp_di
ct)

print(final_planet_list)
```

Here, let's go over this code once. First, we are creating an empty list to store all the dictionaries of the planets.

Then, we are iterating over all the planet_data_rows that we had.

We are then creating a temp_dict which is a dictionary, and we are mapping all the values we need in key value pairs inside this dictionary.

Finally, we are adding another key value pair for specifications and for this, we are looking for the value in **final_dict** with key as the name of the planet (or you could use the first element, which is the row num. This depends on what key you used while creating the final_dict).

We are then appending this dictionary into the empty list we created and then finally we are printing the list outside the for loop.

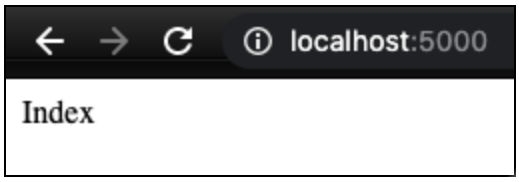
```
[ ]  final_planet_list = []

     for planet_data in planet_data_rows:
       temp_dict = {
                       "name": planet_data[1],
                       "distance_from_earth": planet_data[2],
                       "planet_mass": planet_data[3],
                       "planet_type": planet_data[6],
                       "planet_radius": planet_data[7],
                       "distance_from_their_sun": planet_data[8],
                       "orbital_period": planet_data[9],
                       "gravity": planet_data[20],
                       "orbital_speed": planet_data[21]
                   }
       temp_dict["specifications"] = final_dict[planet_data[1]]
       final_planet_list.append(temp_dict)

     print(final_planet_list)
```

```
[{'name': '11 Comae Berenices b', 'distance_from_earth': '305.0',
```

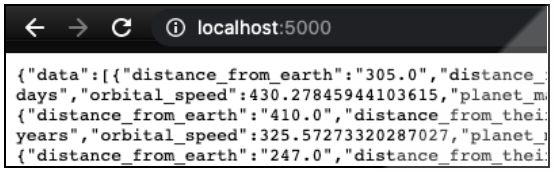
	Here, we get the list of dictionaries!	
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
• **Ask Student to press ESC key to come back to panel** • **Guide Student to start Screen Share** • **Teacher gets into Fullscreen**		
ACTIVITY • **Student writes the FLASK API** • **Student tests the API**		
Step 3: Student-Led Activity (15 min)	Okay! Let's start with the basics! Create a virtual environment and activate it. **python3.8 -m venv venv** **MacOS or Ubuntu** **source venv/bin/activate** **Windows** **.venv\Stripts\activate**	*Students create the virtual environment.*
	Next, install flask. **pip install flask**	*Students install Flask*

	Now, let's create a file **data.py**. Inside this file, we want to create a variable known as **data** and we will copy paste the list of dictionaries that we printed in the colab here.	*Students create the file, copies the printed data from colab and pastes it to assign to the variable named as **data**.*

	Now, let's create a file called **main.py** which will be our main server file.
	
	Here, we will first import Flask.
	
	`from flask import Flask`
	
	We will then define the app variable.
	
	`app = Flask(__name__)`
	
	We will create an index/home function for our view. This route */* is what the user will open on the browser. The return statement here means that we want to return the word **Index** to the browser so as to display it.
	
	`@app.route("/")`
	`def index():`
	`  return "Index"`
	
	And we will finally run the app.
	
	`if __name__ == "__main__":`
	`  app.run()`
	
	Try running this file from terminal.
	
	python main.py
		The student writes the basic template for a Flask App.

	Go to the **localhost:5000** in your browser and you should see the index written there!  Great!	

	Now, we want to display the list of all the planets in our home page of the mobile app. For that, we need to make sure that our home/index function returns all the data that we stored in our **data.py** file.
	
	Let's quickly do that. Remember, we need to return the data as a json file. Python has a function **jsonify** that helps us with that!
	
	Change the initial import line and import jsonify as well.
	`from flask import Flask, jsonify`
	
	Next, import data to your main.py.
	`from data import data`
	
	Finally, change the home function.
	`@app.route("/")`
	`def index():`
	`  return jsonify({`	

	<pre>"data": data,
	"message": "success"
	}), 200</pre>
	Re-run the server and check on your localhost:5000 what you get.
	 	

	Great! Now our first API is ready, that returns that data for all the planets!
	
	We need one more API, which returns the data for only one planet at a time!
	
	Let's quickly build that. For this, we need to have a unique identifier with which we can tell what planet's data is the user requesting.
	
	We will use the name of the planet for the same. The user can provide the name of the planet as a URL parameter and we can send them the planet's data based on that.
	
	For that, we need to import **request** from flask as well.
	
	<pre>from flask import Flask, jsonify,
	request</pre>
	Now, we will create another route function which will take the name of
		The student creates the second API.

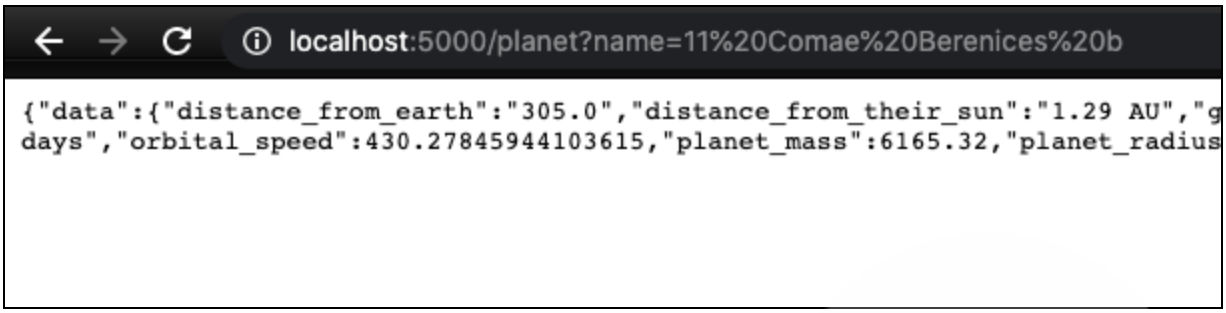
the planet from the URL argument and then find its data in the dictionary and then return the data.

```
@app.route("/planet")
def planet():
   name = request.args.get("name")
   planet_data = next(item for item
in data if item["name"] == name)
   return jsonify({
       "data": planet_data,
       "message": "success"
   }), 200
```

Here, we are first fetching the name of the planet from the URL argument. We are then using the **next()** function to find a dictionary that satisfies the condition, which is, the value of name should match with the name we are providing!

We are then finally returning only the planet's data this time.

Re-run the server and goto the index route, copy the name of any planet and try to get the data for that planet using this API.



	Great! Our API is working!	
Teacher Guides Student to Stop Screen Share		
FEEDBACK • **Appreciate the student for their efforts** • **Identify 2 strengths and 1 area of progress for the student**		
Step 4: Wrap-Up (5 min)	We know how to use NGROK as well. We are now, all set to create the mobile app based on all the data we have curated! How was your experience?	**ESR:** varied
	Amazing. While working on this project, we also made sure that we are at the top of all the concepts we have acquired so far like how to build APIs and segregate data, etc. Next class, we will be building the mobile app!	-
Teacher Clicks ✖ End Class		

Activity	Activity Name	Links
Teacher Activity 1	Solution Colab	https://colab.research.google.com/gist/shubhamwhj/99aff4165b9f4eac7ae83a964914d11e/pro-136-reference-code.ipynb
Teacher Activity 2	Solution Flask API	https://github.com/whitehatjr/planet_flask