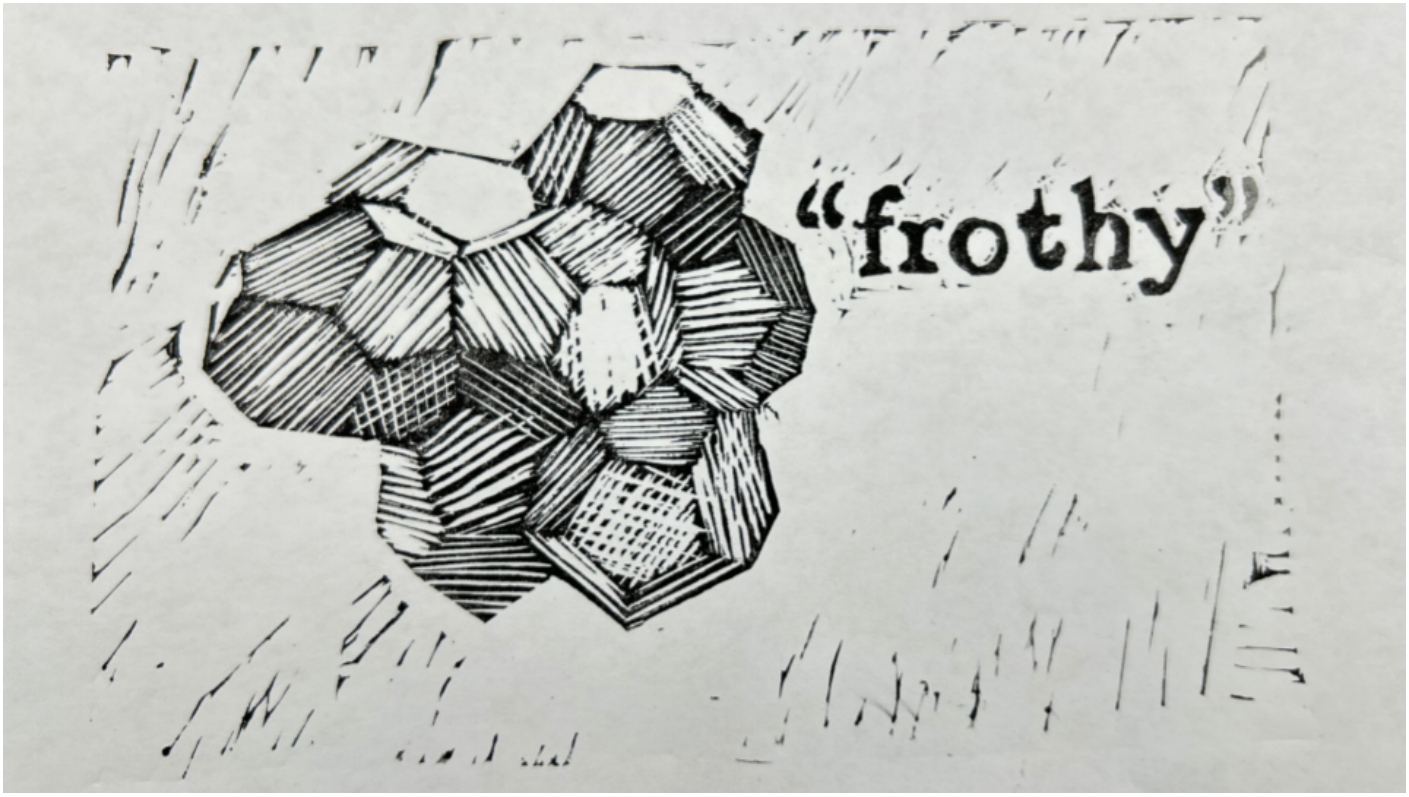
Cover Page: Frothy Cyber Solutions



Designers: Franklin Liu & Ishaan Gupta

PLTW | Computer Science

Table of Contents

Plan & Design Your Solution: SCRUM

Product Backlog

Our company, Frothy Cyber Solutions, has been tasked with the honor of investigating a data breach within a local casino. Our client, Casino Casino, requests that we analyze the data breach by interpreting how the system was broken into, what methods were used to enter the system, and most importantly, what actions were taken to prevent similar incidents from occurring in the future.

Sprint Planning & Sprint Backlog

Tasks	Estimated Time	+/- Time	Completed By	Sprint #
Inside the logs, using the email records, examine which key **employees have used any devices in Casino, Casino.**	5 mins	-1	Franklin	1
Inside the logs, using the email records, identify all **login information, and how strong each one was kept**.	5 mins	+1	Franklin	1
Inside the logs, using the email records, analyze which **employees accessed the fish tank,** and in what ways.	5 mins	-2	Ishaan	1
Inside the logs, using the email records, analyze any **suspicious communication** that may have occurred.	5 mins	-2	Ishaan	1
Inside the downloads, using the python file **fishtank.py**, analyze and see if there are any faults	7 mins	+3	Ishaan	2
Inside the downloads, using the python file **alkalinity.py**, analyze and see if there are any faults	7 mins	+2	Franklin	2
Inside the downloads, using the python file **calcium.py**, analyze and see if there are any faults	7 mins	+1	Franklin	2
Inside the downloads, using the python file **magnesium.py**, analyze and see if there are any faults	7 mins	-1	Ishaan	2
Inside the downloads, using the python file	7 mins	-1	Ishaan	2

ph.py, analyze and see if there are any faults				
Inside the downloads, using the python file **phosphate.py**, analyze and see if there are any faults	7 mins	-1	Franklin	2
Inside the downloads, using the python file **salinity.py**, analyze and see if there are any faults	7 mins	-2	Franklin	2
Inside the downloads, using the python file **temperature.py**, analyze and see if there are any faults	7 mins	+10	Ishaan	2
Look at the files in the Downloads to find any suspicious files.	5 mins	-1	Franklin	2
Analyze the golden key, and attempt to decode the instructions using rsa_decrypt.py from previous lessons.	10	-2	Ishaan	2

Sprint 1

Sprint 1 Tasks

Tasks	Estimated Time	+/- Time	Completed By
Inside the logs, using the email records, examine the activity of each employee.	5 mins	-1	Franklin
Inside the logs, using the email records, identify all login information, and how strong each one was kept.	5 mins	+1	Franklin
Inside the logs, using the email records, analyze which employees accessed the fish tank, and in what ways.	5 mins	-2	Ishaan
Inside the logs, using the email records, analyze any suspicious communication that may have occurred.	5 mins	-2	Ishaan
Inside the Logs folder, look at the 0.log file to understand what it is documenting	10 mins	-2	Franklin

Sprint Retrospective (Ishaan)

Franklin and I worked with synergy, filling the gaps of information that one had, and pitching ideas to each other. I believe that we worked effectively as a group, as we were able to effectively communicate who should be analyzing which specific aspects in the logs between coworkers. This

communication was important because it ensured we were not looking for the exact same thing. However, 1 thing I believe we should work on is to ask questions while working more often. Unlike the Hangman project Franklin and I collaborated upon, we were able to communicate what key ideas we should focus upon. However, this time, while working on sifting through logs, Franklin and I got confused regarding specific messages many times, and we could have easily solved this by simply furthering our communication while completing our independent tasks.

Sprint Retrospective (Franklin)

Ishaan and I worked well together, synergizing with admirable dispatch. We communicated effectively, dividing up our work equally and completing them quickly. Importantly, when looking at the email records, we were looking at the same file but searching for different aspects. We collaborated much better on this project than the Hangman one; in the Hangman project we coded separately and chose 1 implementation, while we communicated more when we were working on this problem. We would have benefited from even more communication, because we communicated before and after doing our tasks, but we could have been even more efficient if we communicated while doing tasks.

Sprint Review (Ishaan)

Overall, we faced only 1 hardship when discovering and exploring our first evidence. Though no code was completed, we spent our time analyzing the log files between co-workers. This was relatively easy and efficient, as we could simply read through lines of communication and try to poke holes in a malicious coworker or malicious email. Franklin and I were easily able to identify some key issues with the communication right away. However, this simply wasn't enough, which was where the hardship came in. Just like in the board game "Battleship", the small boats, or in this case, the small errors, were hard to find. We just barely missed the "s" from the http, which caused us to realize major security issues that could potentially arise with the company.

Sprint Review (Franklin)

We were not terribly excited to dig through a bunch of logs, but we did it anyway. Our main challenge was finding the minute details indicating suspicious activity; for example 'fishtankmonitors' vs 'fishtankmonitors' and 'update.zip' vs 'updates.zip'. However after looking over the duplicate emails again we found them.

Python File(s) & Client Report File(s) for Sprint 1:

Client Report Brainstorm

Sprint 2

Sprint 2 Tasks

Tasks	Estimated Time	+/- Time	Completed By

Look at the files in the Downloads to find any suspicious files.	5 mins	-1	Franklin
Analyze the golden key, and attempt to decode the instructions using rsa_decrypt.py from previous lessons.	10	-2	Ishaan
Inside the downloads, using the python file fishtank.py, analyze and see if there are any faults. Fix them if so.	7 mins	+3	Ishaan
Inside the downloads, using the python file alkalinity.py, analyze and see if there are any faults. Fix them if so.	7 mins	+2	Franklin
Inside the downloads, using the python file calcium.py, analyze and see if there are any faults. Fix them if so.	7 mins	+1	Franklin
Inside the downloads, using the python file magnesium.py, analyze and see if there are any faults. Fix them if so.	7 mins	-1	Ishaan
Inside the downloads, using the python file ph.py, analyze and see if there are any faults. Fix them if so.	7 mins	-1	Ishaan
Inside the downloads, using the python file phosphate.py, analyze and see if there are any faults. Fix them if so.	7 mins	-1	Franklin
Inside the downloads, using the python file salinity.py, analyze and see if there are any faults. Fix them if so.	7 mins	-2	Franklin
Inside the downloads, using the python file temperature.py, analyze and see if there are any faults. Fix them if so.	7 mins	+10	Ishaan

Sprint 2 Review (Ishaan):

Through this Sprint, we noticed that our collaboration took a major hit. With lots of independent work, the review and discussion we had after sprint 1 took a 180. Rather than discussing and communicating beforehand, we only asked each other questions while working on our assigned tasks. However, I believe that this method was better for the sprint we were working on because the main focus was to actually analyze and edit files in programming, which generally requires more questions than Sprint 1. In the future however, I would like to be in full communication and collaboration with Franklin, where we are able to previously decide the tasks effectively and also discuss questions that arise during the completion of them.

Sprint 2 Review (Franklin):

In this sprint, we split up the work (we both took 3 files), but did not talk as much before and after. However, while we were working on the files, if one of us had a question about the code we would

ask the other. This worked for sprint 2, but it would be better if we talked more before and after our sprints so we are fully synchronized.

Sprint 2 Retrospective (Ishaan):

In this sprint, we spent a majority of our time programming and editing files. Initially, Franklin and I discovered the files had some major errors, which resulted in the unexpected error message to not even be represented on the monitors. This was extremely bad because, although, for example, the alkalinity level may be too low, or an error may arise, neither of those potential outputs would be shown, and the status would show that everything was okay. However, we also found a few key hardships when doing this. The little, small errors ended up being where Franklin and I spent a majority of our time. For instance, we did not realize that the initial message before finding a manual check stated that the status was okay, even though it truly wasn't.

Sprint 2 Retrospective (Franklin):

We spent our time reading the code in the updates.zip file and fixing any errors. They were in general riddled with errors, spelling mistakes and poor code. For example, several files had code that would always result in an error, but in the fishtank management GUI they would always show up as OK. We spotted most of these, but had some challenges finding small issues; an example is the fish tank gui saying "All factors OK" regardless of whether there was an issue.

Python File(s) & Client Report File(s) for Sprint 2:

Replit Fixed Code
Client Report Brainstorm

Final Report & Client Recommendations

Video Submission

Final Video Submission Link

Final Client Report/Presentation

Final Client Report

Program Code

[Final Program Code Screenshots]

Rubric

If I used this as my create task submission my score would be: **15/18**

Based on the rubric, we believe that our client report and project overall represents a full score on the CSE Create Task Rubric. We were able to apply many different methods of collaboration, from independently working to discussing with other employees on potential ideas and solutions regarding the malicious activity. Our team fostered a positive attitude towards engagement because of how we discussed and presented our ideas to one another. Using SCRUM strategies and working in sprints ensured that we stayed engaged and were consistently reflecting upon both the work we were doing and the collaboration that came with it.

As seen in our demonstration video, we were able to fully fix the errors that resulted in the terminal which created an unexpected error, and present the proper status of the water treatment. For instance, when the sensor returns the temperature to the program, no longer was there a fault in the code, but the monitor now showed the temperature was too high, too low, or the perfect temperature. The program's purpose was to accurately monitor the fishtank, and present the conditions the sensors were reading to the monitor. Specific stakeholders including employees such as Camila seem to be in charge of this monitoring, along with the overall Casino, Casino. We also showed instances of how our fixes in the program and monitors ensured that every sensor was being properly read, and thus, our Clients' concerns of saving the fish were addressed.

Furthermore, we integrated a multitude of algorithms, where each sensor monitoring counted as a technical algorithm. The program for every algorithm would take in readings, and crosscheck those readings with the ideal conditions of the fishtank.

For instance, let's analyze the fishtank.py file and the temperature.py. Alone, temperature.py uses try/except algorithm to determine whether the temperature of the fish tank is at an ideal position. It begins with the best case scenario of an OK temperature, where the program should then set a max and min temp, then take the readings from the sensor. After reading the average of the 3 readings, the code should determine whether the temp is too hot or too cold, and return this information.

Next, the fishtank.py file with the monitor function should read the environmental factors of the tank. After appending the results from each file for every factor, the program should then use the imported message box to return the results onto the monitor. These 2 functions work in collaboration because the fishtank.py file requires a correct monitoring of temperature.py in order to accurately present results on the monitor.

Moreover, our team has developed abstraction by ensuring that every function has a specific, designated task that can be called upon effectively. For instance, in every file, each function, such

as a monitor, can be used to monitor their respective conditions. Each function has their own designated minimum and maximum condition to crosscheck with the fishtank, and determine whether the temperature is ideal.

The abstraction can manage complexity effectively because it ensures that each reading is subject to its own file and own function. If every reading code was all in 1 function, it would become more difficult for another editor to read and interpret code.

Additionally, in order to understand the issues with temperature.py, alkalinity,py, and magnesium.py, we should have set specific breakpoints at where we believed the code may go wrong. However, because the program was very short and blocked out effectively, we were able to quickly identify where the errors occurred without using breakpoints. User interface was redesigned on the cover of the readings, where we ensured to check whether the conditions were okay or not. Because every person has their own level of skepticality, depending who they are, they may notice and doubt the initial reading, or perform a manual check regardless.

Finally, our team was able to format a video that documented our changes in multiple video clips designed to show the changes we made to the program, and accurately represent each condition working properly. Therefore, because our team believes to have passed all requirements except for using a breakpoint/inspecting vars with a debugger, and we could have improved our usages of abstraction, we deserve a 15/18.

Conclusion

Ishaan's Reflection:

At the end of the day, Franklin and I made up a great team. We were able to consistently work together, and because we enjoyed one another's company, it never became tiring to ask one another questions, or work together while having a side discussion. Overall, we were able to manage time effectively. This was because we followed our amazing educators' key advice- to divide and conquer. We were able to get all the tasks done, yet we could've improved our time by spending more time on the project. Whenever Franklin and I did work together, it was extremely efficient and productive. However, we could have also spent more time overall on the project rather than procrastinating. In the future, Franklin and I should set up a timeline by which we can follow and adhere throughout the project. A lack of security awareness overall was a major contributing factor to the fish tank monitoring software, as this lack thereof directly led to the malicious updates being installed. For example, multiple basic safety protocols had been breached, including the management of a safe password, or the communication with other team members regarding the implementation of various changes to a product. Finally, debugging techniques can help me fix the problems with software because they enable me to properly understand how to tackle the issues in the software. When Franklin and I first opened up the downloads folder, we discovered multiple

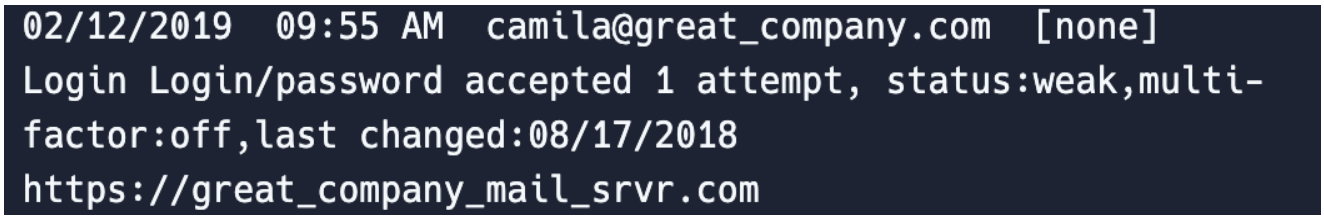
different files with chunks of code. Thus, we had to isolate each program by debugging each and only running 1 at a time. This enabled us to figure out the issue with each program, rather than trying to manage them all at once.

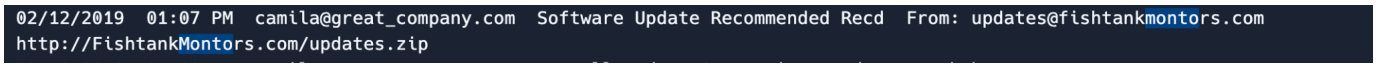
Franklin's Reflection:

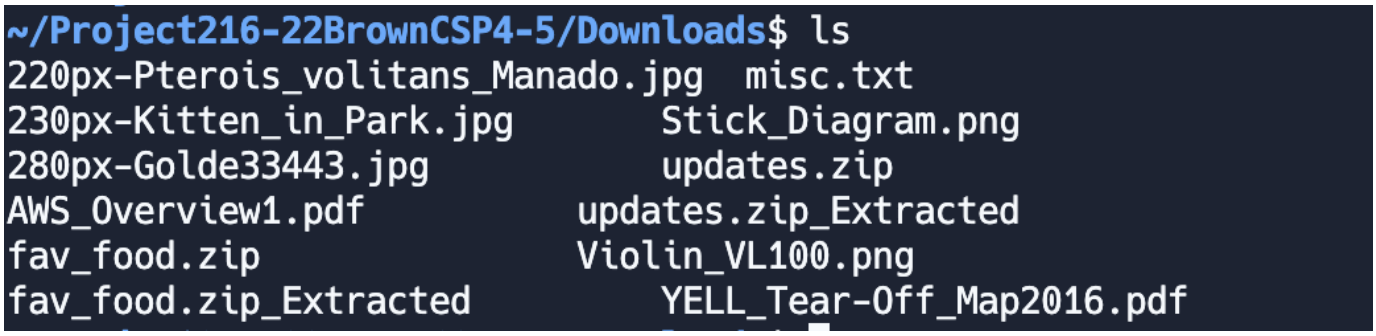
Overall, Ishaan and I worked well together. However, there were several things we could improve on. During class, when we did work we did it efficiently; the only issue was that sometimes we were distracted from doing work. We plan to amend this by reducing sources of distractions (closing other tabs or windows), and reminding ourselves that the more work we get done in class the less we have to do outside.
When analyzing the problems with the fish tank monitoring software, we found that a chain of events led to the malicious software making its way in. It starts with the phishing attack, which could have easily been avoided had Camila been properly trained in security best practices.
When fixing problems with the software, we littered it with print statements (a common debugging technique), allowing us to find what file the error came from and inspect intermediate values in the program to pinpoint the exact source of the issues.

Project Log

Franklin	Ishaan
Today Ishaan and I looked at Evidence 1. We started by viewing the situation through the lens of an attacker - looking at each casino employee's security practices, so we could find people who could potentially be exploited. Then we decided to look at other aspects of the logs, such as the topics the employees were emailing about and who they were emailing to. I focused on looking at the domain names that the employees were emailing.	Today, on Feb 10, Franklin and I worked on analyzing Evidence 1. We decided to take a look at the logs, and try to gain preliminary data about who was emailing and who was accessing what information. After this, we reconvened and decided to break up our Sprint tasks, so that we could efficiently conclude what errors or mistakes might have occurred. I worked on analyzing employees accessing the fish tank, and what suspicious communication may have occurred.

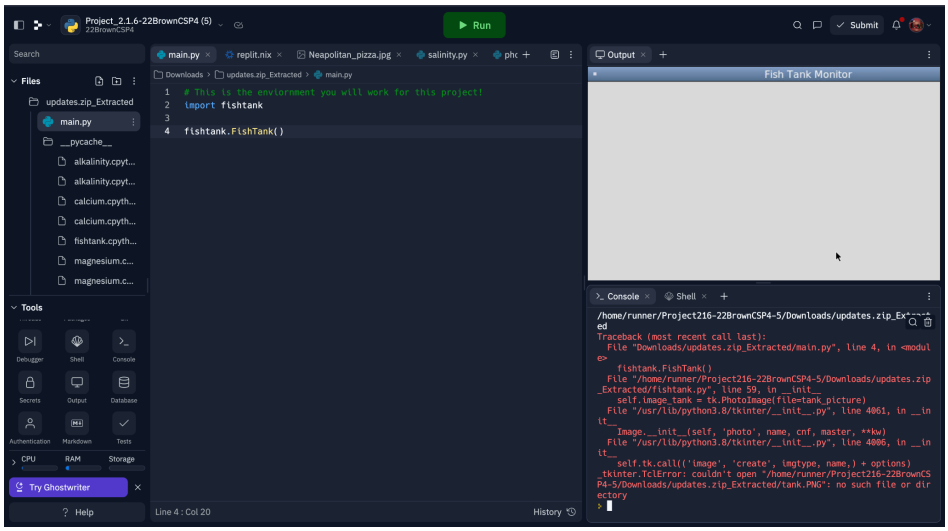






Today Ishaan and I continued looking at Evidence 1. I focused on the activity of each employee, to see if there were any suspicious communications. Ishaan noticed that Bob emailed Camila about the fish tank, and I noticed that Camila got multiple emails about the fish tank. We presented our findings to Mr. Brown and got the golden ticket, so we began looking at Evidence 2. We quickly discovered the updates.zip file, which we determined to be the file that Camila downloaded from the suspicious website. Then we moved on to Evidence 3. I focused on running the fishtank.py file. It turns out that the golden ticket is for decrypting a file in the Downloads folder called misc.txt, but I didn't realize this until after I wrote the code to use fishtank.py

Today, on Feb 22, Franklin & I decided to work on continuing to analyze Evidence 1 in search of the ticket. We were able to efficiently discover the updates.zip file, and thus explore potential issues that may arise with the faulty file. Compared to the last time we worked on the project, I decided to place my focus directly on communications between those who controlled the fish tank. This enabled me to realize the person of the suspect, and continue to search for clues regarding that one person. After we received the golden ticket for our results, Franklin and I split up the work for searching Evidence 2. My job in evidence 2 was to analyze specific python files for the functionality of the fish tank, so that we can scope out any potential issues.



Today Ishaan and I continued looking at the code in the updates.zip file. We found several portions of code in the files magnesium.py, temperature.py and alkalinity.py that would cause an error or were poorly coded. There were also spelling errors, indicating that the code was written in a rush or without care. We fixed several errors, to see the intended results, and we have decided on what recommendations we should make to our client.	Today, on Feb 24, Franklin and I continued to work on sprint 2, and wrap up our analysis for the events and code in the updates.zip file. We noticed that there were many errors and faults in the program, which Franklin and I documented, as well as attempted to solve. We noticed many errors in files such as Magnesium.py, both in grammar and the functionality. After solving these errors, Franklin and I agreed upon some future recommendations that we would like to make to the client in improving the overall quality of security. An example of this was the usage of http in websites related to the company, indicating there is minimal security when accessing potentially important information.
Today Ishaan and I worked on identifying any final problems in the code and fixing them, and drafting our client report.	Today, Franklin and I worked on three key things. First, we finalized our code to ensure there weren't any bugs or there wasn't any general programming error that may have not affected the functionality of the code. Then, we decided to look back over for potential errors in the log and in updates.zip. Finally, we started to brainstorm and work on the client report that we would be submitting to our client, discussing what went wrong, how we fixed it, and future recommendations.
Today we finished up our Client Report, compiling the evidence we gathered into a polished story we can show our esteemed client Casino Casino. We also finished our conclusion	Today, on 3/2, Franklin and I spent a few hours finalizing our Client Report. This enabled us to formulate a story of what exactly happened. Following this, we worked on the

and videos, so we could hedge against any uncertainties that may occur in class.	creation of our video. Finally, we decided to finish off our conclusion and final rubric scoring. This way, we could spend the 40 minutes in class reviewing rather than rushing.
Today, I worked on editing the video for demonstration.	Today, on 3/3, I spent the last 40 minutes in class for the project polishing the client report and our final document.

Appendix

EVIDENCE 1

Before we analyze the issues with the exact events that occurred, let's go over some unique hazardous issues.

1) Foremost, we also see that the access methods overall are not very strong. Lots of employees use “http” sites rather than the secure “https”. This compromises their personal privacy and the company's privacy.

```
02/12/2019  08:30 AM  bob@great_company.com   New info      Recd  From: el@great_company_partner.com
http://great_company_partner.com/projectInfo.html
```

2) Moreover, we notice that many of the users are accessing the devices for their own personal use. This can be dangerous as the sites or information placed in can be compromised and thus make the company more vulnerable.

```
02/12/2019  03:48 PM  bob@great_company.com   Request approved    Recd  From: campingreservations.com
https://campingreservations.com
```

```
02/12/2019  09:57 AM  alice@great_company.com   Travel Arrangements     Recd  From:
admin@travelerhelper.com  https://travelerhelper.com/scheduler.zip
```

3) Furthermore, the employees use the common workplace mail and account for personal communications. Even though the employees work with each other, the communication not related to work should be kept private and not on public logs, unlike seen below.

```
02/12/2019  09:13 AM  alice@great_company.com   Dinner tonight      Sent  To: jj@mailserver.com
```

4) The following evidence represents our evidence regarding our beliefs for the incident. We noticed a multitude of errors within two key areas: the logs and the downloads. First, in the email record logs: The following screenshot depicts an employee, named Camila, logging into the system with a weak password status, multi-factor authentication off, and an unchanged password for around 4 months. These are harmful because a weak password allows an easier password logon, with minimal difficulty. Having multi factor authentication off causes potential adversaries to log into Camila’s

account without another user knowing. When the password isn't changed for a while, the longer the password stays the same, the more likely there may be for a potential compromise.

```
02/12/2019  09:55 AM  camila@great_company.com  [none]
Login Login/password accepted 1 attempt, status:weak,multi-
factor:off,last changed:08/17/2018
https://great_company_mail_srvr.com
```

```
02/12/2019  08:15 AM  bob@great_company.com   [none]       Login
Login/password accepted 1 attempt,status:strong,multi-factor:on,last
changed:01/08/2019    https://great_company_mail_srvr.com
```

5) This shows that Camila is in a vulnerable position, and is openly available to fall for a phishing attempt or malicious attack. Initially, Camila received a forwarded email from her co-worker, Bob, regarding a Software update from the FishTankMonitors, indicating that it was necessary for her to fix the update. However, just under 3 hours later from the timestamp, Camila receives another email regarding a fish tank update with a zip file, this time from the FishTank Motors. Because the email is different from the original one sent in the morning, and Bob, a co-worker, forwarded the initial message, this indicates that there may have been a potential phishing attempt towards Camila.

Bob is not in a vulnerable situation as his password consists of a strong status, multi-factor authentication on, and a recent password change.

```
02/12/2019  10:04 AM  bob@great_company.com   Software Update Recommended
Recd  From: updates@fishtankmonitors.com
https://FishtankMonitors.com/update.zip
02/12/2019  10:04 AM  camila@great_company.com  Software Update
Recommended Recd  From: updates@fishtankmonitors.com
https://FishtankMonitors.com/update.zip
```

6) Here, both Bob and Camila received an email from updates@fishtankmonitors.com, with an update recommended. Later on, Bob forwards his email to Camila regarding the software update, indicating that Camila is in charge of the fish tank monitors.

```
02/12/2019  10:20 AM  camila@great_company.com  Fw: Software Update Recommended Recd  From: bob@great_company.com
https://FishtankMonitors.com/update.zip
```

```
02/12/2019  01:07 PM  camila@great_company.com  Software Update Recommended Recd  From: updates@fishtankmontors.com
http://FishtankMontors.com/updates.zip
```

We also looked at other logs. For example, 0.log contained errors from PyDev (a plugin for the Eclipse IDE for writing Python code), which complained about being unable to find a module that it required. We found this strange, because we did not expect Camila to be writing code, but this in of itself was not suspicious. However, the log did tell us that the host operating system was Windows 7, an old and vulnerable version of Windows that should definitely be upgraded.

```
 - OS:
Windows 7

 - Std output:


 - Err output:
ImportError: No module named site

!STACK 0
java.lang.RuntimeException: Error creating python process - exited before creating sockets - exitValue = (1).
ProcessInfo:

 - Executed: C:\Users\user\AppData\Local\Enthought\Canopy\User\python.exe -u C:\Users\user\.p2\pool\plugins\org.python.pydev_5.1.2.201606231256\pysrc\pycompletionserver.py 57142
```

In addition, in 1-Python.log we found messages from Pylint, which is a static analysis tool for Python (it checks code quality, finds syntax errors, etc). Near the middle of the file we found several messages complaining about errors in fishtank.py, magnesium.py, alkalinity.py and temperature.py, which interestingly were the same three files that we found problems with. The specific errors mentioned are no longer present, but this still suggests that those three files had poor code quality.

```
##########Linting Output - pylint##########
********** Module magnesium
14,0,error,E0001:invalid syntax (<unknown>, line 14)
##########Linting Output - pylint##########
```

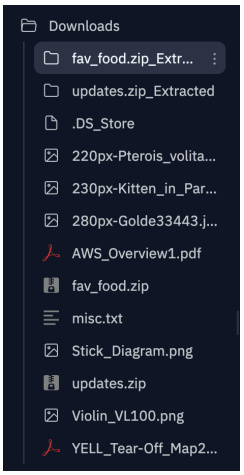
```
##########Linting Output - pylint##########
********** Module temperature
9,8,error,E0602:Undefined variable 'current'
9,18,error,E1136:Value 'temps' is unsubscriptable
11,10,error,E0602:Undefined variable 'current'
11,20,error,E1136:Value 'temps' is unsubscriptable
```

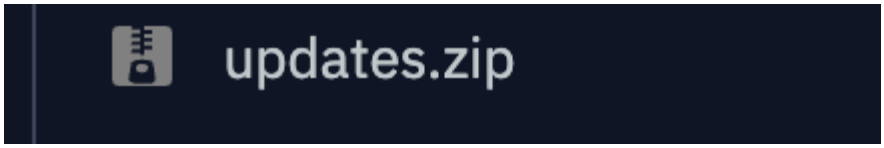
```
##########Linting Output - pylint##########
********** Module temperature
9,18,error,E0602:Undefined variable 'temps'
11,20,error,E0602:Undefined variable 'temps'
```

```
##########Linting Output - pylint##########
********** Module fishtank
53,0,warning,W1401:Anomalous backslash in string: '\P'. String constant might be missing an r prefix.
53,4,warning,W0612:Unused variable 'filename'
```

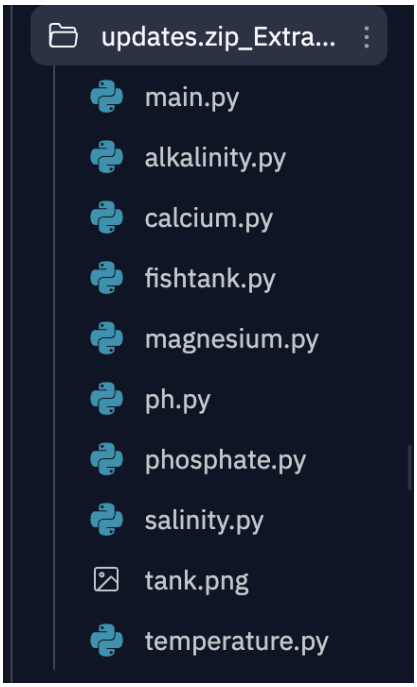
```
##########Linting Output - pylint##########
********** Module alkalinity
10,4,error,E1128:Assigning to function call which only returns None

-------------------------------------------------------------------

Your code has been rated at 7.06/10 (previous run: 10.00/10, -2.94)
```

EVIDENCE 2:



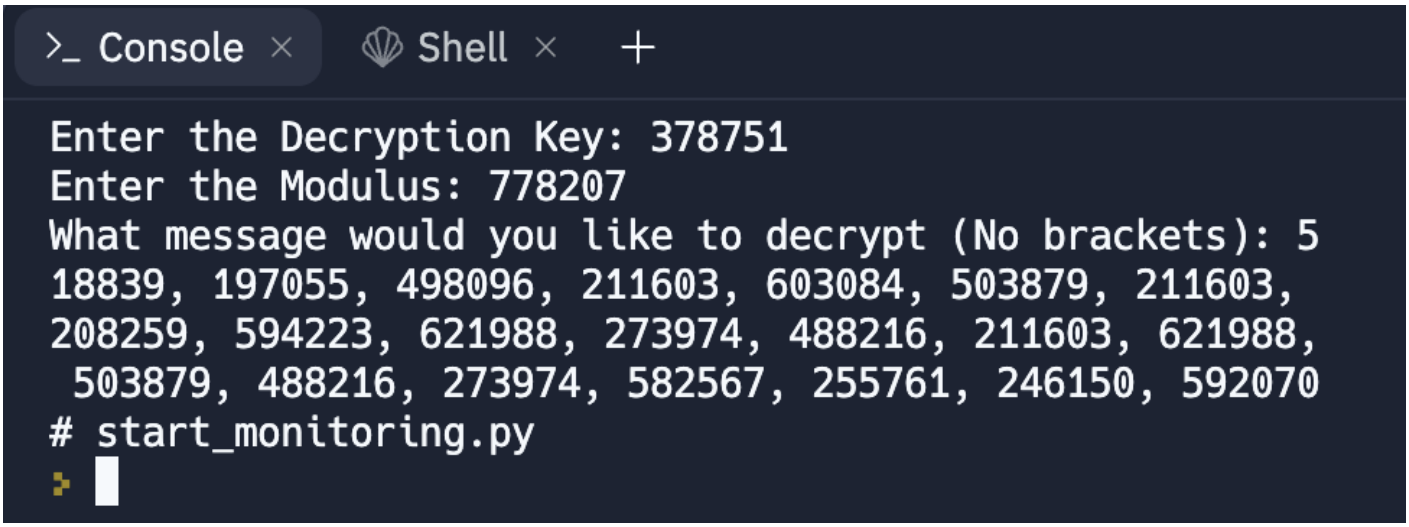


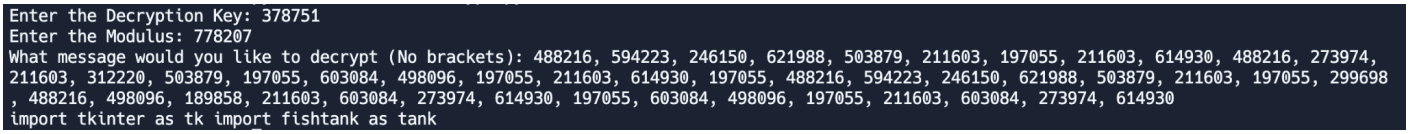

```
~/Project216-22BrownCSP4-5/Downloads$ ls
220px-Pterois_volitans_Manado.jpg  misc.txt
230px-Kitten_in_Park.jpg           Stick_Diagram.png
280px-Golde33443.jpg               updates.zip
AWS_Overview1.pdf                  updates.zip_Extracted
fav_food.zip                       Violin_VL100.png
fav_food.zip_Extracted             YELL_Tear-Off_Map2016.pdf
```

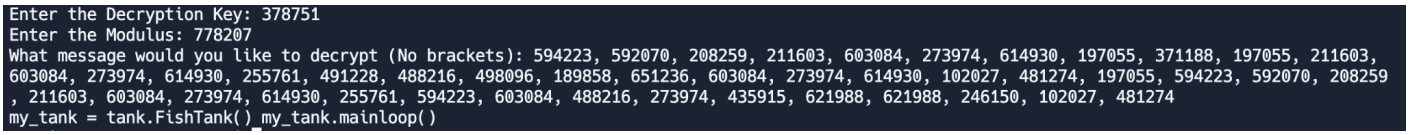


We inspected the Downloads folder, finding miscellaneous pictures (like those of pizza), a Yellowstone map, and an AWS-related PDF. However, what really caught our attention was updates.zip, which matched the name of the file that Camila downloaded. We then extracted it to find Python code, so we decided that this was probably the malicious update for the fish tank.

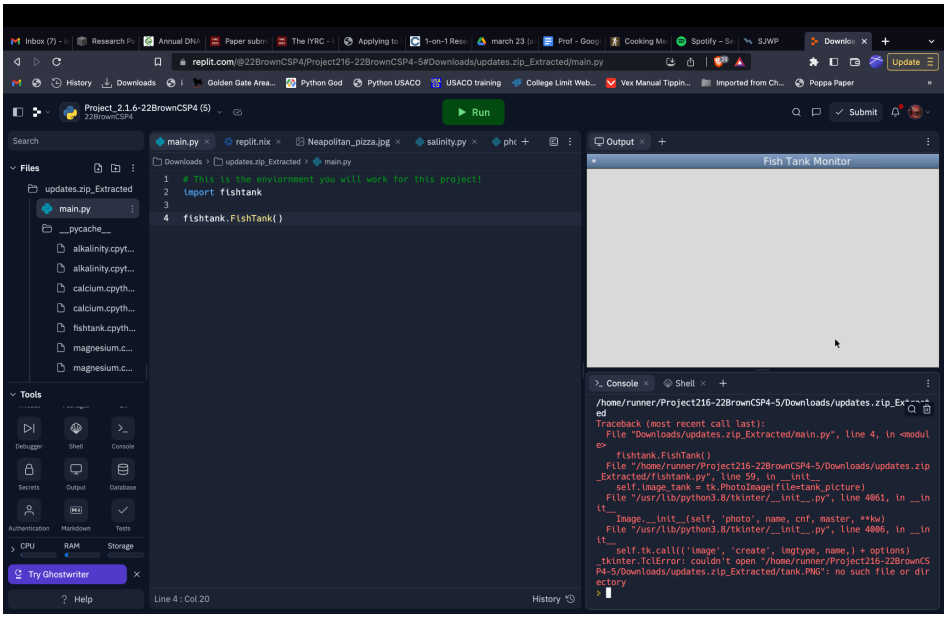
EVIDENCE 3:
After receiving our “golden ticket”, we explored the downloads folder to find the misc.txt file. Using our rsa_decrypt python file, we were able to decrypt the message to give clues as to how we should operate the code.





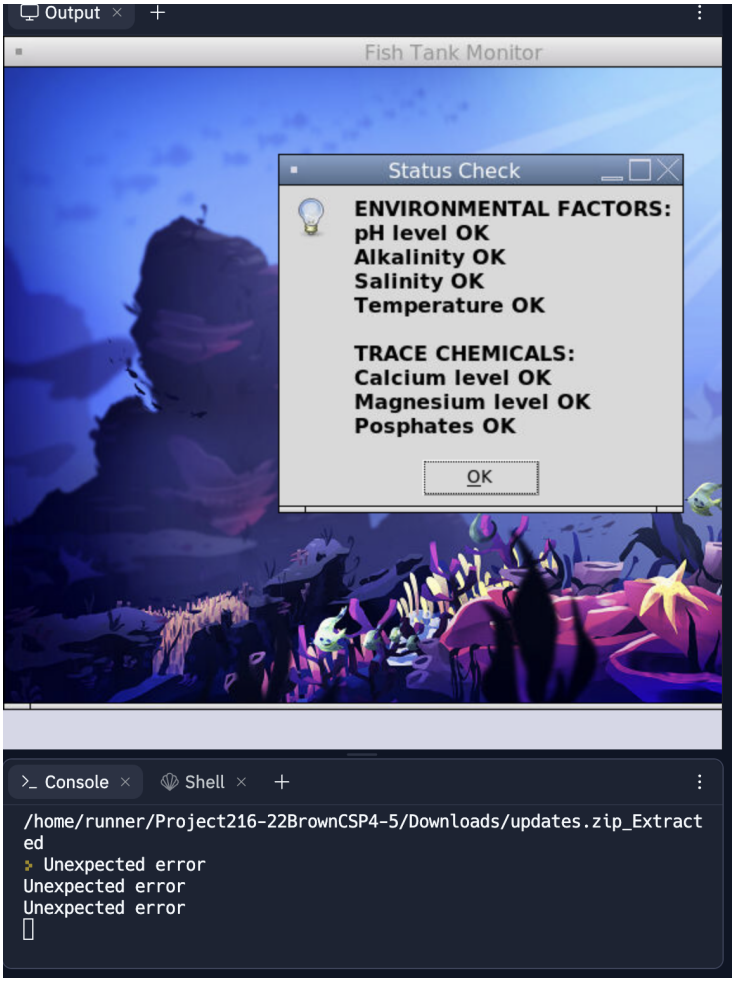


1) We attempted to run the code by importing the FishTank class from fishtank.py and creating a FishTank object, but this resulted in an error because the class’s code referenced a “tank.PNG” file, while in the updates there was only a file named “tank.png”. When we fixed this error, this resulted in a Fish tank prompting us to verify the status.




```
tank_picture = os.path.join(dirname, 'tank.png')
```

2) When we verify the status is working correctly, the resulting prompt shows that all Environmental variables and Trace Chemicals are at OK levels. However, what isn't shown on the output and rather in the terminal is multiple “unexpected errors” ‘s. This tells us that there is an issue inside the program files that is returning an error, yet assuring the fish tank monitor that everything is okay.



2a) When investigating temperature.py, we found that every time, the code would divide by zero, therefore raising an error, yet outputting to the monitor that all was well.

```
  # get multiple temperature readings
  temp_readings = get_temps()
  num_readings = 0

  # sum adds up all items in list
  ave_temp = sum(temp_readings)
  ave_temp = ave_temp / num_readings

  if (ave_temp < temps[0]):
    mesg = "Average temperature too cold!"
  elif (ave_temp > temps[5]):
    mesg = "Average temperature too warm!"

except:
  print("Unexpected error")

return mesg
```

2b) When investigating alkalinity.py: there are a multitude of spelling errors, indicating this code was written without much care for the actual quality, something we doubt Casino Casino to accept. Moreover, we noticed the alkalinity levels would always return an error because when the range function is called, the first value is larger than the second value, so list(range(val1, val2+1)) will just return an empty list. Then when the conditionals try to access the values at indices 0 and 5, an IndexError will occur, which is caught by the except clause to print “Unexpected Error”.

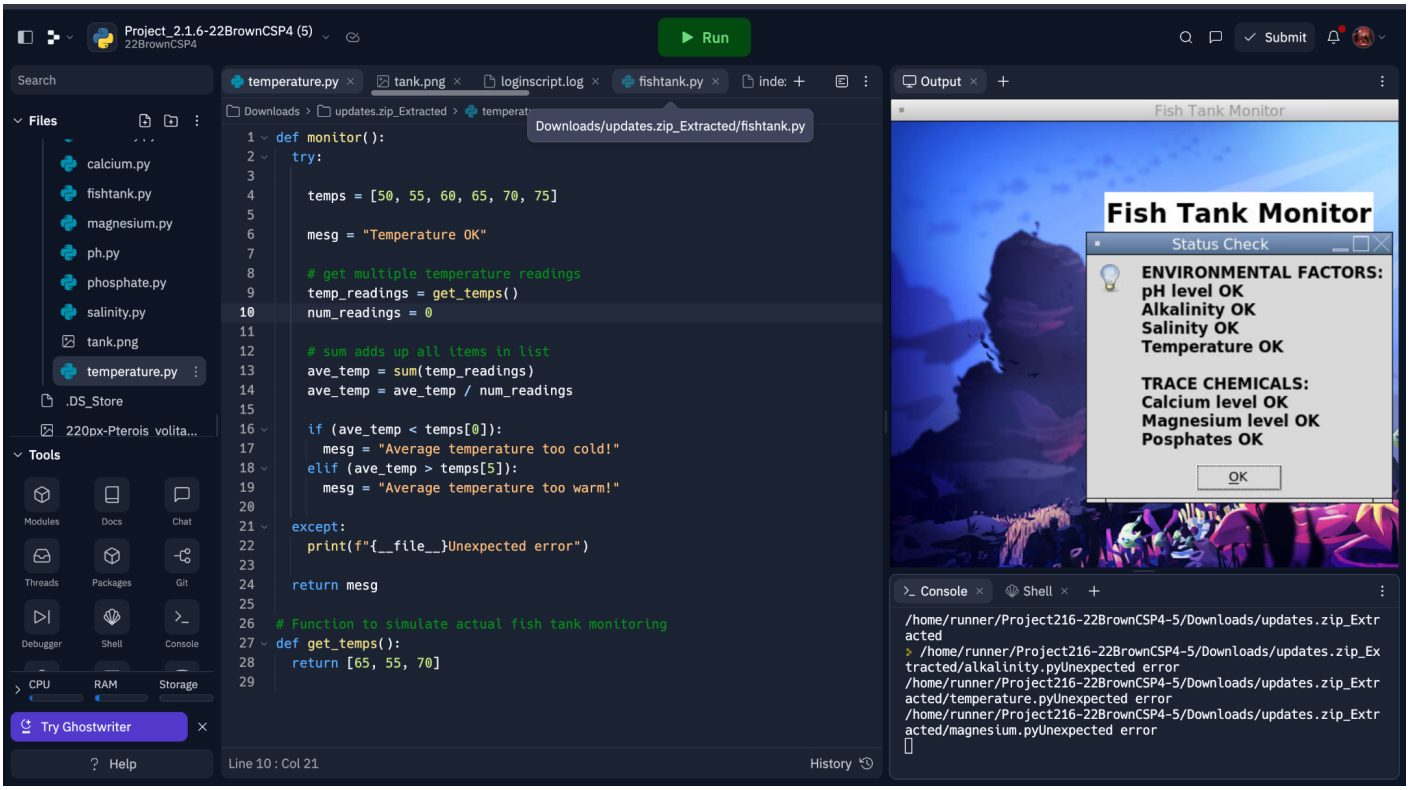
```
try:

  val1 = 17
  val2 = 12

  alkilines = list(range(val1, val2+1))

  current = get_alkalinity()
  mesg = "Alkalinity OK"

  if (current < alkilines[0]):
    mesg = "Alkalinity too low!"
  elif (current > alkilines[5]):
    mesg = "Alkalinity too high!"

except:
  print("Unexpected error")

return mesg
```

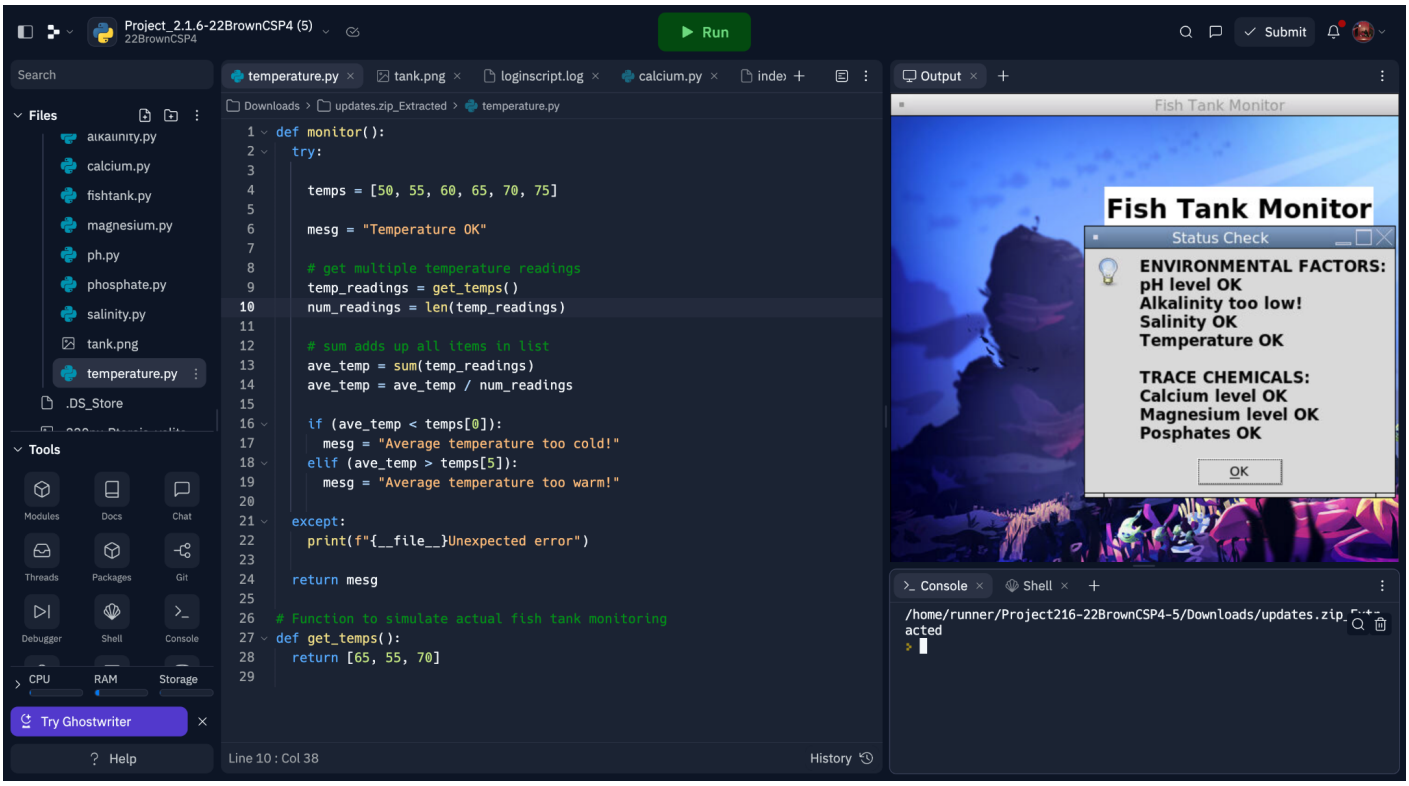
2c) When investigating magnesium.py, we notice that the elif statement calls for mag_levels at position of num_levels, which was conveniently defined as the length of magnesium levels, which happens to be greater than the actual indices of the list.

```
try:

  val1 = 1250
  val2 = 1350

  mag_levels = list(range(val1, val2, 10))

  current = get_magnesium_level()
  mesg = "Magnesium level OK"

  num_levels = len(mag_levels)
  if (current < mag_levels[0]):
    mesg = "Magnesium level too low!"
  elif (current > mag_levels[num_levels]):
    mesg = "Magnesium level too high!"

except:
  print(f"{__file__}Unexpected error")

return mesg
```

2d) We also noticed consistent spelling errors, an unusual occurrence to happen when forming a quality product. Simple errors such as “phosphate” and “alkaline” were made, even though the file name was spelled correctly, indicating that there were some oversights and rushed code portions.

Below, you can find the initial method by which we found the file errors, and the result of us fixing the error. Notice how, though initially Alkalinity was presented as OK, it was, in reality, too low, which could have harmed the fish and the overall tank environment.


Project_2.1.6-22BrownCSP4 (5)
22BrownCSP4
Run
Submit
Search
Files
calcium.py
fishtank.py
magnesium.py
ph.py
phosphate.py
salinity.py
tank.png
temperature.py
.DS_Store
220px-Pterois volita...
Tools
Modules
Docs
Chat
Threads
Packages
Git
Debugger
Shell
Console
CPU
RAM
Storage
Try Ghostwriter
Help
temperature.py
tank.png
loginscript.log
fishtank.py
Downloads
updates.zip_Extracted
Downloads/updates.zip_Extracted/fishtank.py
def monitor():
try:
temps = [50, 55, 60, 65, 70, 75]
mesg = "Temperature OK"
# get multiple temperature readings
temp_readings = get_temps()
num_readings = 0
# sum adds up all items in list
ave_temp = sum(temp_readings)
ave_temp = ave_temp / num_readings
if (ave_temp < temps[0]):
mesg = "Average temperature too cold!"
elif (ave_temp > temps[5]):
mesg = "Average temperature too warm!"
except:
print(f"{__file__}Unexpected error")
return mesg
# Function to simulate actual fish tank monitoring
def get_temps():
return [65, 55, 70]
Line 10 : Col 21
History
Output
Fish Tank Monitor
Fish Tank Monitor
Status Check
ENVIRONMENTAL FACTORS:
pH level OK
Alkalinity OK
Salinity OK
Temperature OK
TRACE CHEMICALS:
Calcium level OK
Magnesium level OK
Posphates OK
OK
Console
Shell
/home/runner/Project216-22BrownCSP4-5/Downloads/updates.zip_Extracted
/home/runner/Project216-22BrownCSP4-5/Downloads/updates.zip_Extracted/alkalinity.pyUnexpected error
/home/runner/Project216-22BrownCSP4-5/Downloads/updates.zip_Extracted/temperature.pyUnexpected error
/home/runner/Project216-22BrownCSP4-5/Downloads/updates.zip_Extracted/magnesium.pyUnexpected error


Project_2.1.6-22BrownCSP4 (5)
22BrownCSP4
Run
Submit
Search
Files
calcium.py
fishtank.py
magnesium.py
ph.py
phosphate.py
salinity.py
tank.png
temperature.py
.DS_Store
Tools
Modules
Docs
Chat
Threads
Packages
Git
Debugger
Shell
Console
CPU
RAM
Storage
Try Ghostwriter
Help
temperature.py
tank.png
loginscript.log
calcium.py
Downloads
updates.zip_Extracted
temperature.py
def monitor():
try:
temps = [50, 55, 60, 65, 70, 75]
mesg = "Temperature OK"
# get multiple temperature readings
temp_readings = get_temps()
num_readings = len(temp_readings)
# sum adds up all items in list
ave_temp = sum(temp_readings)
ave_temp = ave_temp / num_readings
if (ave_temp < temps[0]):
mesg = "Average temperature too cold!"
elif (ave_temp > temps[5]):
mesg = "Average temperature too warm!"
except:
print(f"{__file__}Unexpected error")
return mesg
# Function to simulate actual fish tank monitoring
def get_temps():
return [65, 55, 70]
Line 10 : Col 38
History
Output
Fish Tank Monitor
Fish Tank Monitor
Status Check
ENVIRONMENTAL FACTORS:
pH level OK
Alkalinity too low!
Salinity OK
Temperature OK
TRACE CHEMICALS:
Calcium level OK
Magnesium level OK
Posphates OK
OK
Console
Shell
/home/runner/Project216-22BrownCSP4-5/Downloads/updates.zip_Extracted