Topic	OUTPUT VALIDATION	
Class Description	**The student will review the output and correct the code to make sure their final output is correct.**	
Class	**PRO C135**	
Class time	**45 mins**	
Goal	• Reviewing the output • Debugging the code • Fixing the code to achieve the right output	

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**	**10 mins** **10 mins** **20 mins** **05 mins**
Credit & Permissions:	**Exoplanet Exploration by NASA**	
WARM-UP SESSION - 10 mins		
Teacher Starts Slideshow **Slide # to #** <**Note**: Only Applicable for Classes with VA>		

Refer to speaker notes and follow the instructions on each slide.	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It's great to see you! **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide # to #** <**Note**: Only Applicable for Classes with VA>	
Activity Details **Following are the session deliverables:** • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring more interest in students.	
Teacher Action	**Student Action**
Great! Now, we generated a dictionary containing all the data in the last class, but was the output correct? In today's class, we will be seeing how we can validate the output, debug the code, etc. Are you excited? Let's start coding.	**ESR:** Yes, I am excited about it!

<table>
<tr><td colspan="2">Teacher Ends Slideshow  </td></tr>
<tr><td colspan="2">TEACHER-LED ACTIVITY - 10 mins</td></tr>
<tr><td colspan="2">Teacher Initiates Screen Share</td></tr>
<tr><td colspan="2"><u>ACTIVITY</u>
• Verifying the output
• Debugging the code</td></tr>
<tr><td>Teacher Action</td><td>Student Action</td></tr>
<tr><td>Note: open Teacher Activity 1 for the Boilerplate code.

In today's class, we will be reviewing the output we achieved in the last class, which is the list of all the specifications that a particular planet exhibits.

To begin with, let's just see how many planets have gravity as a feature listed in the dictionary!
1. Iterate over all the key-value pairs in the dictionary. We know that the values would be the list of specifications or features.
2. Check if gravity exists in the list or not and print the count of such planets having gravity as a feature.
3. Iterate over all the key-value pairs in the dictionary. We know that the values would be the list of specifications. We can simply just check if gravity exists in the list or not and print the count of such planets having gravity as a feature.

To maintain the count in the gravity_planet_count variable and we are checking if the value has the gravity keyword in it or not.
If the gravity is there, we are increasing the count by 1 and we are finally printing the count.</td><td></td></tr>
</table>

```
gravity_planet_count = 0
for key, value in final_dict.items():
  if "gravity" in value:
    gravity_planet_count += 1

print(gravity_planet_count)
```

```
3951
```

Great! The count comes out to be 3,951.

This is exactly the number of planets with suitable gravity that we found out earlier! Let's move ahead to the **planet_type** now.

This time, we will be checking all the planets that have the feature **planet_type** listed in them. We will find out the count using the same type of code as before.

Here, we are again maintaining the count in a variable known as **type_planet_count** and we are iterating over all the keys and values of a dictionary. We are checking if **planet_type** is listed in the value or not, and if it is, we are increasing the variable's value by **1**.

Finally, we are printing the count.

```
type_planet_count = 0
for key, value in final_dict.items():
  if "planet_type" in value:
    type_planet_count += 1

print(type_planet_count)
```

```
1485
```

This time, the value came out to be 1,485 but if we remember, earlier it came out to be 1,452. How did the values change? Do you have any ideas?	**ESR:** There might be planets that do not support gravity and only planet type. We found out the list of planets earlier with planets that support gravity.
Great! Let's validate that as well. According to what we can analyze, there should be 33 such planets that do not support gravity but have supported planet types! Let's write a code for that. To achieve this, we can simply create a list of planets that does not support gravity, then iterate over these planets and check if they are of supported types. The count of such planets should be 33. Here, we first create a list **planet_not_gravity_support** where we will keep all the planets that support gravity. Next, we will iterate over all the planets and check if it exists in **low_gravity_planets** or not. If it does not, we add it to the list we created. Then, we will create a variable **type_no_gravity_planet_count** which will maintain the count of planets that support type but not gravity for us. Once this is done, we will iterate over all the planets we found out that do not support gravity and check their type. If they are of Terrestrial or Super Earth-type, we will increase our count by **1**. Finally, we have the print statements where we print the count of planets that support only the type and subtract	

this number is from the number of planets that have supported type.

```
planet_not_gravity_support = []
for planet_data in planet_data_rows:
  if planet_data not in low_gravity_planets:
    planet_not_gravity_support.append(planet_data)
```

```
type_no_gravity_planet_count = 0
for planet_data in planet_not_gravity_support:
  if planet_data[6].lower() == "terrestrial" or planet_data[6].lower() == "super earth":
    type_no_gravity_planet_count += 1
```

```
print(type_no_gravity_planet_count)
print(type_planet_count - type_no_gravity_planet_count)
```

```
33
1452
```

Here, we get the count to be 33! Awesome. So far so good!

Now we just need to check the number of planets that are in the goldilocks zone, and the number of planets that support speed.

We can do that using a similar code as above!

Here, again we are doing the same steps as above but this time, we are checking for the feature **goldilocks** in the specification list as we did in the previous class.

```
goldilock_planet_count = 0
for key, value in final_dict.items():
  if "goldilock" in value:
    goldilock_planet_count += 1

print(goldilock_planet_count)
```

```
696
```

Here, again we are doing the same steps as above but this time, we are checking for the feature **speed** in the specification/feature list.

```
speed_planet_count = 0
for key, value in final_dict.items():
  if "speed" in value:
    speed_planet_count += 1

print(speed_planet_count)
```

```
676
```

We got the planets in the goldilocks zone to be 696 and planets supporting speed to be 676.

Also, from our previous data, we have the Habitable planets as 25 and the count of goldilocks planets from the suitable planet list was also 25.

Although we haven't validated the code yet, let's make sure we are not committing any mistakes when creating the dictionary.

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 debugs the code** • **Student fixes the code**	
Teacher Action	**Student Action**
Here, the first thing we need to do is to revisit the code. If you remember, in class 134, we made a few changes in the columns! We ensured that the 9th column on orbital radius should be a float value and that the orbital period value is converted to date and is in float. If we look at the code we have written to create the final dictionary, we can see that we are not performing any such changes to these values and we did not make these	

changes in our main list. We also handled the **Unknown** values previously on the **suitable_planets** list but we did not do that here! Let's do that first!

Open Student Activity 1 for the boilerplate code.

The earlier code looks like this -

```
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])*6371000*6371000) * 6.674e-11
  try:
    if gravity < 100:
      features_list.append("gravity")
  except: pass
  try:
    if planet_data[6].lower() == "terrestrial" or planet_data[6].lower() == "super earth":
      features_list.append("planet_type")
  except: pass
  try:
    if planet_data[8] > 0.38 or 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")
  except: pass
  final_dict[index] = features_list
```

Here, when we are trying to add the goldilocks zone, we are not performing any sort of operations on the string.

The string was in the following format **X AU**.

Here, we need to split the string from a <space>. Take the first element and convert it into a **float** value. Let's quickly do that!

***Note:** We also want to change the **or** operator to **and** operator in the **if** condition for Goldilock planets.*

```
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])*6371000*6371000) * 6.674e-11
  try:
    if gravity < 100:
      features_list.append("gravity")
  except: pass
  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:  #Add float and split here

      features_list.append("goldilock")
  except: pass
```

Great! Similarly, we need to ensure that the speed is calculated the right way as well before we put a condition to it. For that, we need to fix the distance (by converting it into a float after splitting it) and we also need to convert **orbital_period** into days!

Here, we are using the **split()** function with indexing to find the value in digits for time and distance, we are then converting these values to **float** from the string. Now, we are checking the unit of time. If it is not in days, we are multiplying it by **365** to convert it into days.

```
  try:
    distance = 2 * 3.14 * (float(planet_data[8].split(" ")[0]) * 1.496e+8)     #Add float and split here
    time, unit = planet_data[9].split(" ")[0], planet_data[9].split(" ")[1]
    if unit.lower() == "days":                                                 # Check if days
      time = float(time)                                                       # Add float for time
    else:
      time = float(time) * 365
    time = time * 86400
    speed = distance / time

    if speed < 200:
       features_list.append("speed")
  except: pass
  final_dict[index] = features_list

print(final_dict)
```

Let's run this and again check the values of goldilocks planets and planets that have speeds we can tolerate.

```
goldilock_planet_count = 0
for key, value in final_dict.items():
  if "goldilock" in value:
    goldilock_planet_count += 1

print(goldilock_planet_count)
```

```
390
```

```
speed_planet_count = 0
for key, value in final_dict.items():
  if "speed" in value:
    speed_planet_count += 1

print(speed_planet_count)
```

```
1987
```

These planets were also supporting both **Gravity and Planet Type**. To check if we are correct, let's write a code where we find all the planets having all of Gravity, Planet Type, and Goldilock as specifications.

We also need to find all the planets that support Gravity, Planet Type, Goldilock, and Speed. This should come out to be 25 (based on what we did in the previous class).

```
goldilock_gravity_type_count = 0
for key, value in final_dict.items():
  if "goldilock" in value and "planet_type" in value and "gravity" in value:
    goldilock_gravity_type_count += 1

print(goldilock_gravity_type_count)
```

```
0
```

```
speed_goldilock_gravity_type_count = 0
for key, value in final_dict.items():
  if "goldilock" in value and "planet_type" in value and "gravity" in value and "speed" in value:
    speed_goldilock_gravity_type_count += 1

print(speed_goldilock_gravity_type_count)
```

```
0
```

Here, both the values are coming out to be **0**! Which is wrong. So let's try to debug the code and check if we have added any wrong steps.

For doing that, we can just simply remove the **try-except** statement from the goldilocks part of our code that generates the dictionary, to see if there are errors that it is handling that we do not know about!

```
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])*6371000*6371000) * 6.674
  try:
    if gravity < 100:
      features_list.append("gravity")
  except: pass
  try:
    if planet_data[6].lower() == "terrestrial" or planet_data[6].lower() == "super earth":
      features_list.append("planet_type")
  except: pass

  if float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:  #remove try-except here
      features_list.append("goldilock")

  try:
   distance = 2 * 3.14 * (float(planet_data[8].split(" ")[0]) * 1.496e+8)     #Add float and split here
   time, unit = planet_data[9].split(" ")[0], planet_data[9].split(" ")[1]
   if unit.lower() == "days":                                                 # Check if days
     time = float(time)                                                       # Add float for time
```

Run the above code to check the output.

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-1eb241c8ecf5> in <module>()
     14   except: pass
     15
---> 16   if float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:
     17       features_list.append("goldilock")
     18

ValueError: could not convert string to float: 'Unknown'
```

SEARCH STACK OVERFLOW

Here, it says it cannot handle the unknown value! Let's handle it ourselves and rerun this.

```
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])*6371000*6371000) * 6.674e-11
  try:
    if gravity < 100:
      features_list.append("gravity")
  except: pass
  try:
    if planet_data[6].lower() == "terrestrial" or planet_data[6].lower() == "super earth":
      features_list.append("planet_type")
  except: pass

  if not planet_data[8].lower()=='unknown' and float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:
      features_list.append("goldilock")
```

Run the above code to check the output.

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-28-2765e43e2189> in <module>()
     14   except: pass
     15
---> 16   if not planet_data[8].lower()=='unknown' and float(planet_data[8].
     17       features_list.append("goldilock")
     18

AttributeError: 'float' object has no attribute 'lower'
```

It says that it is trying to split a float value. This means that there might be some values that are already a float!

Let's quickly fix this by handling the case where if the value is float and gives errors, it takes the value as it is.

```
[ ]  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")
      except: pass
      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
```

We can make the same changes for the speed as well, assuming that this weird behavior is happening due to a mix of different types of values (Float, String, Unknown).

```
    try:
      try:
        distance = 2 * 3.14 * (float(planet_data[8].split(" ")[0]) * 1.496e+8)
      except:
        try:
          distance = 2 * 3.14 * (float(planet_data[8]) * 1.496e+8)
        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")
    except: pass
    final_dict[planet_data[1]] = features_list

print(final_dict)
```

Let's run this and check whether **final_dict** is created or not.

```
{'11 Comae Berenices b': ['goldilock', 'speed'], '11 Ursae Minoris b': ['goldilock', 'speed'],
```

So our dictionary is in the form of keys that are the name of the planet and values are the features of these planets.

Now, let's try re-running the code blocks that gave us 0 values for planets supporting Gravity, Type, and Goldilock (near to 25) and planets supporting all of the Gravity, Speed, Type, and Goldilock (near 25).

```
goldilock_gravity_type_count = 0
for key, value in final_dict.items():
  if "goldilock" in value and "planet_type" in value and "gravity" in value:
    goldilock_gravity_type_count += 1

print(goldilock_gravity_type_count)
```

```
24
```

```
speed_goldilock_gravity_type_count = 0
for key, value in final_dict.items():
  if "goldilock" in value and "planet_type" in value and "gravity" in value and "speed" in value:
    speed_goldilock_gravity_type_count += 1

print(speed_goldilock_gravity_type_count)
```

```
24
```

Awesome! Now, we have successfully debugged our code and our output is validated!

So, in this class, we validated the output and debugged our code. It was essential for us to cross-check the output/result before building a Flask API for the same, or else it would have been a disaster that our app displays false data.

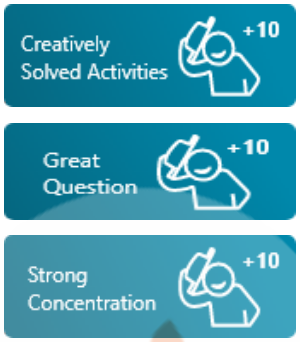
As you might have noticed, simply missing small details can cause a huge impact on the output. The values that we were getting earlier without fixing our code were far away from the actual values! Thus, it is essential to always revisit your code and make sure you're not missing out on steps.

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!	*Make sure you have given at least 2 hats-off during the class for:*

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. Next class, we will be learning new concepts and building new projects.	
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks 	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Boilerplate Code	https://colab.research.google.com/drive/1iS47ylUKk0Vb07_bNMQnE4A49vHv0MiK?usp=sharing
Teacher Activity 2	Reference Code	https://colab.research.google.com/drive/1rpk1QPKfMUMQOUGlJufafsrHLGLhPEZ4?usp=sharing
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/97aed3c5-9fbf-4cb6-ad89-15aaf2c32b4a.pdf
Teacher Reference 2	Project Solution	https://colab.research.google.com/drive/1PbXO5MGkX7OQzJ6PmVwjaKERyrPXZZDA?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/1859d989-092f-44ca-a719-324125b26a75.pdf
Student Activity 1	Boilerplate Code	https://colab.research.google.com/drive/1FeWCG2ZLIqL8idIN_N9Z7J_7QzN2-RLS?usp=sharing