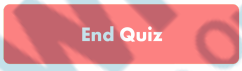
Topic	Output Validation	
Class Description	**Students will review the output and correct the code to make sure their final output is correct.**	
Class	**C135**	
Class time	**45 mins**	
Goal	• Reviewing the output • Debugging the code • Fixing the code to achieve the right output	
Resources Required	• Teacher Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen • 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

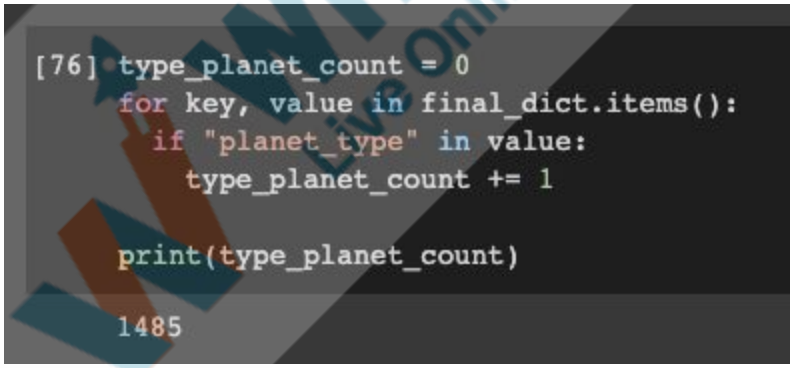
- **Review the concepts learned in the earlier classes**

Class Steps	Teacher Action	Student Action
Step 1: Warm Up (5 mins)	Hi <Student Name>! In the last class, we came up with the final list of habitable planets that meet all the 4 filters we had applied! Can you tell me what these filters were?	**ESR:** - Gravity - Planet Type - Goldilock - Speed

	I have an exciting quiz question for you! Are you ready to answer this question? Teacher click on the  button on the bottom right corner of your screen to start the In-Class Quiz. A quiz will be visible to both you and the student. Encourage the student to answer the quiz question. The student may choose the wrong option, help the student to think correctly about the question and then answer again. After the student selects the correct option, the  button will start appearing on your screen. Click the End quiz to close the quiz pop-up and continue the class.	
	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?	**ESR:** "Yes!"

	Let's start.	
Teacher Initiates Screen Share		
CHALLENGE • **Verifying the output** • **Debugging the code**		
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.)*	
	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! *<Let the student do majority of the coding>* For this, we will 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. `gravity_planet_count = 0`	*Student writes the code.*

	`` for key, value in final_dict.items(): if "gravity" in value: gravity_planet_count += 1 print(gravity_planet_count) ``` Hare, we are maintaining the count in **gravity_planet_count** variable and we are checking if the value has the gravity keyword in it or not. If it is there, we are increasing the count by 1 and we are finally printing the count.	
	`` [75] 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 it. We will find out the count using the same type of code as before.	*Student writes the code.*

	`` type_planet_count = 0 for key, value in final_dict.items(): if "planet_type" in value: type_planet_count += 1 print(type_planet_count) ``` 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.	
	 [76] 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 comes 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 who 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. `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`	*Student writes the code.*

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

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 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 goldilock zone, and the number of planets that support speed.

We can do that using similar code as above!

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

print(goldilock_planet_count)
```

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

Student writes the code.

<table>
<tr><td colspan="3">

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

     print(goldilock_planet_count)

     696
```

</td></tr>
<tr><td></td><td>

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

print(speed_planet_count)
```

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.

</td><td>Student writes the code.</td></tr>
<tr><td colspan="3">

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

     print(speed_planet_count)

     8
```

</td></tr>
<tr><td></td><td>We got the planets in the goldilock zone to be 696 and planets supporting speed to be 8.
Although we haven't validated it yet, let's make sure we are not committing</td><td></td></tr>
</table>

	any mistakes when creating the dictionary.	
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 debugs the code** • **Student fixes the code**		
Step 3: Student-Led Activity (15 min)	Here, the first thing we need to do is to revisit the code. If we remember, 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! 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+9)
    time = planet_data[9] * 86400
    speed = distance / time
    if speed < 200:
      features_list.append("speed")
  except: pass
  final_dict[index] = features_list

print(final_dict)
```

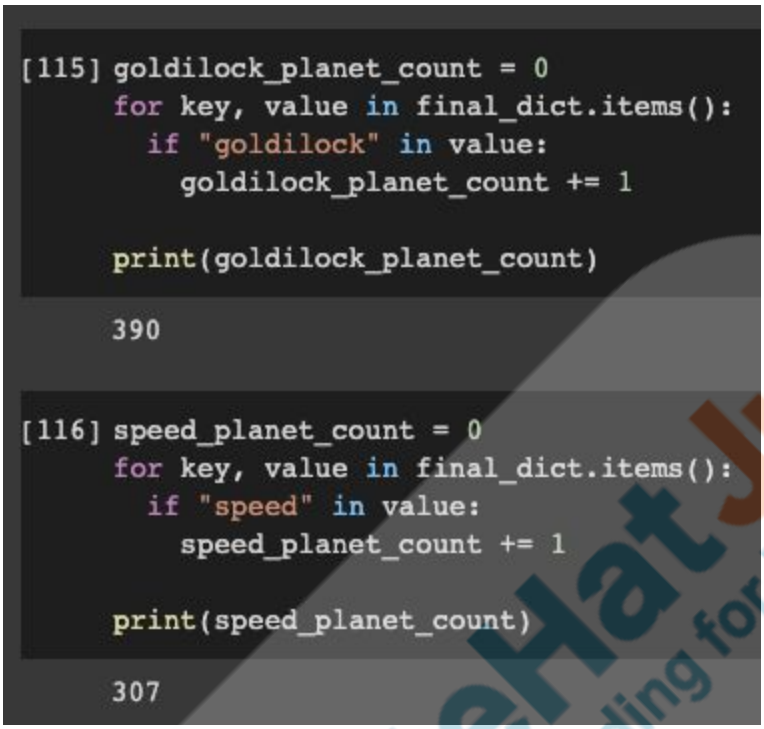
	Here, when we are trying to add the goldilock zone, we are not performing any sort of operations on the string. The string was in the following format **X AU**. Here, we need to split the string from a <space>, take the first element and convert it into a float value. Let's quickly do that! Note - We also want to change the **or** operator to **and** operator in the if condition for Goldilock planets. `if float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:`	*Student changes the code.*

	`` features_list.append("goldilock") ``` We are now performing the operations required in the string.	

```
try:
  if float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:
    features_list.append("goldilock")
```

	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! ``` distance = 2 * 3.14 * (float(planet_data[8].split(" ")[0]) * 1.496e+9) time, unit = planet_data[9].split(" ")[0], planet_data[9].split(" ")[1] if unit.lower() == "days": time = float(time) else: time = float(time) * 365 time = time * 86400 speed = distance / time ``` 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 string. For time, we are checking the	*The student fixes the code and performs string operations.*

	unit of the time. If it is not in days, we are multiplying it by 365 to convert it into days.	
		
	Let's run this and again check the values of goldilock planets and planets that have speeds we can tolerate.	



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

      print(goldilock_planet_count)

      390

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

      print(speed_planet_count)

      307
```

	Okay, so now the values have gotten a bit closer. Our speed supporting planets were 8 and our goldilock planets were coming out to be 25. These planets were also supporting both **Gravity and Planet Type**. To check if we are correct, let's write a code snippet where we find all the planets having all of Gravity, Planet Type and Goldilock as specifications. This should be close to 25. We also need to find all the planets that support Gravity, Planet Type, Goldilock and Speed. This should come out to be 6 (based on what we did in the last class). `goldilock_gravity_type_count = 0`	*Student writes the code.*

```
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_cou
nt)
```

```
speed_goldilock_gravity_type_cou
nt = 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_cou
nt += 1

print(speed_goldilock_gravity_ty
pe_count)
```

Here, we are just checking for all the specifications in the list after iterating over all the key value pairs in our dictionary, maintaining a count for them and then finally printing the count.

```
[117] 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
```

```
[118] 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. This is not right! Let's debug the code. For doing that, we can just simply remove the try and except statement from the goldilock part of our code that generates the dictionary, to see if there are errors that it is handling that we do not know about!	*Student removes the try-except block.*

```
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 float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:
    features_list.append("goldilock")
  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")
  except: pass
  final_dict[planet_data[1]] = features_list

print(final_dict)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-122-b636ffc8e65e> in <module>()
     12       features_list.append("planet_type")
     13   except: pass
---> 14   if float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:
     15     features_list.append("goldilock")
     16   try:

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.	*Student codes to create higher levels in the game.*

```
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")
```

<table>
<tr><td colspan="3">

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-125-261f91623f28> in <module>()
     12         features_list.append("planet_type")
     13     except: pass
---> 14     if not planet_data[8].lower() == "unknown" and float(planet_data[8].split(" ")[0]) > 0.38 and float(planet_data[8].split(" ")[0]) < 2:
     15       features_list.append("goldilock")
     16     try:

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

</td></tr>
<tr><td></td><td>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 it errors, it takes the value as it is.

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).</td><td>The student fixes this for both Goldilock and speed.</td></tr>
</table>

```
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:
  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")
```

	Now, let's try re-running our code blocks that gave us 0 values for planets supporting Gravity, Type and Goldilock (near to 25) and planets supporting all of Gravity, Speed, Type and Goldilock (near to 6).	*Student runs the code.*

```
[128] 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
```

```
[130] 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)

      6
```

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

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)	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 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. 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. Next class, we will be learning new concepts and building new projects.	-
Teacher Clicks 		

Activity	Activity Name	Links
Teacher Activity 1	Solution	https://colab.research.google.com/drive/1-2OHWiQAg1hcMj2f7AloUl0182lRuoI_?usp=sharing