WhiteHat Jr
Live online Coding for Kids

Topic	FLASK API	
Class Description	The student will curate all the data they have in a list of dictionaries and create a Flask API for it.	
Class	PRO 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 ○ Smartphone ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	Warm-Up Teacher-Led Activity 1 Student-Led Activity 1 Wrap-Up	5 mins 15 mins 20 mins 05 mins
Credit & Permissions:	Flask by pallets projects under BSD license Exoplanet Exploration by NASA	
WARM-UP SESSION - 10 mins		
Teacher Starts Slideshow **Slide # to #** <**Note**: Only Applicable for Classes with VA>		

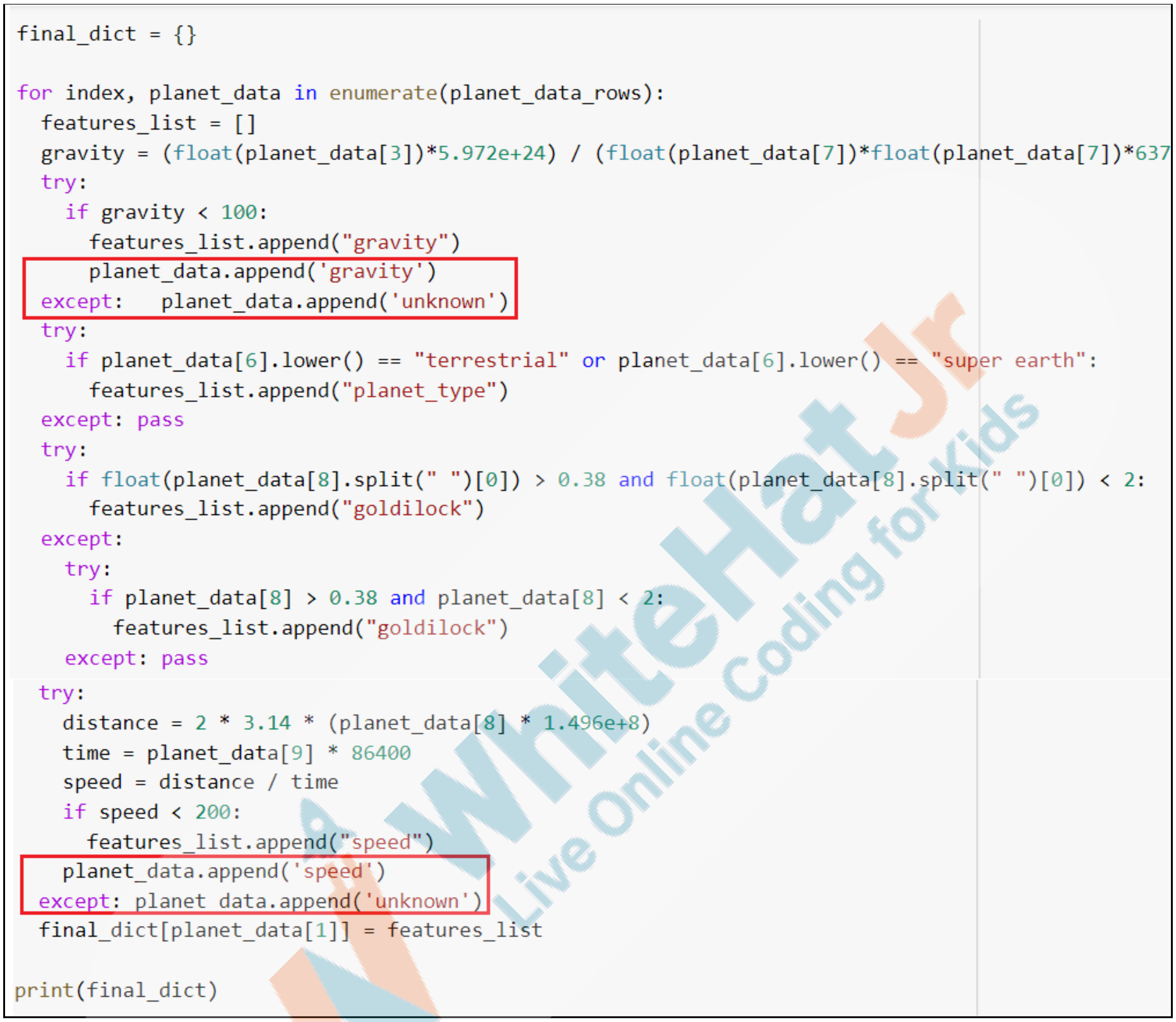
<table>
<tr><td colspan="2">Refer to speaker notes and follow the instructions on each slide.</td></tr>
<tr><th>Teacher Action</th><th>Student Action</th></tr>
<tr><td>Hey &lt;student's name>. How are you? It’s great to see you!
Are you excited to learn something new today?

<b>Following are the WARM-UP session deliverables:</b>
● Greet the student.
● Revision of previous class activities.
● Quizzes.</td><td><b>ESR</b>: Hi, thanks!
Yes, I am excited about it!

Click on the slide show tab and present the slides</td></tr>
<tr><td colspan="2"><b>WARM-UP QUIZ</b>
Click on In-Class Quiz</td></tr>
<tr><td colspan="2"><b>Continue WARM-UP Session</b>
<b>Slide # to #</b>
&lt;<b>Note</b>: Only Applicable for Classes with VA&gt;</td></tr>
<tr><td colspan="2"><b>Activity Details</b>

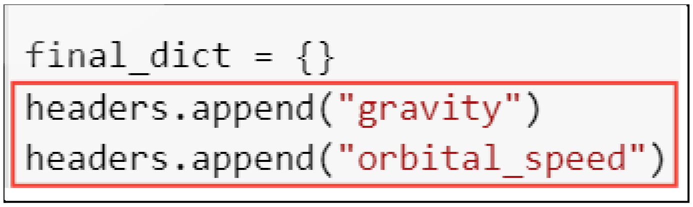
<b>Following are the session deliverables:</b>
● Appreciate the student.
● Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.</td></tr>
<tr><th>Teacher Action</th><th>Student Action</th></tr>
<tr><td>In the last class we have finally validated our output, and we know that we have the right output. We debugged our code.
Can you tell me which block we used to handle errors?</td><td><b>ESR:</b>
- Try Except Statements!</td></tr>
<tr><td colspan="2"><b>Teacher Ends Slideshow</b></td></tr>
</table>

TEACHER-LED ACTIVITY - 10 mins	
Teacher Initiates Screen Share	
● **Curate all the data into a list of dictionary**	
Teacher Action	**Student Action**
Note: Open Teacher Activity 1 for boilerplate code. It is the same as the previous class code. Let's start by thinking about it! We have analyzed 4 specifications of a planet. What are they? Okay! Now that we know this, we already have planet type in the CSV. 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 to 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 list **planet_data** and after calculating the speed, we also want to append speed into it! Let's do that!	**ESR:** ~ Gravity ~ Planet Type ~ Goldilock Zone ~ Speed



```
final_dict = {}

for index, planet_data in enumerate(planet_data_rows):
  features_list = []
  gravity = (float(planet_data[3])*5.972e+24) / (float(planet_data[7])*float(planet_data[7])*637
  try:
    if gravity < 100:
      features_list.append("gravity")
      planet_data.append('gravity')
  except:   planet_data.append('unknown')
  try:
    if planet_data[6].lower() == "terrestrial" or planet_data[6].lower() == "super earth":
      features_list.append("planet_type")
  except: pass
  try:
    if float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:
      features_list.append("goldilock")
  except:
    try:
      if planet_data[8] > 0.38 and planet_data[8] < 2:
        features_list.append("goldilock")
    except: pass
  try:
    distance = 2 * 3.14 * (planet_data[8] * 1.496e+8)
    time = planet_data[9] * 86400
    speed = distance / time
    if speed < 200:
      features_list.append("speed")
    planet_data.append('speed')
  except: planet_data.append('unknown')
  final_dict[planet_data[1]] = features_list

print(final_dict)
```

Also, add the respective headers.



```
final_dict = {}
headers.append("gravity")
headers.append("orbital_speed")
```

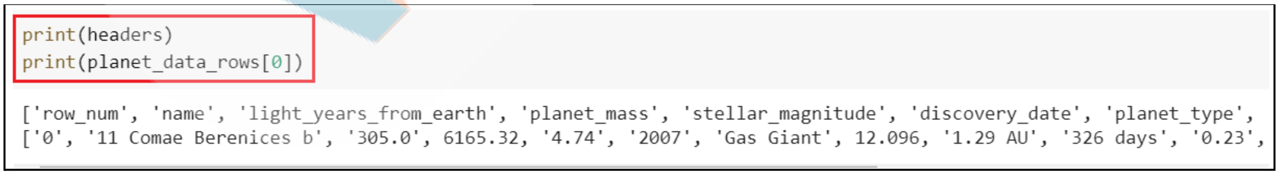
Here, let’s make a note that we handled the unknown values as well inside the **Except** clause. This means that if for some reason we were not able to calculate the speed or gravity of a planet, we are still having **unknown** value there to have consistency in data.

Now here, this code is the same as the earlier one. The only change we made is that we have 2 extra lines. These are to add gravity and speed to our planet’s data!

We are also adding **‘unknown’** to the **planet_data** in the Except clause so that it is well handled and we have consistency!

Make these changes and re-run the cell.
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.



```
print(headers)
print(planet_data_rows[0])
```

```
['row_num', 'name', 'light_years_from_earth', 'planet_mass', 'stellar_magnitude', 'discovery_date', 'planet_type',
['0', '11 Comae Berenices b', '305.0', 6165.32, '4.74', '2007', 'Gas Giant', 12.096, '1.29 AU', '326 days', '0.23',
```

Awesome! Now, let’s create a list of dictionaries containing specific data points for all the planets.

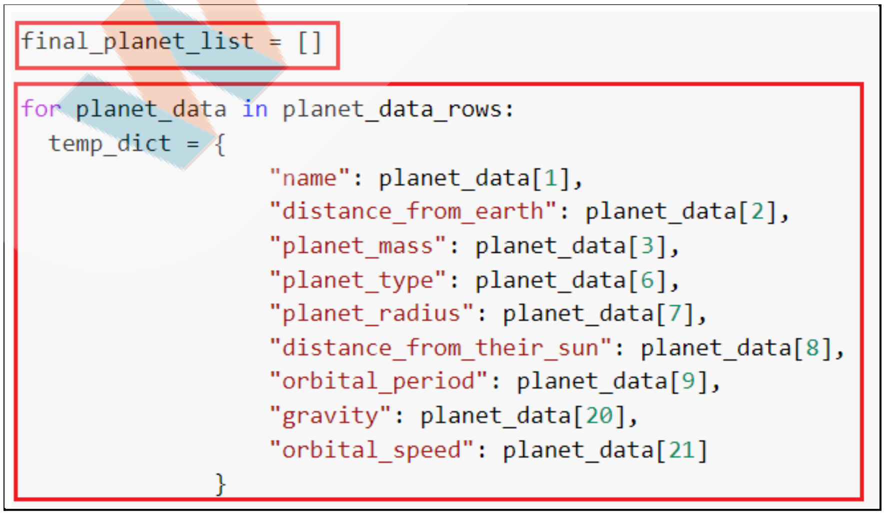
We want to include:

- **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 dictionary called **final_dict**

Let's code it!
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.

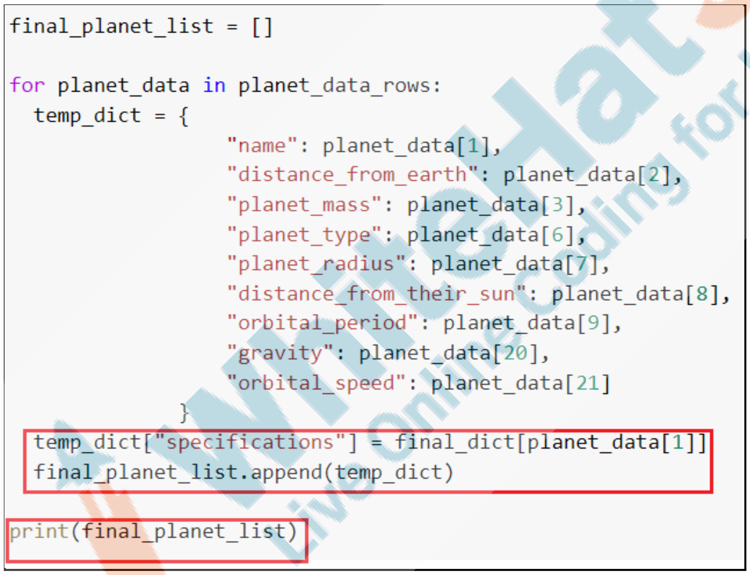
We are then creating a **temp_dict** which is a dictionary, and we are mapping all the values we need in key-value pairs inside this dictionary.



```
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]
              }
```

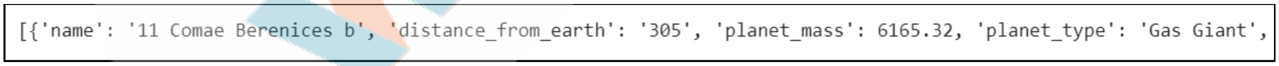
Finally, we are adding another key-value pair for specifications and for this, we are looking for the value in **final_dict** with key as the name of the planet (or you could use the first element, which is the row num. This depends on what key you used while creating the **final_dict**).

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)
```

Here, we get the list of dictionaries!



```
[{'name': '11 Comae Berenices b', 'distance_from_earth': '305', 'planet_mass': 6165.32, 'planet_type': 'Gas Giant',
```

So, we have the data in a presentable format. It is ready to be displayed. Thus, we'll be using Flask to display the data on the webpage. All we have to do is create a Python file called **data.py** in VS Code. We'll create a variable called data and store the **final_planet_list** in it.

Teacher Stops Screen Share

So now it's your turn.

Please share your screen with me.	
Teacher Starts Slideshow **Slide # to #** <**Note**: Only Applicable for Classes with VA> Refer to speaker notes and follow the instructions on each slide.	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
Teacher Ends Slideshow	
STUDENT-LED ACTIVITY - 20 mins	
Ask the student to press the ESC key to come back to the panel. **Guide the student to start Screen Share.** **The teacher gets into Full Screen.**	
Student Initiates Screen Share	
ACTIVITY **Student writes the FLASK API** **Student tests the API**	
Teacher Action	**Student Action**
Note: *Guide the student to open Student Activity 1 for the final colab file. The data file is given in Student Activity 2.* Okay! Let's start with the basics! Create a virtual environment on command prompt and activate it.	

python3.8 -m venv venv

MacOS or Ubuntu
source venv/bin/activate

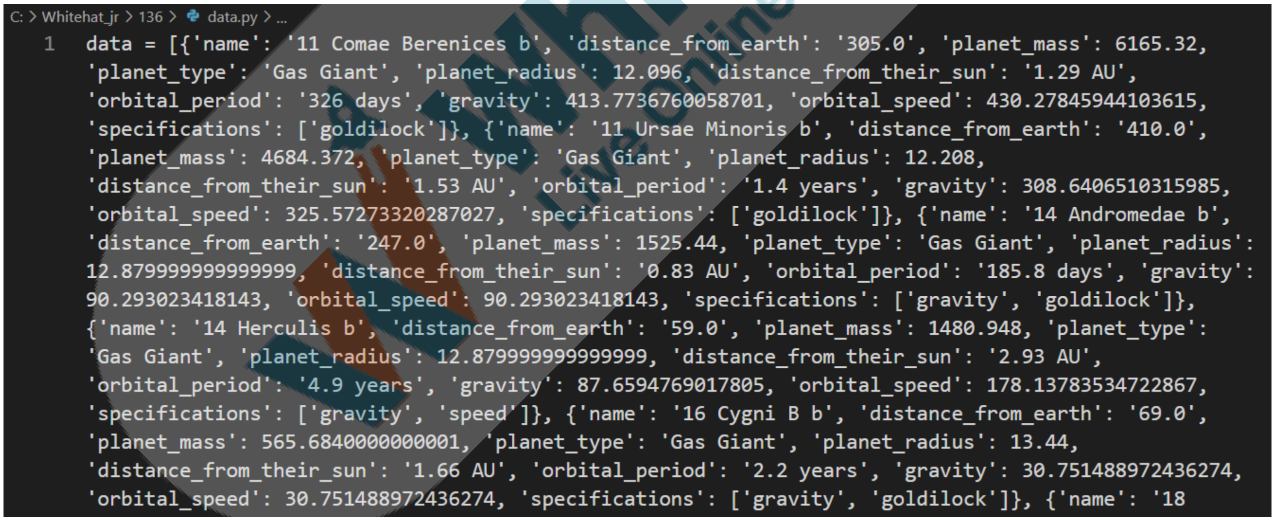
Windows
.venv\Stripts\activate

Next, install flask.

pip install flask

Now, let’s open the 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.

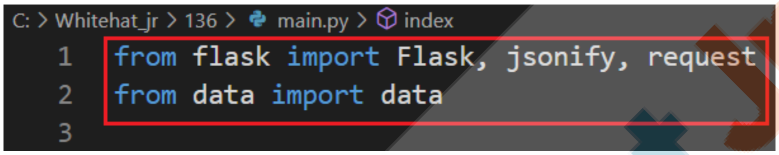


```
data = [{'name': '11 Comae Berenices b', 'distance_from_earth': '305.0', 'planet_mass': 6165.32,
'planet_type': 'Gas Giant', 'planet_radius': 12.096, 'distance_from_their_sun': '1.29 AU',
'orbital_period': '326 days', 'gravity': 413.7736760058701, 'orbital_speed': 430.27845944103615,
'specifications': ['goldilock']}, {'name': '11 Ursae Minoris b', 'distance_from_earth': '410.0',
'planet_mass': 4684.372, 'planet_type': 'Gas Giant', 'planet_radius': 12.208,
'distance_from_their_sun': '1.53 AU', 'orbital_period': '1.4 years', 'gravity': 308.6406510315985,
'orbital_speed': 325.57273320287027, 'specifications': ['goldilock']}, {'name': '14 Andromedae b',
'distance_from_earth': '247.0', 'planet_mass': 1525.44, 'planet_type': 'Gas Giant', 'planet_radius':
12.879999999999999, 'distance_from_their_sun': '0.83 AU', 'orbital_period': '185.8 days', 'gravity':
90.293023418143, 'orbital_speed': 90.293023418143, 'specifications': ['gravity', 'goldilock']},
{'name': '14 Herculis b', 'distance_from_earth': '59.0', 'planet_mass': 1480.948, 'planet_type':
'Gas Giant', 'planet_radius': 12.879999999999999, 'distance_from_their_sun': '2.93 AU',
'orbital_period': '4.9 years', 'gravity': 87.6594769017805, 'orbital_speed': 178.13783534722867,
'specifications': ['gravity', 'speed']}, {'name': '16 Cygni B b', 'distance_from_earth': '69.0',
'planet_mass': 565.6840000000001, 'planet_type': 'Gas Giant', 'planet_radius': 13.44,
'distance_from_their_sun': '1.66 AU', 'orbital_period': '2.2 years', 'gravity': 30.751488972436274,
'orbital_speed': 30.751488972436274, 'specifications': ['gravity', 'goldilock']}, {'name': '18
```

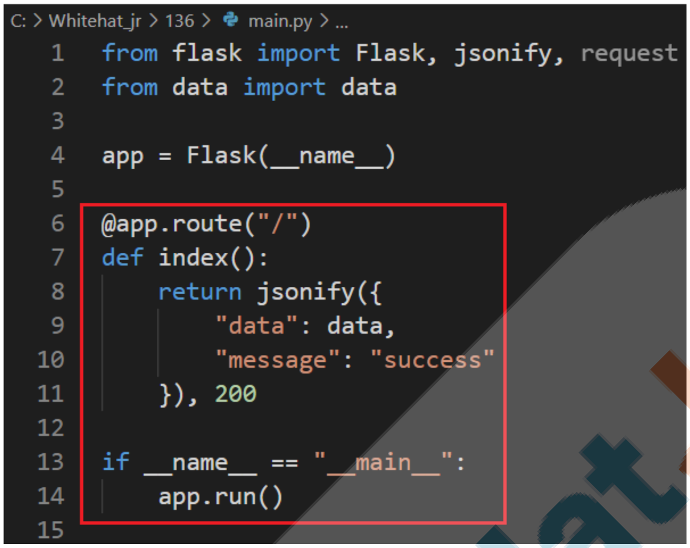
Now, let’s create a file called **main.py** which will be our main server file.

1. Import **flask**. Also we need to send the data to the browser.

Do you remember the method used to send the data to the browser from Python file in **JASON** format? 2. So, we'll import **request** and **jasonify** libraries as well. 3. Now, we have data in **data.py**. This file has variable data that stores the list. Thus, import this data from **data.py**.	**ESR:** The **jasonify** method


 	
`from flask import Flask, jsonify, request`
`from data import data`	

What would be the next step for creating an API? Great!! We will then define the **app** variable. 1. Create an **index/home** function. This **@app.route** specifies the **end point** that the user will open on the browser. 2. The **return** statement here means that we want to return the data to the browser to display it. 3. Use the **jasonify()** function since the data is sent in json format, Inside this send data and message to the browser. 4. Outside the function write **200** for the successful execution of the code. 5. Write **app.run()** function to run the app.	**ESR:** To create a variable called app. **app=** Flask(__name__)

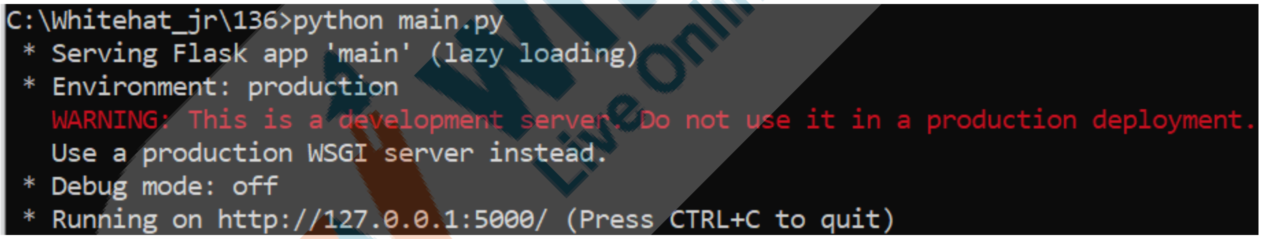


C: > Whitehat_jr > 136 > main.py > ...

```python
from flask import Flask, jsonify, request
from data import data

app = Flask(__name__)

@app.route("/")
def index():
    return jsonify({
        "data": data,
        "message": "success"
    }), 200

if __name__ == "__main__":
    app.run()

```

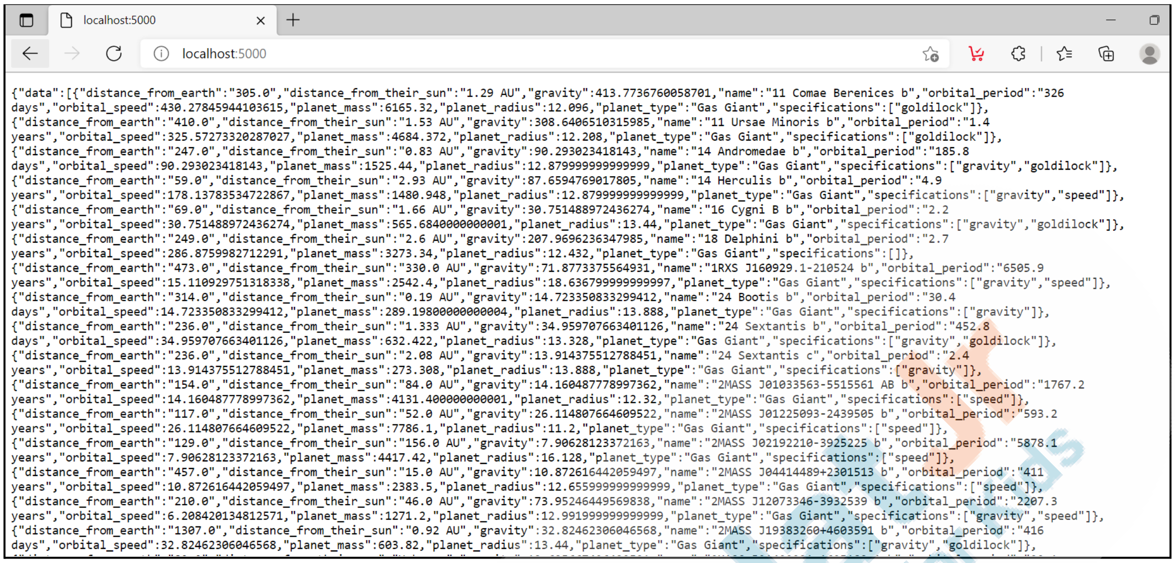
Try running this file in command prompt using the command:

python main.py



```
C:\Whitehat_jr\136>python main.py
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Go to the **localhost:5000** in your browser and you should see the index written there!(else copy paste the URL as shown on command prompt)



```
{"data":[{"distance_from_earth":"305.0","distance_from_their_sun":"1.29 AU","gravity":413.7736760058701,"name":"11 Comae Berenices b","orbital_period":"326
days","orbital_speed":430.27845944103615,"planet_mass":6165.32,"planet_radius":12.096,"planet_type":"Gas Giant","specifications":["goldilock"]},
{"distance_from_earth":"410.0","distance_from_their_sun":"1.53 AU","gravity":308.6406510315985,"name":"11 Ursae Minoris b","orbital_period":"1.4
years","orbital_speed":325.57273320287027,"planet_mass":4684.372,"planet_radius":12.208,"planet_type":"Gas Giant","specifications":["goldilock"]},
{"distance_from_earth":"247.0","distance_from_their_sun":"0.83 AU","gravity":90.293023418143,"name":"14 Andromedae b","orbital_period":"185.8
days","orbital_speed":90.293023418143,"planet_mass":1525.44,"planet_radius":12.879999999999999,"planet_type":"Gas Giant","specifications":["gravity","goldilock"]},
{"distance_from_earth":"59.0","distance_from_their_sun":"2.93 AU","gravity":87.6594769017805,"name":"14 Herculis b","orbital_period":"4.9
years","orbital_speed":178.13783534722867,"planet_mass":1480.948,"planet_radius":12.879999999999999,"planet_type":"Gas Giant","specifications":["gravity","speed"]},
{"distance_from_earth":"69.0","distance_from_their_sun":"1.66 AU","gravity":30.751488972436274,"name":"16 Cygni B b","orbital_period":"2.2
years","orbital_speed":30.751488972436274,"planet_mass":565.6840000000001,"planet_radius":13.44,"planet_type":"Gas Giant","specifications":["gravity","goldilock"]},
{"distance_from_earth":"249.0","distance_from_their_sun":"2.6 AU","gravity":207.9696236347985,"name":"18 Delphini b","orbital_period":"2.7
years","orbital_speed":286.8759982712291,"planet_mass":3273.34,"planet_radius":12.432,"planet_type":"Gas Giant","specifications":[]},
{"distance_from_earth":"473.0","distance_from_their_sun":"330.0 AU","gravity":71.8773375564931,"name":"1RXS J160929.1-210524 b","orbital_period":"6505.9
years","orbital_speed":15.110929751318338,"planet_mass":2542.4,"planet_radius":18.636799999999997,"planet_type":"Gas Giant","specifications":["gravity","speed"]},
{"distance_from_earth":"314.0","distance_from_their_sun":"0.19 AU","gravity":14.723350833299412,"name":"24 Bootis b","orbital_period":"30.4
days","orbital_speed":14.723350833299412,"planet_mass":289.19800000000004,"planet_radius":13.888,"planet_type":"Gas Giant","specifications":["gravity"]},
{"distance_from_earth":"236.0","distance_from_their_sun":"1.333 AU","gravity":34.959707663401126,"name":"24 Sextantis b","orbital_period":"452.8
days","orbital_speed":34.959707663401126,"planet_mass":632.422,"planet_radius":13.328,"planet_type":"Gas Giant","specifications":["gravity","goldilock"]},
{"distance_from_earth":"236.0","distance_from_their_sun":"2.08 AU","gravity":13.914375512788451,"name":"24 Sextantis c","orbital_period":"2.4
years","orbital_speed":13.914375512788451,"planet_mass":273.308,"planet_radius":13.888,"planet_type":"Gas Giant","specifications":["gravity"]},
{"distance_from_earth":"154.0","distance_from_their_sun":"84.0 AU","gravity":14.160487778997362,"name":"2MASS J01033563-5515561 AB b","orbital_period":"1767.2
years","orbital_speed":14.160487778997362,"planet_mass":4131.400000000001,"planet_radius":12.32,"planet_type":"Gas Giant","specifications":["speed"]},
{"distance_from_earth":"117.0","distance_from_their_sun":"52.0 AU","gravity":26.114807664609522,"name":"2MASS J01225093-2439505 b","orbital_period":"593.2
years","orbital_speed":26.114807664609522,"planet_mass":7786.1,"planet_radius":11.2,"planet_type":"Gas Giant","specifications":["speed"]},
{"distance_from_earth":"129.0","distance_from_their_sun":"156.0 AU","gravity":7.90628123372163,"name":"2MASS J02192210-3925225 b","orbital_period":"5878.1
years","orbital_speed":7.90628123372163,"planet_mass":4417.42,"planet_radius":16.128,"planet_type":"Gas Giant","specifications":["speed"]},
{"distance_from_earth":"457.0","distance_from_their_sun":"15.0 AU","gravity":10.872616442059497,"name":"2MASS J04414489+2301513 b","orbital_period":"411
years","orbital_speed":10.872616442059497,"planet_mass":2383.5,"planet_radius":12.655999999999999,"planet_type":"Gas Giant","specifications":["speed"]},
{"distance_from_earth":"210.0","distance_from_their_sun":"46.0 AU","gravity":73.95246449569838,"name":"2MASS J12073346-3932539 b","orbital_period":"2207.3
years","orbital_speed":6.208420134812571,"planet_mass":1271.2,"planet_radius":12.991999999999999,"planet_type":"Gas Giant","specifications":["gravity","speed"]},
{"distance_from_earth":"1307.0","distance_from_their_sun":"0.92 AU","gravity":32.82462306046568,"name":"2MASS J19383260+4603591 b","orbital_period":"416
days","orbital_speed":32.82462306046568,"planet_mass":603.82,"planet_radius":13.44,"planet_type":"Gas Giant","specifications":["gravity","goldilock"]},
```

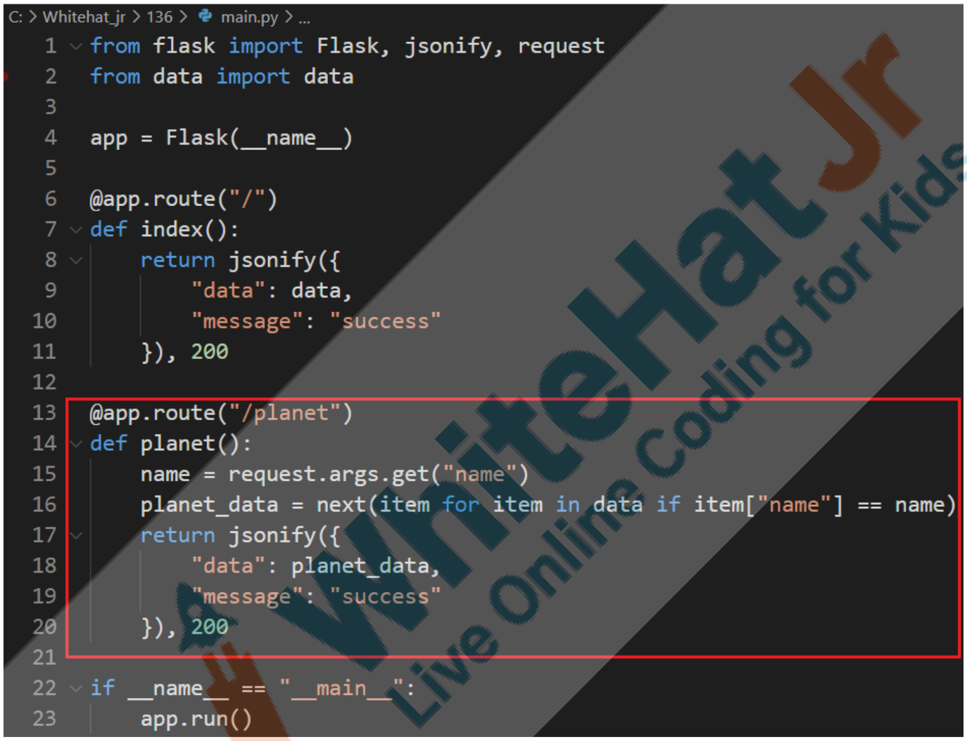
Great! Now our first API is ready, that returns the data of all the planets!

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 the user is 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.

1. Create another **route()** function which will take the name of the planet from the URL argument and then find its data in the dictionary and then return the data **(/planet)**.
2. Define the function to get the data of the planet.
3. The **requests.args** function is used to get the name from the URL. Here, we are first fetching the name of the planet from the URL argument.

4. Use the **next()** function to create and iterator and find a dictionary (element of list **planet_data**) that satisfies the condition, which is, the value of name should match with the name we are providing!
5. We are then finally returning only the planet's data this time.

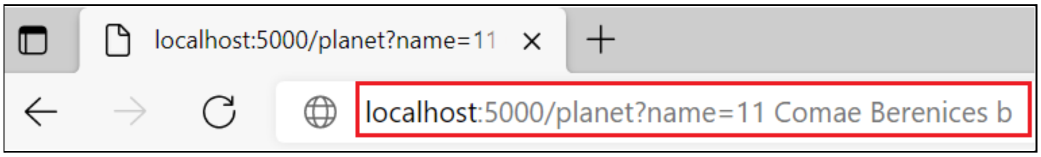


C: > Whitehat_jr > 136 > main.py > ...

```
from flask import Flask, jsonify, request
from data import data

app = Flask(__name__)

@app.route("/")
def index():
    return jsonify({
        "data": data,
        "message": "success"
    }), 200

@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

if __name__ == "__main__":
    app.run()
```

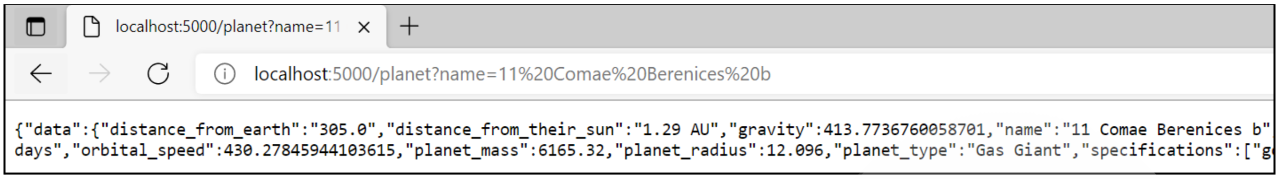
Re-run the server and go to the index route, copy the name of any planet and try to get the data for that planet using this API. Write the route name as shown below.

Go to :
localhost:5000/planet?name= <<name of the planet>>



localhost:5000/planet?name=11 Comae Berenices b

This gives us the details of the required planet only.


localhost:5000/planet?name=11
localhost:5000/planet?name=11%20Comae%20Berenices%20b

```
{"data":{"distance_from_earth":"305.0","distance_from_their_sun":"1.29 AU","gravity":413.7736760058701,"name":"11 Comae Berenices b"
days","orbital_speed":430.27845944103615,"planet_mass":6165.32,"planet_radius":12.096,"planet_type":"Gas Giant","specifications":["g
```

Well done!!!
We have successfully created the dictionaries for all the planets and displayed them using Flask.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

Teacher Starts Slideshow
Slide # to #
<**Note**: Only Applicable for Classes with VA>

Activity details

Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ
Click on In-Class Quiz

Continue WRAP-UP Session
Slide # to #
<**Note**: Only Applicable for Classes with VA>

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Teacher Action	Student Action
You get "hats-off" for your excellent work! 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? 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.	*Make sure you have given at least 2 hats-off during the class for:* **ESR:** varied

Next class, we will be building the mobile app!	
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks ✖ End Class	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Boilerplate Code	https://colab.research.google.com/drive/1OF7sa1Vxq6h8AOywubOiMhNTX5sefWdd?usp=sharing
Teacher Activity 2	Reference Code	https://colab.research.google.com/drive/1g3ZFIwBxw9tQXsEXSJPvvOVOsS_hSROZ?usp=sharing
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/9dd243c8-8f4a-4797-9198-545fd33109fe.pdf
Teacher Reference 2	Project Solution	https://colab.research.google.com/drive/1UoyIqkJ79bhtk0p_fnT4QedMROpgD64p?usp=sharing
Teacher Reference 3	Visual-Aid	Will be added after VA creation
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/214b786e-6604-4552-a69b-b79a374d65f7.pdf
Student Activity 1	Boilerplate Code	https://colab.research.google.com/drive/133kE03STNgsVP8be8CkMxu9OZzHqBHty?usp=sharing
Student Activity 2	Boilerplate Code	https://github.com/procodingclass/PRO-C136-Studetnt-Activity-2