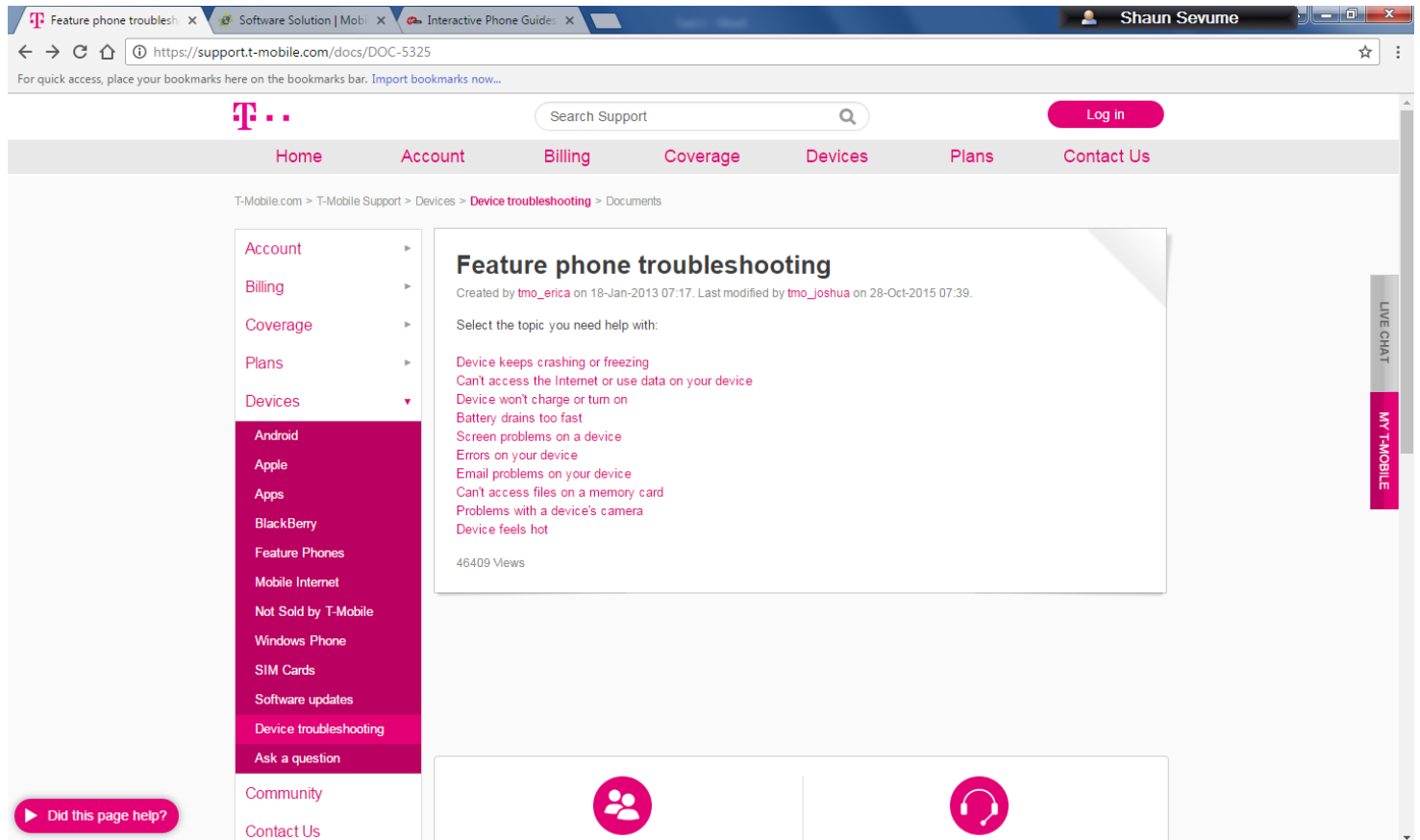


1. Analysis

I'm required to create a troubleshooting program for mobile phones that will be able to identify keywords entered by the user to give them the appropriate help they need.

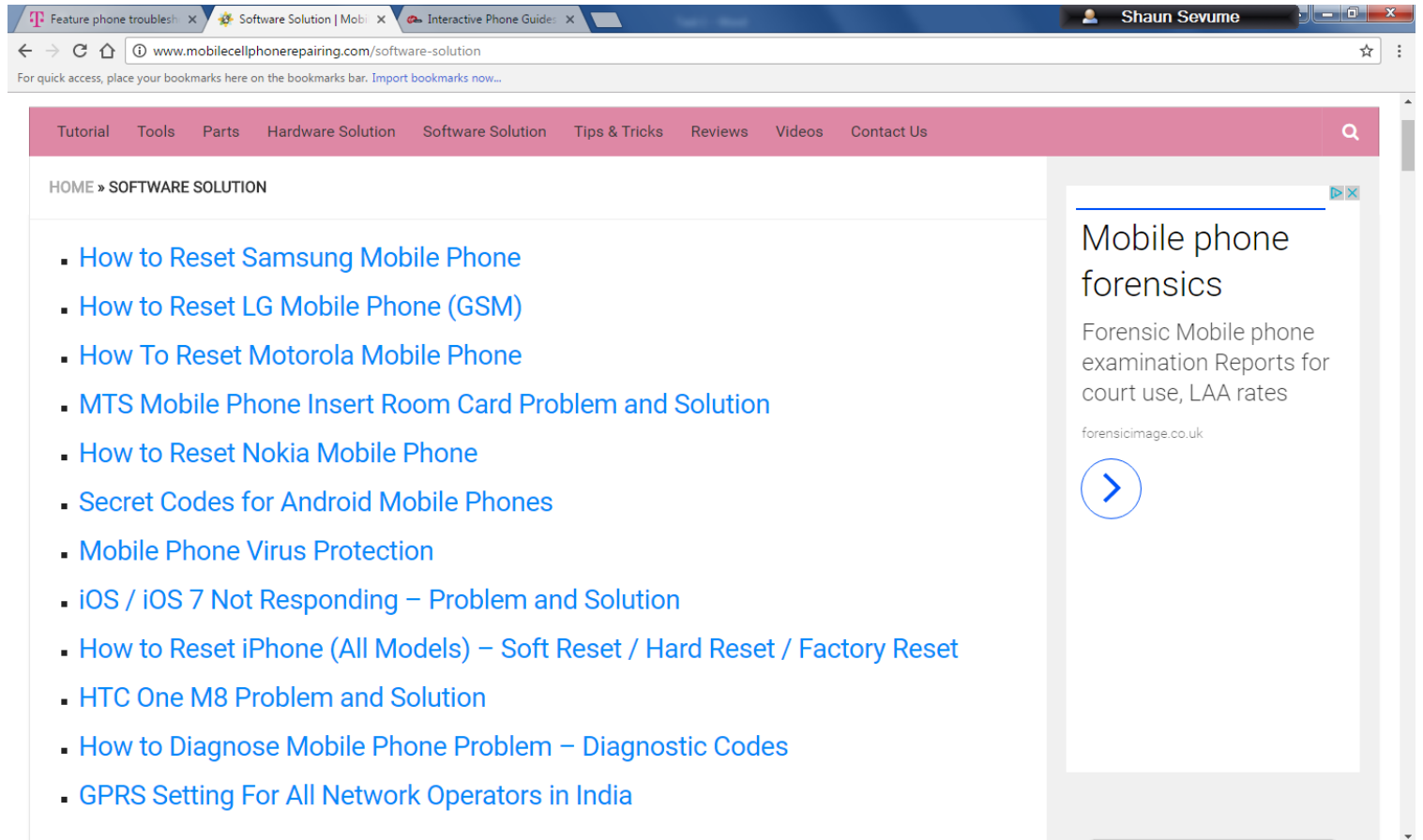
I did some research onto the current mobile troubleshooting programs online, and I found three, all of which were designed in a very similar fashion. In all of them, they required the user to click around to find what they were looking for.

The first website was T-Mobile, a popular mobile network. <https://support.t-mobile.com/docs/DOC-5325>



This site is a good example of what my program should be like. It has clear, broad problems and solutions. There is even an “Ask a question” tab and “Contact us”, so T-Mobile are there to support you if you don't find your answer. Though not every problem is covered, it's easy to add to the list. There's also a search at the top to get around the site quickly. The only downside was the fact that solutions were not device specific, so instructions would be vague examples rather than clear direct ones for the user's particular device/OS.

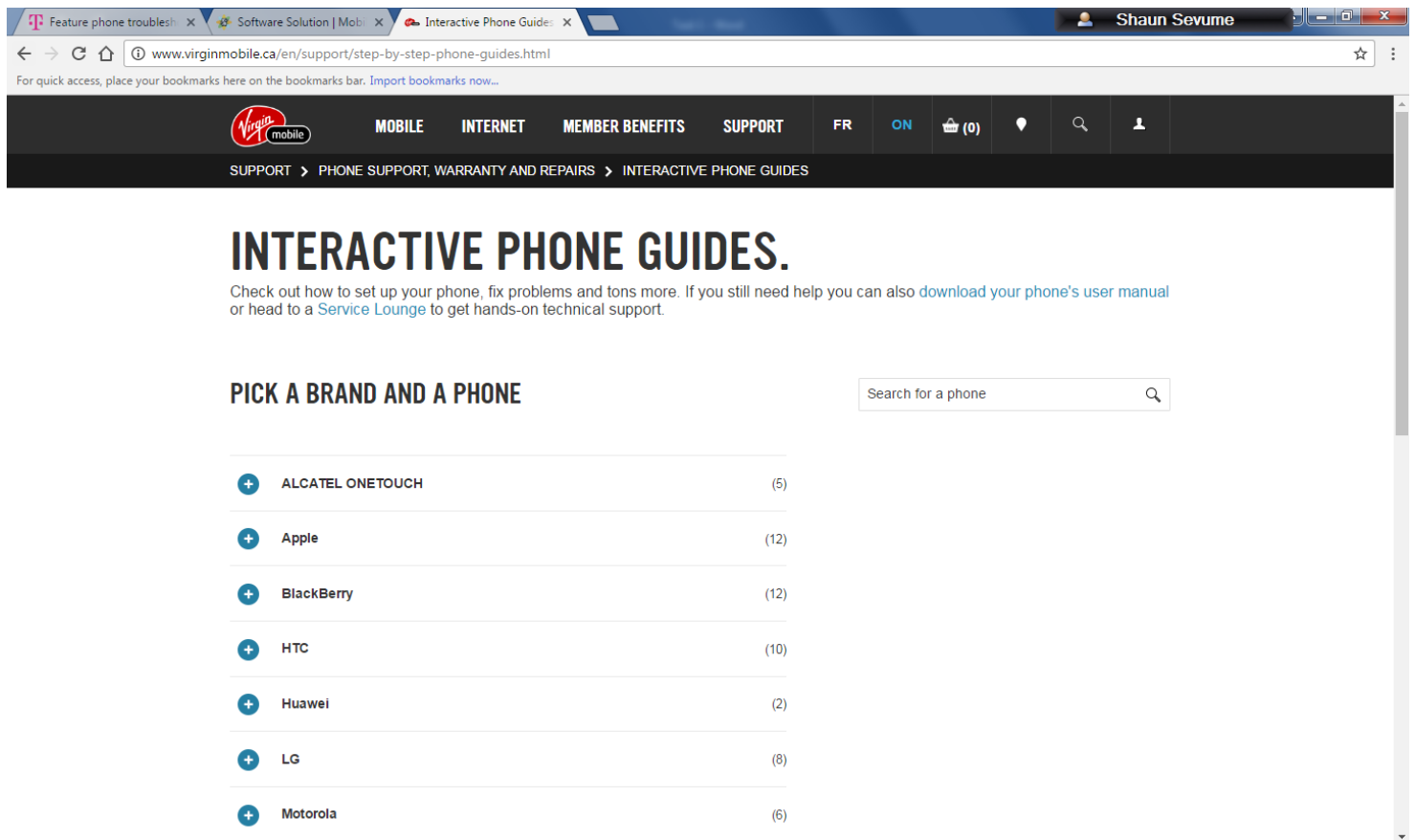
The second website is Mobile phone cell repairing. It seemed to be a simple mobile troubleshooting site. <http://www.mobilecellphonerepairing.com/software-solution>



Similar to the T-Mobile website, this website had clear problems and solutions, and some were even device specific. A search bar and a “Contact us” tab also featured here, but at times user friendliness felt limited, and a lot of reading had to be done before making choices.

The third website was Virgin Mobile, another popular network provider.

<http://www.virginmobile.ca/en/support/step-by-step-phone-guides.html>



Feature phone troubles... X Software Solution | Mobile X Interactive Phone Guides X Shaun Sevume

www.virginmobile.ca/en/support/step-by-step-phone-guides.html

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

Virgin mobile MOBILE INTERNET MEMBER BENEFITS SUPPORT FR ON (0) 🔍 👤

SUPPORT > PHONE SUPPORT, WARRANTY AND REPAIRS > INTERACTIVE PHONE GUIDES

INTERACTIVE PHONE GUIDES.

Check out how to set up your phone, fix problems and tons more. If you still need help you can also [download your phone's user manual](#) or head to a [Service Lounge](#) to get hands-on technical support.

PICK A BRAND AND A PHONE

Search for a phone 🔍

+	ALCATEL ONETOUCH	(5)
+	Apple	(12)
+	BlackBerry	(12)
+	HTC	(10)
+	Huawei	(2)
+	LG	(8)
+	Motorola	(6)

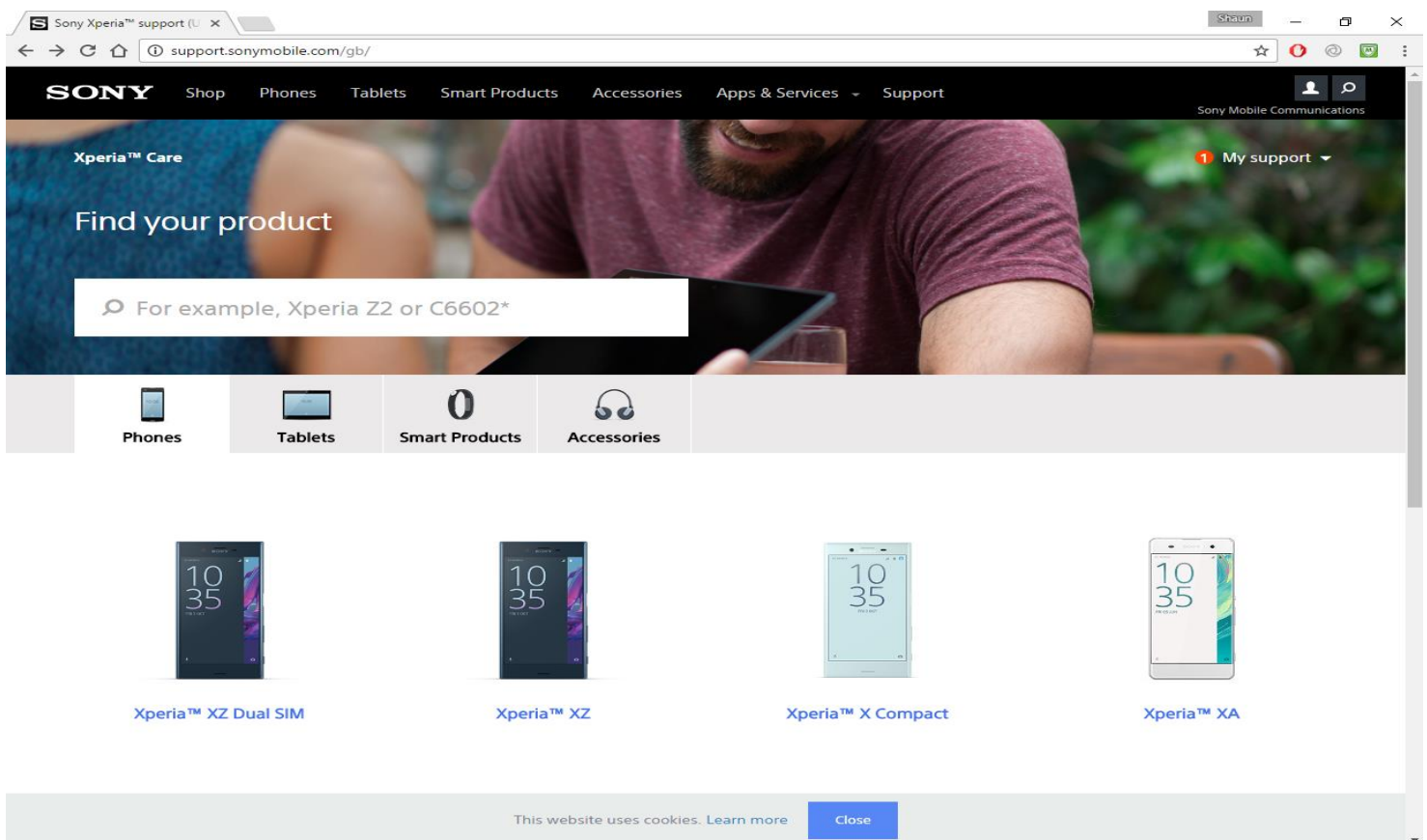
This particular website required you to scroll down until you found your device model, and then click it, which would present a list of options to choose from, which the user of course had to click on. You could search for the make and model, but not all instances were there present. I don't find this very user-friendly, because it requires a lot of input from the user, and they have to find their way around the site to get their answer.

As noted in task 1, these three were not actually programs, but rather a collection of hyperlinks on a site that contain information on specific subjects.

My program must be user friendly and easily understandable, targeted at people of a 12+ age. It must be able to ask the user for their name, and identify keywords correctly. For example, if one user inputs 'broken screen' and another inputs 'cracked screen', the program must be able to identify the keyword 'screen' and give help based on that. If the problem cannot be solved, then the program will try to issue further advice so it doesn't completely leave the user in the dark. It must be able to run on most PC's, as it isn't a resource consuming program. Minimum requirements will be an Intel Pentium processor @500MHz and 256MB of RAM and an operating system of Windows XP. Other components needed include a mouse, a monitor and a keyboard as well as additional components found on a motherboard, such as a hard drive etc. It will be created in python. I did research into possible software like this on the internet, however, I was unable to find one that worked in the way the requirements specified. Like the three websites found in task 1, they all worked on the principal of clicking around to find your solution.

I was able to find a device specific trouble-shooter on the Sony website. It was slightly better, because you could search by your device model, but it still didn't meet all the requirements for task 2.

<http://support.sonymobile.com/gb/>



Search engines like google could act like trouble-shooters, able to source appropriate help, but it didn't actually help you, just pointed you in the right direction, so it doesn't count. In addition, although it is very good at returning relevant results, it isn't able to do things like store your name, etc., which is why I believe my program will be unique. It will also be available to download, so you won't need a constant internet connection to access it.

Success Criteria

- Greet the user with their name
- Ask them about their issue
- Identify certain keywords based on their input, such as 'battery' or 'screen'.
- Give an appropriate solution based on what it's identified and attempt to give further advice should the problem not be resolved.
- Based on the input, give an appropriate output (e.g if the input is 'my phone won't charge', it won't just output something about Wi-Fi).
- Maintain a sense of user-friendliness so that it stands out from other programs.
- Solve display problems
- Solve a phone not being able to charge
- Solve a phone's touch screen problems
- Solve a phone's volume problems
- Solve texting issues
- Solve phones unable to connect to the internet
- Solve phone brightness problems
- Solve phone microphone problems

Test plan

Test no.	Testing	Input (if applicable)	Expected output
1	Does it address the user by their name based on their input?	Shaun	Okay Shaun...
2	Does it wait the required time before carrying out the next function using time.sleep?	Shaun	(After the amount of time specified in the time.sleep function) Okay Shaun...
3	Does it give suggestions of keywords if none are detected?	N/A	Try to enter something with some of these keywords. (A list of some keywords)
4	Does it crash when something unexpected is inputted?	qwerty (anything random)	Error
5	Is my code correctly indented?	N/A	(Indentation) Error
6	Are variables correctly defined?	N/A	(Unidentified variable) Error
7	Is there an else statement in case the if statements are not met?	N/A	If you don't put else, there will be no output if no conditions are met.
8	Does it give appropriate outputs based on input?	(Did this solve your issue?) N	We are sorry we couldn't help you solve your issue.
9	Does it give an appropriate output based on the input?	My phone won't connect to the internet	Here's how to fix internet issues...

10	Does it keep on asking for an input for a specific variable before a condition is met? (Iteration)	(No keywords detected)	Try to enter something with some of these keywords. (A list of some keywords)
----	---	------------------------	---

2. Design

Flowchart

This starts the flowchart

N represents the line number. It varies depending on the solution to be outputted.

There is a welcoming message that is too long to put into the flowchart, so I've just put Welcome message in brackets. The full message can be found in the code.

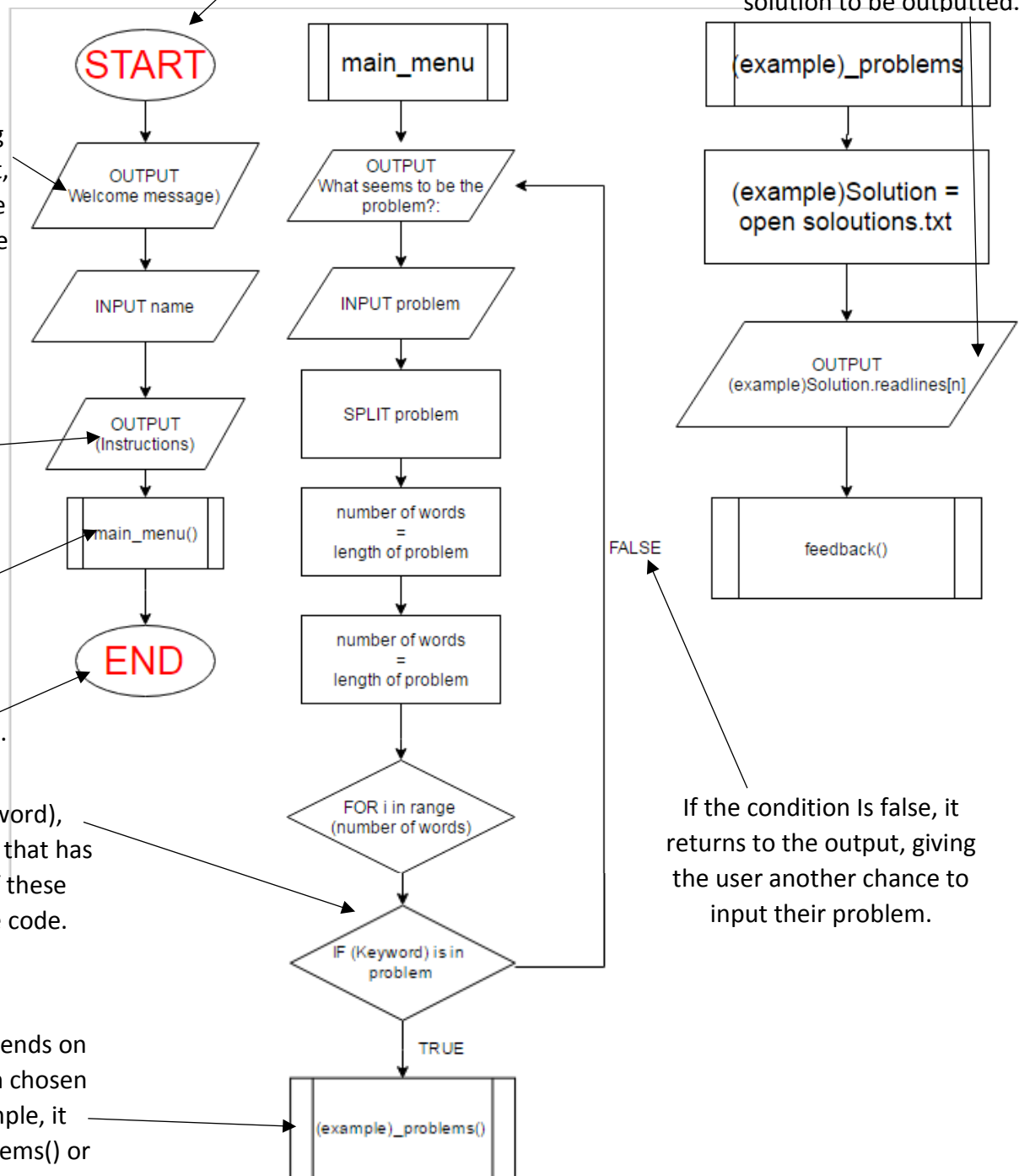
The same applies to the instructions. The full message can be found in the code.

This goes to the next flowchart. In this case, it's the main_menu() one.

This ends the flowchart.

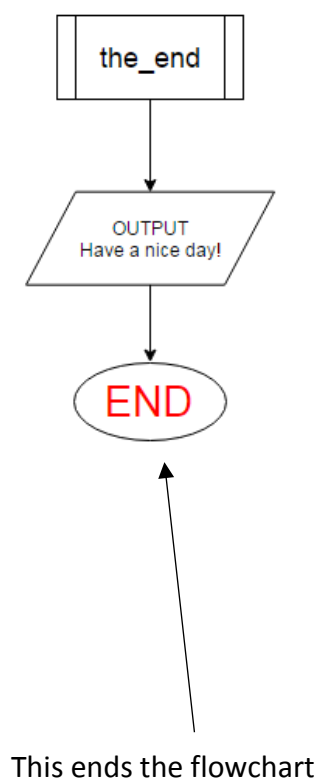
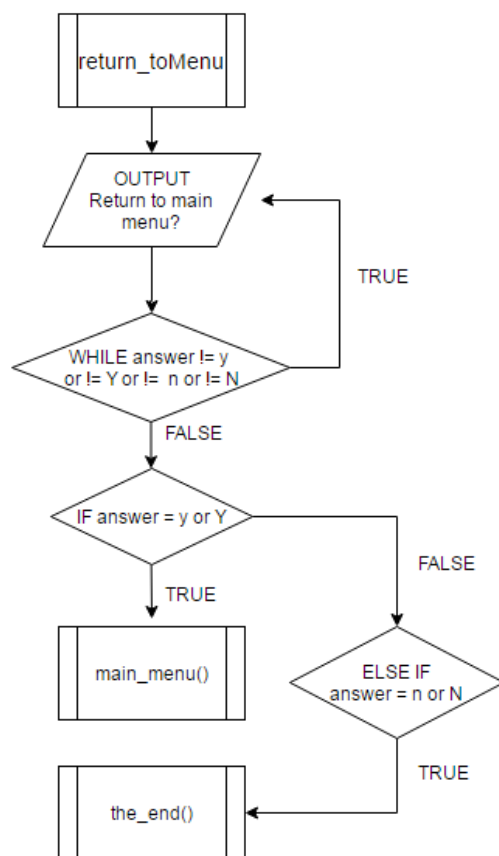
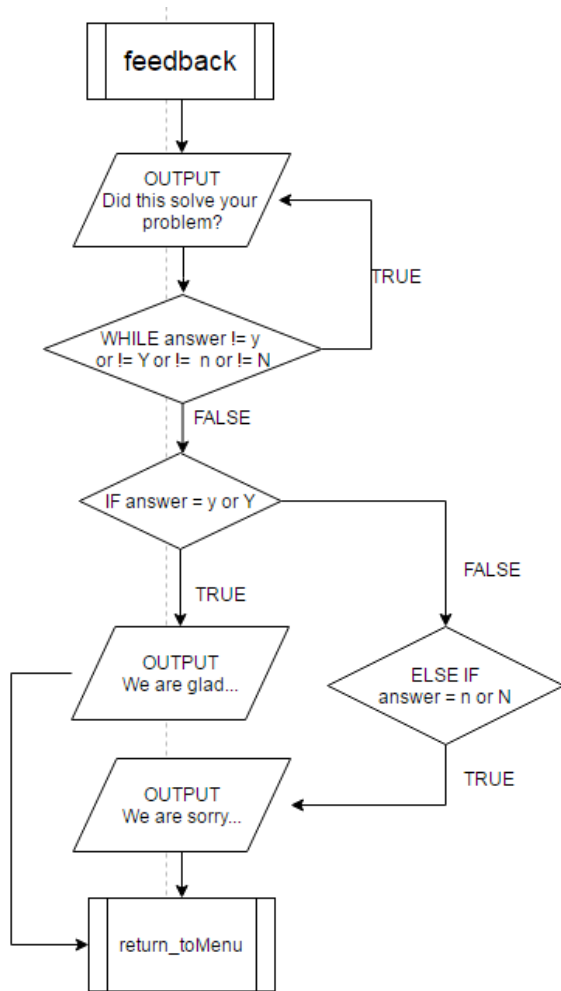
Where it says (keyword), represents a keyword that has been defined. All of these keywords are in the code.

The word example depends on what solution has been chosen for the user. For example, it could be volume_problems() or texting_problems()



FALSE

If the condition is false, it returns to the output, giving the user another chance to input their problem.



Pseudocode

START

IMPORT time

IMPORT re

OUTPUT "(Welcome message)"

OUTPUT "What is your name?"

INPUT name

OUTPUT (Instructions)

FUNCTION MAIN MENU:

 OUTPUT "What is the problem with your device?"

 INPUT Problem

 LOWERCASE Problem

 SPLIT Problem

 Number of words = LENGTH OF Problem

 Counter = 0

 WHILE Counter != 999:

 FOR I IN THE RANGE of Number of words:

 IF Problem = "display" OR "blank" OR "turn" AND "on" THEN

 CALL DISPLAY PROBLEMS

 ELSE IF PROBLEM = "charge" OR "battery" OR "dead" OR "charging" THEN

 CALL BATTERY PROBLEMS

 ELSE IF PROBLEM = "screen" OR "touch" THEN

 CALL TOUCH SCREEN PROBLEMS

 ELSE IF PROBLEM = "volume" OR "sound" OR "speaker" THEN

 CALL VOLUME PROBLEMS

 ELSE IF PROBLEM = "call" OR "ring" OR "dial" THEN

 CALL CALLING PROBLEMS

 ELSE IF PROBLEM = "text" OR "message" OR "SMS" THEN

 CALL TEXTING PROBLEMS

 ELSE IF PROBLEM = "internet" OR "connect" OR "Wi" AND "Fi" OR "web" THEN

 CALL INTERNET PROBLEMS

 ELSE IF PROBLEM = "dim" OR "bright" OR "brightness" THEN

 CALL BRIGHTNESS PROBLEMS

 ELSE IF PROBLEM = "mic" OR "microphone" THEN

 CALL Microphone Problems

```

        END IF
    END FOR
    Counter+1
    IF counter = 999 THEN
        CALL Nothing Defined
    END IF
END WHILE

```

FUNCTION Feedback:

```

    WHILE answer ISN'T y OR n:
        OUTPUT "Did this solve your problem? (y/n)"
        INPUT answer
        LOWERCASE answer
        IF answer = y THEN
            OUTPUT "We are glad..."
        ELSE IF answer = n THEN
            OUTPUT "We are sorry..."
        CALL Return to menu
        END IF
    END WHILE

```

FUNCTION Display Problems:

```

    Solution = OPEN "Solutions.txt", READ
    OUTPUT Line 0 of Solutions.txt
    CALL Feedback

```

FUNCTION Battery Problems:

```

    Solution = OPEN "Solutions.txt", READ
    OUTPUT Line 1 of Solutions.txt
    CALL Feedback

```

FUNCTION Touchscreen Problems:

```

    Solution = OPEN "Solutions.txt", READ
    OUTPUT Line 2 of Solutions.txt
    CALL Feedback

```

FUNCTION Volume Problems:

Solution = OPEN "Solutions.txt", READ

OUTPUT Line 3 of Solutions.txt

CALL Feedback

FUNCTION Calling Problems:

Solution = OPEN "Solutions.txt", READ

OUTPUT Line 4 of Solutions.txt

CALL Feedback

FUNCTION Texting Problems:

Solution = OPEN "Solutions.txt", READ

OUTPUT Line 5 of Solutions.txt

CALL Feedback

FUNCTION Internet Problems:

Solution = OPEN "Solutions.txt", READ

OUTPUT Line 6 of Solutions.txt

CALL Feedback

FUNCTION Brightness Problems:

Solution = OPEN "Solutions.txt", READ

OUTPUT Line 7 of Solutions.txt

CALL Feedback

FUNCTION Microphone Problems:

Solution = OPEN "Solutions.txt", READ

OUTPUT Line 8 of Solutions.txt

CALL Feedback

FUNCTION THE END:

OUTPUT Have a nice day!

EXIT

```

FUNCTION Return to menu:
    WHILE return ISN'T y OR n:
        OUTPUT return to main menu?
        INPUT return
        LOWERCASE return
        IF return = y THEN
            OUTPUT MAIN MENU
        ELSE IF RETURN = n THEN
            CALL THE END
        END IF
    END WHILE

CALL Main Menu

END

```

Variable and Validation table

Variable Name	Type	Validation	Description
Name	String	Any character	When prompted, the user input will be assigned to the variable name, that will be used later on in the program.
Feedback	String	y/Y or n/N	The user is asked if their problem has been resolved beforehand. The answer is then assigned to the variable feedback and an appropriate output will be displayed depending on if the user inputted y or n.
(example)Solution	(assigned to) function	N/A	As explained with the flowchart, the word example represents the problem based on the user input. For instance, it could be batterySolution or displaySolution. This variable is assigned to the 'open' function that opens the "solutions.txt" file that has all the solutions and read it. It is then used with the print function to display the proposed solution. It is also designed to only read specific lines.
Return1	String	y/Y or n/N	Similar to the variable feedback. The user is asked if they wish to return to the main menu and the answer is assigned to the variable return. If y/Y then the user is returned to the main menu. If n/N, then the program terminates.
Problem	String	Characters within the range (a-z, A-Z)	When the user inputs their problem, it is assigned to this variable, which is then converted into lowercase to match the case the keywords

			were defined in by using the. lower function(). Afterwards, it is split by using the .split() function and is assigned to another variable in split form.
User_problem	String	Characters within the range (a-z, A-Z)	This is the variable that the other variable problem is assigned to after it has been converted into lowercase and split.
numberOfWords	(assigned to) Integer	(Presumably) numbers with no decimal point larger than 0.	The len function is used to find out the length (how many characters there are) of the user_problem variable, which is the split and lowercased version of the variable problem, which is the user input. Once the length is found out, it is assigned to this variable.
Validation	Integer	Numbers with no decimal point in between 0 and 999	It acts like a counter for how many times the FOR loop has run which identifies keywords. After the set amount of times has elapsed, python will execute the specific command to ask the user to try and input something again.
nothingDefined	Function	If no keywords are detected in the user's input.	This def function is run if no keywords are detected in the user's input. It prints a message letting them know nothing could be found before taking them to the return_toMenu() function.
feedback	Function	After a solution has been printed, this function is called.	The user is asked if their problem has been resolved beforehand. Depending on the input, they may be taken to the return_toMenu function or given an apology first.
(example)_problems	Function	If the keywords relating to whatever 'example' may be are detected in the user's input, the appropriate function is run. For example, if the keyword relating to 'display' is detected, then the function display_problems will be called.	There is one def function for each possible solution. In each of the function, they open the "solutions.txt" text file and read the appropriate line in accordance to the user's input. For example, if the display solutions were on line 1, then line 1 of "solutions.txt" would be read and outputted.
the_end	Function	If the user inputs n when asked if they want to return to the menu	This is the function that terminates the program, using the SystemExit command.
return_toMenu	Function	Called if the user answers "n/N" when asked if their problem was solved or after the nothingDefined function is called.	Contains a while loop that keeps asking the user if they want to return to the menu until they enter y or n.
main_menu	Function	Is run first, but can be called back on later in the	As the main part of the code, it contains the piece of code that asks the user what their

		code, such as within the return_toMenu function.	problem is, before checking the input against pre-defined keywords and looking for a match. If any are found, the appropriate function will be called. If no keyword are found, then the nothingDefined function is called.
--	--	--	---

3. Development

This is a code made from a previous lesson that uses the IN function.

```
for x in range (4):  
    n = int(input("Enter your number: "))  
    print("Number you've entered ", n)  
    print("Loop number is ", x)
```

I tried to make another code that used the IN function, but was adapted to the task I was meant to complete. Here is what I came up with.

```
name = input("h-h-h-hai")  
  
if "hi" in name:  
    print("OHHH")  
else:  
    print("lel it didnt work")
```

This was the output:

```
h-h-h-haihi  
OHHH  
>>> |
```

This meant that it worked, so I tried to add more keywords to it using the OR function.

```
name = input("h-h-h-hai")  
  
if "hi" or "yo" or "SNM" in name:  
    print("OHHH")  
else:  
    print("lel it didnt work")
```

This was the result:

```
h-h-h-haiyo  
OHHH  
>>>  
===== RESTART:  
h-h-h-haiSNM  
OHHH  
>>>
```

Though the code worked, it couldn't be used for task 2, because it outputted "OHH" no matter what was entered, so it was inefficient. I tried to use the DEF function instead.

```
def keywords(red, yellow, pink, green):
    word = input("colour ")

    if word in keywords(red, yellow, pink, green):
        print("DAMN")
    else:
        print("hah")

print(keywords())
```

This was the output.

```
Traceback (most recent call last):
  File "C:/Users/SShau/Documents/d.py", line 9, in <module>
    print(keywords())
TypeError: keywords() missing 4 required positional arguments: 'red', 'yellow', 'pink', and 'green'
>>> |
```

It didn't work.

I made another code where I used a variable to define all the keywords first and if the variable "word" (which was user input) was in the variable keywords, it would print "meh".

```
keywords = "red", "p", "lel", "hai"

word = input("word ")

if word in keywords:
    print("meh")
else:
    print("nah")
```

These were the results.

word red	word p	word hai	word lel
meh	meh	meh	meh
>>>	>>>	>>>	>>>

Then I tested out the ELSE statement and this was the result.

```
word no
nah
>>>
```

I changed the code to fit task 2 better by adding phrases that users would be likely to enter.

```
keywords = "screen wont work", "screen not showing", "display wont work", "display not showing"

problem = input("problem? ")

if problem in keywords:
    print("OHHHH IT WORKED")
else:
    print("wow ok")
```


These were the results.

```
problem? screen wont work      problem? screen not showing      problem? display wont work
OHHHH IT WORKED                OHHHH IT WORKED                OHHHH IT WORKED
>>>                            >>>                            >>>

                                problem? display not showing
                                OHHHH IT WORKED
                                >>>
```

And this was the result when I tested the ELSE statement.

```
problem? display is broken
wow ok
```

When I was making my code, I used the DEF function which would be executed if the user wished to exit the program. It used the raise SystemExit function again, like in the previous task.

```
def the_end():
    print("Have a nice day!")
    raise SystemExit
```

I also added a code for returning to the main menu in a def function, because it had been repeated a lot in task 1, so doing this made it more efficient.

```
def return_tomenu():
    return1 = 0

    while return1 != "y" or return1 != "Y" or return1 != "n" or return1 != "N":

        return1 = input("Return to main menu?(y/n)")

        if return1 == "y" or return1 == "Y":
            main_menu()

        elif return1 == "n" or return1 == "N":
            the_end()
```

The solution I had come up with that used the IN function was actually quite inefficient, as it required you to define all of the possible phrases, not actually picking out the specific words. This meant that I wasn't actually achieving what I had to do in task 2 which was identify keywords based on user inputs, not identify pre-set phrases based on user input. It also didn't use text files as required in the success criteria.

```
def main_menu():
    problem = input("Go ahead. What seems to be your problem? ")
    user_problem = problem.split()
    for i in range(len(user_problem)):
        if user_problem[i] == ("display" or user_problem[i] == "blank" or user_problem[i] == "screen" or user_problem[i] == "turn on"):
```

However, Python didn't identify any keywords, so the outcome wasn't what was expected.

```
Go ahead. What seems to be your problem? my screen is broken
Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'. Remember that our program is currently limited in words it can understand.
Go ahead. What seems to be your problem? my screen wont turn on
Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'. Remember that our program is currently limited in words it can understand.
Go ahead. What seems to be your problem?
```

The program was supposed to ask if the problem was right based on user input. But it kept skipping to the else statement instead. This was the code.

```
problem = input("Go ahead. What seems to be your problem? ")
user_problem = problem.split()
for i in range(len(user_problem)):
    if user_problem[i] == ("display" or "blank" or "screen" or "turn on"):

        time.sleep(1)
        feedback = 0

        while feedback != "y" or feedback != "n":
            feedback = input("Is the problem to do with your screen not turning on? (y/n) ")
```

It was supposed to identify the word 'display' and print the appropriate help. Instead it executed this ELSE statement.

```
else:
    print("Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'.
    main_menu()
```

This was the output.

```
Go ahead. What seems to be your problem? my display is broken
Try and enter something that's maybe a little shorter, such as
an understand.
Go ahead. What seems to be your problem?
```

I changed the way the code was structured to try and make it more efficient. The old way relied heavily on repeated if statements.

Old code:

```
while feedback != "y" or feedback != "n":
    feedback = input("Is the problem to do with your screen not turning on? (y/n) ")

if feedback == "y":
    displaySolution = open("solutions.txt", "r")
    print(displaySolution.readlines()[1])
    time.sleep(5)
    option = 0

    while option != "y" or option1 != "Y" or option1 != "n" or option1 != "N":
        option = input("Did this solve your problem? (y/n)")

        if option == "y" or option == "Y": #By putting OR in the code, we make sure the code will still execute cc
            print("We are glad to have helped you solve your issue," + name + "!") #Based on what the user entered

        elif option == "n" or option == "N":
            displaySolution = open("solutions.txt", "r")
            print(displaySolution.read()[2]) #Further help will be given to the user if their problem is still not
            time.sleep(5)
            return_tomenu()
```

I changed it to make it more based off of DEF functions so that it was more efficient with less repeated code.

New code:

```
def display_problems():
    displaySolution = open("solutions.txt", "r")
    print(displaySolution.readlines()[1])
    return_tomenu()

def battery_problems():
    batterySolution = open("solutions.txt", "r")
    print(batterySolution.readlines()[2])
    return_tomenu()

def touchScreen_problems():
    touchScreenSolution = open("solutions.txt", "r")
    print(touchScreenSolution.readlines()[3])
    return_tomenu()

def volume_problems():
    volumeSolution = open("solutions.txt", "r")
    print(volumeSolution.readlines()[4])
    return_tomenu()

def calling_problems():
    callingSolution = open("solutions.txt", "r")
    print(callingSolution.readlines()[5])
    return_tomenu()

def texting_problems():
    textingSolution = open("solutions.txt", "r")
    print(textingSolution.readlines()[6])
    return_tomenu()

def internet_problems():
    internetSolution = open("solutions.txt", "r")
    print(internetSolution.readlines()[7])
    return_tomenu()

def dim_problems():
    dimSolution = open("solutions.txt", "r")
    print(dimSolution.readlines()[8])
    return_tomenu()

def mic_problems():
    micSolution = open("solutions.txt", "r")
    print(micSolution.readlines()[10])
    return_tomenu()

def the_end():
    print("Have a nice day!")
    raise SystemExit
```

The use of the DEF function meant I didn't have to repeat the code, and it would be easier to identify where the problem was if there was an error.

The code that I had used before didn't work, properly as explained above, so I had to make a few changes.

Before:

```
problem = input("Go ahead. What seems to be your problem? ")
user_problem = problem.split()
for i in range(len(user_problem)):
    if user_problem[i] == ("display" or user_problem[i]=="blank" or user_problem[i]=="screen" or user_problem[i]=="turn on"):
        display_problems()

else:
    print("Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'".
    main_menu())
```

After:

```
def main_menu():
    problem = input("Go ahead. What seems to be your problem? ")
    user_problem = problem.split()
    numberOfWords = len(user_problem)
    for i in range (numberOfWords):
        if user_problem[i] == "display" or "blank" or "screen" or "turn on":
            display_problems()
```

This made the code broken in more places. It was now bypassing what was specified in the DEF functions (open and print what was in the txt file) and went straight to the DEF function that asked to return to the main menu. Also, if nothing relevant was inputted, it would still show the solution for a broken display which was unwanted, showing that python wasn't even identifying keywords, just executing the next line of code. These were some outputs I got when experimenting with the code. They were all unwanted.

```
Go ahead. What seems to be your problem? my display is blank
Check your phone for any damage. Severe damage may prevent it from working,
he case, you must go to the phone manufacturer's shop or a repair shop to (

Return to main menu?(y/n)y
Go ahead. What seems to be your problem? the blank screen is
Check your phone for any damage. Severe damage may prevent it from working,
he case, you must go to the phone manufacturer's shop or a repair shop to (

Return to main menu?(y/n)y
Go ahead. What seems to be your problem? phone is blank
Check your phone for any damage. Severe damage may prevent it from working,
he case, you must go to the phone manufacturer's shop or a repair shop to (

Return to main menu?(y/n)
Return to main menu?(y/n)
Return to main menu?(y/n)
Return to main menu?(y/n)y
Go ahead. What seems to be your problem?
Return to main menu?(y/n)y
Go ahead. What seems to be your problem?
Return to main menu?(y/n)y
Go ahead. What seems to be your problem?
Return to main menu?(y/n)y
Go ahead. What seems to be your problem?
Return to main menu?(y/n)
Return to main menu?(y/n)
Return to main menu?(y/n)
Return to main menu?(y/n)y
Go ahead. What seems to be your problem?
Return to main menu?(y/n)y
Go ahead. What seems to be your problem? fdf
Check your phone for any damage. Severe damage may prevent it from working,
he case, you must go to the phone manufacturer's shop or a repair shop to (

Return to main menu?(y/n)y
Go ahead. What seems to be your problem?
```

I put the ELSE statement in triple speech marks that kept it as text, but made sure that python didn't execute it as code. This was to hold it until an alternative solution was found.

```
"""else:
    print("Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'. Remember that our
    main_menu()
    """
```

Still the problem remained, so I changed my code a little more. I added back user_problem[i] before every word python had to identify.

Before:

```
def main_menu():
    problem = input("Go ahead. What seems to be your problem? ")
    user_problem = problem.split()
    numberOfWords = len(user_problem)
    for i in range (numberOfWords):
        if user_problem[i] == "display" or "blank" or "screen" or "turn on":
            display_problems()
```

After:

```
problem = input("Go ahead. What seems to be your problem? ")
user_problem = problem.split()
numberOfWords = len(user_problem)
for i in range (numberOfWords):
    if user_problem[i] == "display" or user_problem[i] == "blank" or user_problem[i] == "screen" or user_problem[i] == "turn on":
        display_problems()
```

This was the outcome:

```
Go ahead. What seems to be your problem? screen is broken
Check your phone for any damage. Severe damage may prevent it from working, (e.g dropping it from a
he case, you must go to the phone manufacturer's shop or a repair shop to get your phone looked at.

Return to main menu?(y/n)y
Go ahead. What seems to be your problem? screen wont work
Check your phone for any damage. Severe damage may prevent it from working, (e.g dropping it from a
he case, you must go to the phone manufacturer's shop or a repair shop to get your phone looked at.

Return to main menu?(y/n)y
Go ahead. What seems to be your problem? the phone screen wont work
Check your phone for any damage. Severe damage may prevent it from working, (e.g dropping it from a
he case, you must go to the phone manufacturer's shop or a repair shop to get your phone looked at.

Return to main menu?(y/n)y
Go ahead. What seems to be your problem? the display is broken
Check your phone for any damage. Severe damage may prevent it from working, (e.g dropping it from a
he case, you must go to the phone manufacturer's shop or a repair shop to get your phone looked at.

Return to main menu?(y/n)y
Go ahead. What seems to be your problem? the phone is blank
Check your phone for any damage. Severe damage may prevent it from working, (e.g dropping it from a
he case, you must go to the phone manufacturer's shop or a repair shop to get your phone looked at.
```

Python didn't recognize the keyword "turn on" and didn't respond when it was inputted.

```
Go ahead. What seems to be your problem? my phone wont turn on
>>> |
```

I changed the code from this:

```
if user_problem[i] == "display" or user_problem[i] == "blank" or user_problem[i] == "turn on":
    display_problems()
```

To this. I replaced the keyword "turn on" with "turn" and "on", since it would mean python would only identify it as a keyword if both those words were in a string.

```
if user_problem[i] == "display" or user_problem[i] == "blank" or user_problem[i] == "turn" and "on":
    display_problems()
```

Here was the output, which showed that the change worked.

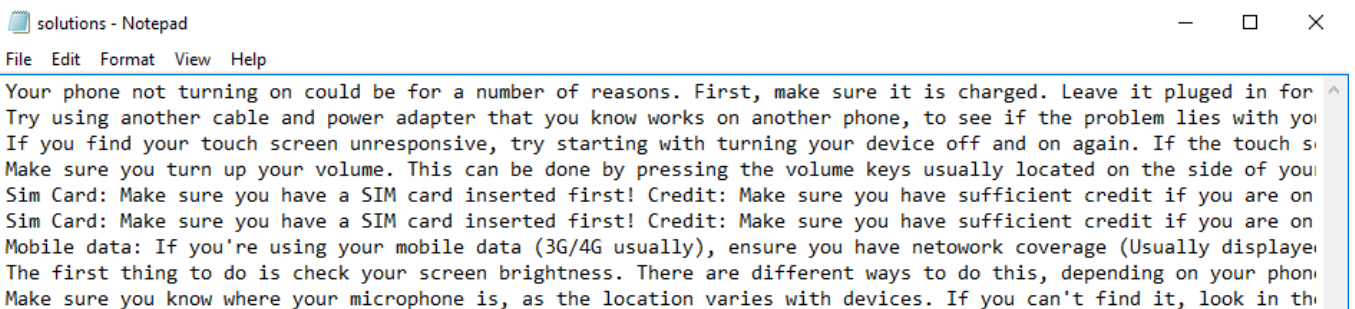
```
Go ahead. What seems to be your problem? my phone wont turn on
Check your phone for any damage. Severe damage may prevent it from
u must go to the phone manufacturer's shop or a repair shop to get

Return to main menu?(y/n)|
```

This is a section of my code that was for problems relating to the phone display.

```
def display_problems():
    displaySolution = open("solutions.txt", "r")
    time.sleep(1)
    print(displaySolution.readlines()[0])
    return_tomenu()
```

The readlines() function was added later, and it greatly improved efficiency, as all the solutions could now be in one txt file. The function only read specific lines. In this case it was line 0, which would be the first line in the txt file as shown below.



```
solutions - Notepad
File Edit Format View Help
Your phone not turning on could be for a number of reasons. First, make sure it is charged. Leave it plugged in for
Try using another cable and power adapter that you know works on another phone, to see if the problem lies with you
If you find your touch screen unresponsive, try starting with turning your device off and on again. If the touch s
Make sure you turn up your volume. This can be done by pressing the volume keys usually located on the side of you
Sim Card: Make sure you have a SIM card inserted first! Credit: Make sure you have sufficient credit if you are on
Sim Card: Make sure you have a SIM card inserted first! Credit: Make sure you have sufficient credit if you are on
Mobile data: If you're using your mobile data (3G/4G usually), ensure you have network coverage (Usually displaye
The first thing to do is check your screen brightness. There are different ways to do this, depending on your phon
Make sure you know where your microphone is, as the location varies with devices. If you can't find it, look in th
```

I noticed that if you entered your problem in upper case, python didn't recognize it as a keyword, since they had been defined in the lower case. This was my code and output at the time.

```
def main_menu():
    problem = input("Go ahead. What seems to be your problem? ")
    user_problem = problem.split()
    numberOfWords = len(user_problem)

    Go ahead. What seems to be your problem? THE BATTERY IS BROKEN
    >>> |
```

To fix this, I added the .lower() function in my code, that meant that when the user inputted something, python would first convert it to lower case before splitting it and trying to identify keywords, so that it matched the case the keywords were defined in. This was my altered code and the output to show that it worked.

```
def main_menu():
    problem = input("Go ahead. What seems to be your problem? ")
    problem_lower = problem.lower()
    user_problem = problem_lower.split()
    numberOfWords = len(user_problem)

    Go ahead. What seems to be your problem? THE BATTERY IS BROKEN
    Try using another cable and power adapter that you know works or
    arges after this, then it was your charger that was the problem.
    . check to see if there's any dust or things obstructing it, and
    re is most likley a problem with your charging port. You will ha

    Did this solve your problem? (y/n)|
```


As previously mentioned before in the development, I had held an ELSE statement in triple quotation marks. This was because even if the user entered a defined keyword, python wouldn't recognize it as one unless it was the first one in the string, meaning that it made the program less flexible, due to the amount of ways that users could word a problem. However, after much trial and error, I was able to find a solution. What I first did was introduce a new variable called Validation, which was to act like a counter. By adding this, I thought that python would keep adding 1 to the variable every time a keyword wasn't found and re-run the for loop. When the loop had been run 999 times (which may look like a big number, but is relatively quick in python considering that the pc's I was using were well over 999Hz or calculations per second) and failed to find a keyword, it would go to the function nothing_defined() which printed a message to try again before restarting the main_menu() function. This was the first try of the code.

```
def main_menu():
    Validation = 0
    problem = input("Go ahead. What seems to be your problem? ")
    problem_lower = problem.lower()
    user_problem = problem_lower.split()
    numberOfWords = len(user_problem)
    |
    for i in range(numberOfWords):
        if user_problem[i] == "display" or user_problem[i] == "blank" or user_problem[i] == "turn" and "on" or user_problem[i] == "screen":
            display_problems()

        elif user_problem[i] == "charge" or user_problem[i] == "battery" or user_problem[i] == "dead" or user_problem[i] == "charging" or user_problem[i] == "microphone":
            battery_problems()

        elif user_problem[i] == "touch" and "screen" or user_problem[i] == "touch":
            touchScreen_problems()

        elif user_problem[i] == "volume" or user_problem[i] == "sound" or user_problem[i] == "speaker":
            volume_problems()

        elif user_problem[i] == "call" or user_problem[i] == "ring" or user_problem[i] == "calls":
            calling_problems()

        elif user_problem[i] == "text" or user_problem[i] == "message" or user_problem[i] == "SMS":
            texting_problems()

        elif user_problem[i] == "internet" or user_problem[i] == "connect" or user_problem[i] == "wi" and "fi" or user_problem[i] == "web":
            internet_problems()

        elif user_problem[i] == "dim" or user_problem[i] == "bright" or user_problem[i] == "brightness":
            dim_problems()

        elif user_problem[i] == "mic" or user_problem[i] == "microphone":
            mic_problems()

    Validation +=1

    if Validation == 999:
        nothingDefined()
```

However, it didn't work in python because the FOR loop wasn't repeating.. Then, after doing some more research on the internet, I decided to embed the FOR loop in a WHILE loop, which forced the for loop to repeat itself until the counter reached 999. This was the improved version of the code with the output.

```
def main_menu():
    Validation = 0
    problem = input("Go ahead. What seems to be your problem? ")
    problem_lower = problem.lower()
    user_problem = problem_lower.split()
    numberOfWords = len(user_problem)
    while Validation != 999:
        for i in range(numberOfWords):
            if user_problem[i] == "display" or user_problem[i] == "blank" or user_problem[i] == "turn" and "on" or user_problem[i] == "screen":
                display_problems()

            elif user_problem[i] == "charge" or user_problem[i] == "battery" or user_problem[i] == "dead" or user_problem[i] == "charging" or user_problem[i] == "screen":
                battery_problems()

            elif user_problem[i] == "touch" and "screen" or user_problem[i] == "touch":
                touchScreen_problems()

            elif user_problem[i] == "volume" or user_problem[i] == "sound" or user_problem[i] == "speaker":
                volume_problems()

            elif user_problem[i] == "call" or user_problem[i] == "ring" or user_problem[i] == "calls":
                calling_problems()

            elif user_problem[i] == "text" or user_problem[i] == "message" or user_problem[i] == "SMS":
                texting_problems()

            elif user_problem[i] == "internet" or user_problem[i] == "connect" or user_problem[i] == "wi" and "fi" or user_problem[i] == "web":
                internet_problems()

            elif user_problem[i] == "dim" or user_problem[i] == "bright" or user_problem[i] == "brightness":
                dim_problems()

            elif user_problem[i] == "mic" or user_problem[i] == "microphone":
                mic_problems()

        Validation +=1

    if Validation == 999:
        nothingDefined()
```

Go ahead. What seems to be your problem? Chicken nuggets
 Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'. Remember that our program is currently limited in the words it can understand.

Go ahead. What seems to be your problem? i dont know
 Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'. Remember that our program is currently limited in the words it can understand.

Go ahead. What seems to be your problem? something
 Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'. Remember that our program is currently limited in the words it can understand.

Go ahead. What seems to be your problem? well, my battery doesn't seem to charge
 Try using another cable and power adapter that you know works on another phone, to see if the problem lies with your phone or your charger. If your phone charges after this, then it was your charger that was the problem. If the phone isn't charging, this could mean there's something wrong with the charging port. Check to see if there's any dust or things obstructing it, and make sure the charger fits in properly(not hanging loosely) and your battery works too. There is most likely a problem with your charging port. You will have to take your phone to the manufacturer to get them to sort it out.

From the output, you can see that this made the code work. Unlike the ELSE statement, the code worked wherever the keyword was in the sentence.

When looking back over my code, I found a way to improve it. Like with task 1, I didn't know how to force text onto a new line at the time of creation, so as a substitute, I had done this:

```
print("")  
print("Before we begin, what is your name?")
```

Python would print an empty line in a separate function, before actually printing the text. To make the code more efficient, I added the `\n` function again.

Improved code:

```
print("\nBefore we begin, what is your name?")
```

Again, like with task 1, I also implemented it around other areas of my code to improve the overall quality in terms of text spacing and appearance.

Before:

```
-----  
Welcome to the multicellular Troubleshooting program. 2.0! This new version comes with a brand new feature: Keyword identification!  
  
You can now enter your problem and our program will do its best to identify keywords to relay help right back to you!  
This program is still in its early stages, so solutions are limited, but we hope to expand soon!  
  
Before we begin, what is your name?  
My name is: Shaun  
Ah, Shaun.  
Okay Shaun, this time there won't be a list of options to select from. Simply enter your problem and we'll try our best to diagnose it!  
  
Since our program is again in its early stage, we kindly ask you to keep your inputs in lower case, with no spaces at the end and no extra punctuation such as  
exclamation marks. This is so that the program runs smoothy. We also ask that you refrain from using contractions such as 'wont' or 'isnt', just for the time being  
until we further develop our program to understand more user inputs.  
  
Go ahead. What seems to be your problem? Do not enter multiple problems at once!Your mum  
Try and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'. Remember that our program is currently limited in the words it c  
an understand.  
Return to main menu?(y/n)y  
Go ahead. What seems to be your problem? Do not enter multiple problems at once!My battery wntcharge  
Try using another cable and power adapter that you know works on another phone, to see if the problem lies with your phone or your charger. If your phone charges after this, th  
en it was your charger that was the problem. If the phone isn't charging, this could mean there's something wrong with the charging port. check to see if there's any dust or th  
ings obstructing it, and make sure the charger fits in properly(not hanging loosley) and your battery works too. There is most likley a problem with your charging port. You wil  
l have to take your phone to the manufacturer to get them to sort it out.  
  
Did this solve your problem? (y/n)y  
We are glad to have helped you solve your issue,Shaun!  
Return to main menu?(y/n)n  
Have a nice day!  
>>>
```

After:

```
Welcome to the multicellular Troubleshooting program. 2.0! This new version comes with a brand new feature: Keyword identification!  
  
You can now enter your problem and our program will do its best to identify keywords to relay help right back to you!  
This program is still in its early stages, so solutions are limited, but we hope to expand soon!  
  
Before we begin, what is your name?  
My name is: Shaun  
  
Okay Shaun, this time there won't be a list of options to select from. Simply enter your problem and we'll try our best to diagnose it!  
  
Since our program is again in its early stage, we kindly ask you to keep your inputs in lower case, with no spaces at the end and no extra punctuation such as  
exclamation marks. This is so that the program runs smoothy. We also ask that you refrain from using contractions such as 'wont' or 'isnt', just for the time being  
until we further develop our program to understand more user inputs.  
  
Go ahead. What seems to be your problem? Do not enter multiple problems at once! My battery wont charge  
  
Try using another cable and power adapter that you know works on another phone, to see if the problem lies with your phone or your charger. If your phone charges after  
this, then it was your charger that was the problem. If the phone isn't charging, this could mean there's something wrong with the charging port. check to see if there  
's any dust or things obstructing it, and make sure the charger fits in properly(not hanging loosley) and your battery works too. There is most likley a problem with y  
our charging port. You will have to take your phone to the manufacturer to get them to sort it out.  
  
Did this solve your problem? (y/n) y  
We are glad to have helped you solve your issue,Shaun!  
  
Return to main menu?(y/n) n  
  
Have a nice day!  
>>> |
```

I used the exact same inputs, but the second screenshot shows just how much neater everything looked with the `\n` function.

Like in task 1, I implemented the .lower() function into my while loops to make them more efficient.

Before:

```
while feedback != "y" or feedback != "Y" or feedback != "n" or feedback != "N": #The while loop here means that until the answer is
    feedback = input("Did this solve your problem? (y/n) ")

if feedback == "y" or feedback == "Y":
    print("We are glad to have helped you solve your issue," + name + "!")
    return_toMenu()#Here, the return_toMenu() function is called. This is on line 93.

elif feedback == "n" or feedback == "N": #After the first IF statement, each subsequent one after that must be ELIF instead.
    print("We are sorry we couldn't help you solve your issue. Please try and contact your manufacturer for further advice.")
    return_toMenu()#See line 28.
```

After:

```
while feedback != "y" or feedback != "n": #The while loop here means that until the answer is y,Y or n,N, it will keep repeating.
    feedback = input("Did this solve your problem? (y/n) ")
    feedback_lower = feedback.lower()

if feedback_lower == "y":
    print("We are glad to have helped you solve your issue," + name + "!")
    return_toMenu()#Here, the return_toMenu() function is called. This is on line 93.

elif feedback_lower == "n": #After the first IF statement, each subsequent one after that must be ELIF instead.
    print("We are sorry we couldn't help you solve your issue. Please try and contact your manufacturer for further advice.")
    return_toMenu()#See line 28.
```

Techniques Used

Technique/Keyword	What does it do?	How is it used in my code?
def	Used to define a function, or a piece of code. You can use this to split your code into blocks so that instead of repeating code over and over again, you simply call the appropriate function which will have the code written in. It means you only have to write the code once, and makes it easier to identify which part of your code has a problem (should any arise).	It is used to split the code into blocks, making it easier to identify problems and not needing to repeat code. For example, the the_end() function is used to exit the program. It only has to be called instead of being repeated.
Import	Enables the use of specific techniques, depending on what you choose to import	I used import time so I can use the time.sleep technique.
Time	Imported with the import function, which is the system time.	I use it with the time.sleep function, where you can specify how long you want the program to be idle for before executing the next line.
Print	Outputs whatever is specified in the speech marks and brackets.	I use the print function on various occasions, mostly to display instructions to the user.
Input	Assigns the user's input to a variable.	An example of when I have used this is when the user is asked their name. What they enter is assigned to the variable called name, made possible because of the input keyword.
While	Used to create a WHILE loop, where a piece of code is constantly repeated until the condition for the loop is either made true or false (depending on how you structured the loop).	I used a WHILE loop when the program was asking the user if their problem was solved or not. The condition was the input not being y or n, and whilst this is true, the while loop is run. Once the condition becomes false, the loop is broken, and the program can proceed.

For	The exact same as a while loop, except you specify how many times the loop is run.	When identifying keywords, I used a for loop, and using the range technique/keyword, I specified for the for loop to run the same amount of times as the amount of words the user inputted.
Range	Used to specify a range of numbers. For example, if you entered 0, 5 in the brackets that follow the keyword, then the range would be from 0-4 but not including 5.	I used it with my for loop to specify how many times it should run. Since there was a variable assigned to the number of words the user had inputted called numberOfWords, I specified that as a range, since it was technically still an integer, because of its numerical value.
Len	Used to get the length/number of items in an object. For example "hello world" would have a value of 11, due to there being 11 characters (spaces included).	I used it to get the length of the variable user problem, which was the lowercased and split version of the user's input, when asked what their problem was. The length would then be assigned to the variable numberOfWords
Or	This is <u>usually</u> (but not limited to) with if statement. It gives python an alternative condition to satisfy that applies to the same variable.	I used it with the variable called feedback. The condition was whether the user's input was y OR n. The or function makes sure that both conditions are checked before the program carries on running.
If /elif	This is a statement that has a condition, which needs to be satisfied first. The condition could be a variable being equal to something, or the user's input matching a specified keyword. Once the condition is satisfied, the rest of the if statement is	An example of me using an if statement is when I put 'IF' return1 = y. This means that if the variable called return has a value of y (based on user input), then whatever was specified to do if the condition was true is done. In this case, it returns to the main menu.

	run. If the condition isn't satisfied, then it will usually run the next if statement (if there are any), run the else statement (if present) or just terminate if nothing is specified for what the program has to do should the statement not be satisfied.	
Raise	The raise keyword forces an exception to occur. There are many different exceptions, but the one I forced to occur was SystemExit.	I used it to raise the SystemExit exception, which terminated the program, when the user was done.
SystemExit	It is an exception, that is used to call the sys.exit() function – which (as the name suggests) exits/terminates the program.	Using the raise function, I raise this exception which calls the sys.exit() function, which terminates the program.
Open	This command tells python to open another file. What file to open is specified in the brackets that follow.	After the user input their problem and the problem is identified, the program opens the txt file I created called 'solutions.txt', that had all the solutions written down.
.readlines	Used with the open technique/keyword. After a txt file is open, you can specify what lines the program needs to read (and output), in case you only want a specific part to be outputted.	Because all my solutions are in one txt file, I had to use this function to specify what lines of the txt file to read, so that the appropriate solution was outputted.
.lower	Converts user's input into lowercase by adding .lower to a variable that is assigned to the user's input.	I used it to make the user's input lowercase when they were asked what their problem was, so that it was the same case as they keywords I had added to identify the problem, making it easier for them to be identified as keywords.

.split	Splits the user's input by spaces so that it can be checked word by word until a keyword is matched, or until the counter reaches 999.	I used it after I used the .lower function, to split the user's input so that it could be checked word by word either until a match was found with one of the keywords, or unless the counter that controlled how many times the for loop ran reached its limit, 999.
Variables	Variables are objects with no value. They can be called literally anything (apart from reserved keywords such as import) and they can be assigned to things such as numbers, letters, different character and even user inputs. You can manipulate variables to make python perform certain things. For example, if you define a variable as 1, you could make an if statement to perhaps print something if that variable was equal to 1.	They are used all over my code. For example, the user's name is assigned to the variable 'name', the user's problem is assigned to the variable 'problem' and so on.
+	Used for adding things. If it's adding integers, then a new answer will be printed. However, if it's adding strings, then it will print them together.	When printing instructions to the user, the program first outputs "okay" followed by the user's name. in the code, I used print("okay" + name), which would print "okay" plus whatever had been assigned to the variable name, done by the user's input.
=/==/!=	Used to make something the same (equal) as something. If you are directly assigning a variable to a string value, then you use 2 equals signs instead of one. An exclamation mark and an equals sign is the same as saying "not equal to".	When declaring variables, such as name, I used the input function, so that the user's input would be equal (=) to the variable name. So if the input was 'Shaun' then name would = Shaun. I used != in my while loops. The condition was that while the user's input != (was not equal to) "y/Y" or "n/N", the question would be repeatedly asked.

#	Used to make annotations in the code.	I used it to annotate my code. Annotated text appears in red.
Camel case	A style of naming variables, whereby the first word is lowercase, and the second word begins with a capital. For example: userName is in camel case.	I used it when naming some of my variables, such as return_toMenu.
\n	Adds a new line. Usually used within the print function.	I used it to space out my code and make my code look neater.

Development review

I was required to create a troubleshooting program for mobile phones that will be able to identify keywords entered by the user to give them the appropriate help they need. The solutions would be stored on a txt file that would be opened, and read out in the code to the user. It had to be user friendly and easily understandable, targeted at people of a 12+ age. One of the requirements for the program was to ask the user's name and greet them by outputting their name. To solve this, I used the input() keyword, which allows the user to input a response to a question or an instruction given to them. Their input is then assigned to the variable called name, so every time the program has to address them by their name, we can use the variable to output whatever they inputted as their name. The code and output are below.

```
name = input("My name is: ")#This will be where the user enters their name. This variable will be re-used in the code, and not only does it avoid repetition of asking for their
print("\nOkay " + name + ", this time there won't be a list of options to select from. Simply enter your problem and we'll try our best to diagnose it!\n\nSince our program is :
```

```
Before we begin, what is your name?
My name is: Shaun
```

```
Okay Shaun, this time there won't be a list of options to select from. Simply enter your problem and we'll try our best to diagnose it!
```

To solve the keyword identification, I used a for loop, but because of the issue of it only running once, I had to put the for loop in the while loop, to force it to run 999 times. I used the .lower() function to make the user's input lowercase, to match the case of the code and make keyword identification easier. If any of the keywords were detected in the user's input, made lowercase for easier recognition by python, they would be taken to the appropriate place. It also had to account for variation in user input, which it did because it only picked out keywords, so it wouldn't matter what was entered as long as one keyword was entered. If they entered 'my phone wont charge' for example, then the keyword 'charge' would be identified.

```
while Validation != 999: #As long as the counter is below 999, this loop is run.
    for i in range(numberOfWords): #'i' can represent any number. The range function specifies what range to look in, which is in this case the amount of words in the user':
        if user_problem[i] == "display" or user_problem[i] == "blank" or user_problem[i] == "turn" and "on" or user_problem[i] == "screen":#If any of these keywords are det
            display_problems()#Calls the display_problems() function on line 35.

        elif user_problem[i] == "charge" or user_problem[i] == "battery" or user_problem[i] == "dead" or user_problem[i] == "charging" or user_problem[i] == "die":#See line
            battery_problems()#Calls the battery_problems() function on line 41.

        elif user_problem[i] == "touch" and "screen" or user_problem[i] == "touch":#See line 117.
            touchScreen_problems()#Calls the touchScreen_problems() function on line 47.

        elif user_problem[i] == "volume" or user_problem[i] == "sound" or user_problem[i] == "speaker":#See line 117.
            volume_problems()#Calls the volume_problems() function on line 53.

        elif user_problem[i] == "call" or user_problem[i] == "ring" or user_problem[i] == "calls":#See line 117.
            calling_problems()#Calls the calling_problems() function on line 59.

        elif user_problem[i] == "text" or user_problem[i] == "message" or user_problem[i] == "SMS":#See line 117.
            texting_problems()#Calls the texting_problems() function on line 65.

        elif user_problem[i] == "internet" or user_problem[i] == "connect" or user_problem[i] == "wi" and "fi" or user_problem[i] == "web":#See line 117.
            internet_problems()#Calls the internet_problems() function on line 71.

        elif user_problem[i] == "dim" or user_problem[i] == "bright" or user_problem[i] == "brightness":#See line 117.
            dim_problems()#Calls the dim_problems() function on line 78.

        elif user_problem[i] == "mic" or user_problem[i] == "microphone":#See line 117.
            mic_problems()#Calls the mic_problems() function on line 84.

    Validation +=1 #Once all the if statements are checked, it adds one to the counter and runs the loop again, this time searching for keywords one space further along.

if Validation == 999: #This if statement is run once the for loop has run 999 times (which is very quick to a computer). By this stage, we can be sure no keywords have b
    nothingDefined()#Calls the nothinDefined() function on line 15
```

```
Go ahead. What seems to be your problem? Do not enter multiple problems at once! my phone wont charge
```

```
Try using another cable and power adapter that you know works on another phone, to see if the problem lies with your phone or your charger. If your phone charges after this, th
en it was your charger that was the problem. If the phone isn't charging, this could mean there's something wrong with the charging port. check to see if there's any dust or th
ings obstructing it, and make sure the charger fits in properly(not hanging loosley) and your battery works too. There is most likley a problem with your charging port. You wil
l have to take your phone to the manufacturer to get them to sort it out.
```

I used def functions, so that I could simply call them when needed. I had a def function for each solution, so if someone had battery problems, the def function for battery problems would be called, and the txt file would be open with the 'open' and 'r' keywords, and then the specific line would be read out with the .readlines() function that contained solutions for battery problems.

```
def battery_problems():#See line 15.  
    time.sleep(1)#See line 8.  
    batterySolution = open("solutions.txt", "r")#See line 36.  
    print("\n" + batterySolution.readlines()[1])#See line 38.  
    feedback()#See line 39.
```

I also added def functions that were there for navigation around the code to improve efficiency, so that it didn't have to be restarted all the time. For example, there was a function that contained a while loop which asked the user whether they wanted to return to the menu or not. If they chose yes, then the function for the main menu would be called, defined at the part of the code where the user had to input their problem. If, however they chose no, then another function would be called, one in charge of terminating the program.

```
Return to main menu?(y/n) n
```

I called it the_end() and all it did was raise the exception SystemExit and basically terminate the program. Making this function avoided me having to input this command on every part of my code where I wanted it to terminate.

```
def the_end():#See line 15.  
    time.sleep(1)#See line 8.  
    print("\nHave a nice day!")  
    raise SystemExit #This terminates the program properly, so it isn't left running with nothing to display and nothing to be inputted.
```

```
Return to main menu?(y/n) n  
  
Have a nice day!  
>>> |
```

Code

```
#Centre number: 10854
#Candidate number: 7116
#Name: Shaun Sevume

import time #This is necessary so that I can use the time.sleep function.
import re #A module that contains regular expressions and operations.
print("Welcome to the multicellular Troubleshooting program. 2.0! This new version comes with a brand new feature: Keyword identification!\n\nYou can now enter your problem and c
time.sleep(2)#This makes the program wait before executing the next line. How long it waits depends on the number specified in the brackets, which is in seconds.
print("\nBefore we begin, what is your name?")#The print function is used to output text to the user.
time.sleep(1)#See line 8
name = input("My name is: ")#This will be where the user enters their name. This variable will be re-used in the code, and not only does it avoid repetition of asking for their
print("\nOkay " + name + ", this time there won't be a list of options to select from. Simply enter your problem and we'll try our best to diagnose it!\n\nSince our program is c
time.sleep(3)#See line 8.

def nothingDefined():#This is where a function is defined, to avoid repetition in the code. This way, you simply have to call this function when needed, instead of rewriting it
print("\nTry and enter something that's maybe a little shorter, such as 'phone not turning on' or 'device not charging'. Remember that our program is currently limited in th
return_toMenu()#Here, the return_toMenu() function is called.

def feedback():#See line 15.
feedback = 0#The variable you're using in the while loop must be declared prior to the loop, or else you get an error saying that the variable hasn't been declared. For that
while feedback != "y" or feedback != "n":#The while loop here means that until the answer is y or n, it will keep repeating.

    feedback = input("Did this solve your problem? (y/n) ")#This is where the user gets to input an answer. Since it only accepts the inputs y and n, it will keep asking the
    feedback_lower = feedback.lower()#This converts the user's input into lowercase, so that it can match the case the if statements were written in.

    if feedback_lower == "y":#This statement will be run if the variable feedback_lower has a value of "y" (given by user input).
        print("We are glad to have helped you solve your issue," + name + "!")#See line 9.
        return_toMenu()#Here, the return_toMenu() function is called. This is on line 94.

    elif feedback_lower == "n": #After the first IF statement, each subsequent one after that must be ELIF instead.
        print("We are sorry we couldn't help you solve your issue. Please try and contact your manufacturer for further advice.")#See line 9.
        return_toMenu()#See line 28.

def display_problems():#See line 15.
displaySolution = open("solutions.txt", "r")#Here it opens the txt file where the solutions are, with "r" standing for "read".
time.sleep(1)#See line 8.
print("\n" + displaySolution.readlines()[0])#Since it is all in one file, this specifies what lines to read, which will contain the appropriate solution.
feedback() #Calls the feedback() function on line 19.

def battery_problems():#See line 15.
time.sleep(1)#See line 8.
batterySolution = open("solutions.txt", "r")#See line 36.
print("\n" + batterySolution.readlines()[1])#See line 38.
feedback()#See line 39.

def touchScreen_problems():#See line 15.
time.sleep(1)#See line 8.
touchScreenSolution = open("solutions.txt", "r")#See line 36.
print("\n" + touchScreenSolution.readlines()[2])#See line 38.
feedback()#See line 39.

def volume_problems():#See line 15.
time.sleep(1)#See line 8.
volumeSolution = open("solutions.txt", "r")#See line 36.
print("\n" + volumeSolution.readlines()[3])#See line 38.
feedback()#See line 39.

def calling_problems():#See line 15.
time.sleep(1)#See line 8.
callingSolution = open("solutions.txt", "r")#See line 36.
print("\n" + callingSolution.readlines()[4])#Since it is all in one file, this specifies what lines to read.
feedback()#See line 39.

def texting_problems():#See line 15.
time.sleep(1)#See line 8.
textingSolution = open("solutions.txt", "r")#See line 36.
print("\n" + textingSolution.readlines()[5])#See line 38.
feedback()#See line 39.

def internet_problems():#See line 15.
time.sleep(1)#See line 8.
internetSolution = open("solutions.txt", "r")#See line 36.
print("\n" + internetSolution.readlines()[6])#See line 38.
feedback()#See line 39.

def dim_problems():#See line 15.
time.sleep(1)#See line 8.
dimSolution = open("solutions.txt", "r")#See line 36.
print("\n" + dimSolution.readlines()[7])#See line 38.
feedback()#See line 39.

def mic_problems():#See line 15.
time.sleep(1)#See line 8.
micSolution = open("solutions.txt", "r")#See line 36.
print("\n" + micSolution.readlines()[8])#See line 38.
feedback()#See line 39.

def the_end():#See line 15.
time.sleep(1)#See line 8.
print("\nHave a nice day!")
raise SystemExit #This terminates the program properly, so it isn't left running with nothing to display and nothing to be inputted.
```

```

def return_toMenu():#See line 15.
    time.sleep(1)#See line 8.
    return1 = 0 #Before putting a variable in a while loop, this was necessary so that it was at least defined beforehand to prevent variable errors.

    while return1 != "y" or return1 != "n":#In this while loop, it will keep asking the user to return to the main menu until a valid input is entered.

        return1 = input("\nReturn to main menu?(y/n) ")#See line 24.
        return1_lower = return1.lower()#See line 25.

        if return1_lower == "y":#See line 27
            main_menu()#Calls the main_menu() function on line 109.

        elif return1_lower == "n":#See line 31
            the_end()#Calls the the_end() function on line 89.

def main_menu():#See line 15.
    Validation = 0 #This is a counter for the loop.
    problem = input("Go ahead. What seems to be your problem? Do not enter multiple problems at once! ")#Here, the user has to input their problem. It is then assigned to the variable problem.
    problem_lower = problem.lower()#See line 25.
    user_problem = problem_lower.split() #The .split function splits the words in each variable by spaces, to better identify keywords.
    numberOfWords = len(user_problem) #The len function counts how many characters are in something.
    while Validation != 999: #As long as the counter is below 999, this loop is run.
        for i in range(numberOfWords): #'i' can represent any number. The range function specifies what range to look in, which is in this case the amount of words in the user's problem.
            if user_problem[i] == "display" or user_problem[i] == "blank" or user_problem[i] == "turn" and "on" or user_problem[i] == "screen":#If any of these keywords are detected, the display_problems() function is called on line 35.
                display_problems()#Calls the display_problems() function on line 35.

            elif user_problem[i] == "charge" or user_problem[i] == "battery" or user_problem[i] == "dead" or user_problem[i] == "charging" or user_problem[i] == "die":#See line 41.
                battery_problems()#Calls the battery_problems() function on line 41.

            elif user_problem[i] == "touch" and "screen" or user_problem[i] == "touch":#See line 117.
                touchScreen_problems()#Calls the touchScreen_problems() function on line 47.

            elif user_problem[i] == "volume" or user_problem[i] == "sound" or user_problem[i] == "speaker":#See line 117.
                volume_problems()#Calls the volume_problems() function on line 53.

            elif user_problem[i] == "call" or user_problem[i] == "ring" or user_problem[i] == "calls":#See line 117.
                calling_problems()#Calls the calling_problems() function on line 59.

            elif user_problem[i] == "text" or user_problem[i] == "message" or user_problem[i] == "SMS":#See line 117.
                texting_problems()#Calls the texting_problems() function on line 65.

            elif user_problem[i] == "internet" or user_problem[i] == "connect" or user_problem[i] == "wi" and "fi" or user_problem[i] == "web":#See line 117.
                internet_problems()#Calls the internet_problems() function on line 71.

            elif user_problem[i] == "dim" or user_problem[i] == "bright" or user_problem[i] == "brightness":#See line 117.
                dim_problems()#Calls the dim_problems() function on line 78.

            elif user_problem[i] == "mic" or user_problem[i] == "microphone":#See line 117.
                mic_problems()#Calls the mic_problems() function on line 84.

        Validation +=1 #Once all the if statements are checked, it adds one to the counter and runs the loop again, this time searching for keywords one space further along.

    if Validation == 999: #This if statement is run once the for loop has run 999 times (which is very quick to a computer). By this stage, we can be sure no keywords have been found.
        nothingDefined()#Calls the nothinDefined() function on line 15

main_menu() #This is important to starting the code. Although all the def functions have been defined by this point, none have been referenced, and so calling one here starts the code.

```

solutions - Notepad

File Edit Format View Help

Your phone not turning on could be for a number of reasons. First, make sure it is charged. Leave it plugged in for at least 5-10 minutes before trying to turn it on. Try using another cable and power adapter that you know works on another phone, to see if the problem lies with your phone or your charger. If your phone charges but is unresponsive, try starting with turning your device off and on again. If the touch screen only unresponsive in certain places, make sure you turn up your volume. This can be done by pressing the volume keys usually located on the side of your device. If not, try going to settings and turning up your volume. Sim Card: Make sure you have a SIM card inserted first! Credit: Make sure you have sufficient credit if you are on a pay as-you-go plan or you have sufficient credit. Mobile data: If you're using your mobile data (3G/4G usually), ensure you have network coverage (Usually displayed in a top corner of your phone) and that it is turned on. The first thing to do is check your screen brightness. There are different ways to do this, depending on your phone. For example, on an iPhone, you swipe up from the bottom of the screen to access the control center, then tap on brightness. Make sure you know where your microphone is, as the location varies with devices. If you can't find it, look in the manual that should've come with your phone.

4. Testing

Test no.	Testing	Code to be tested	Input, if applicable	Expected Output	Actual
1	Does it address the user by their name based on their input?	name = input("Name: ")	Shaun	Ah, Shaun. Such a wonderful name.	My name is: Shaun Ah, Shaun. Such a wonderful name.
2	Does it wait the required time before carrying out the next function using time.sleep?	return1 = input("Return to main menu?(y/n)")	y,Y/n,N	(Returns to the main menu if y or Y and ends if n or N)	Return to main menu?(y/n)N Have a nice day! >>>
3	Does it give suggestions of keywords if none are detected?	Else: print("Try and enter something that's a little shorter e.g..")	Anything without a keyword	Try and enter something that's maybe a little shorter...	Go ahead. What seems to be your problem? Chicken Nuggets Try and enter something that's maybe a little shorter, such as
4	Does it crash when something unexpected is inputted?	if return1 == "y" or return1 == "Y": print(main_menu())	(anything that isn't y/Y or n/N)	(User will be asked again until valid input is detected)	Return to main menu?(y/n)tg Return to main menu?(y/n)htn Return to main menu?(y/n)asdfghj Return to main menu?(y/n)vv Return to main menu?(y/n)y Go ahead. What seems to be your problem?
5	Is my code correctly indented?	elif user_problem[i] == "screen" or user_problem[i] == "touch": touchScreen_problems()	A sentence with the word "screen" or "touch" in it.	"If you find your touch screen..."	your problem? My touch screen is unresponsive If you find your touch screen is unresponsive, try starting with turning your touch screen off (can be found in the settings of some phones)
6	Are variables correctly defined?	Name = input("What is your name?") Print: ("Ah," + name + ".")	(anything that isn't y,Y/n,N)	Ah, (name), such a.... (etc.)	As always, we like to grace our site with your name My name is: Shaun Ah, Shaun. Such a wonderful name. Shaun Shaun. Such a wonderful name.
7	Is there a backup statement if no keywords are defined?	if Validation == 999: nothingDefined()	N/A	Try and enter something that's maybe a little shorter...	Go ahead. What seems to be your problem? Chicken Nuggets Try and enter something that's maybe a little shorter, such as
8	Does it give appropriate outputs based on input?	Feedback = input("Did this solve your problem?")	n/N	We are sorry we couldn't help you solve your issue.	Did this solve your problem? (y/n)n We are sorry we couldn't help you solve your issue.

9	Does it keep on asking for an input for a specific variable before a condition is met? (Iteration)	<pre>while feedback != "y" or feedback != "Y" or feedback != "n" or feedback != "N": feedback = input("Did this solve your problem? (y/n)")</pre>	(anything that isn't y,Y/n,N)	(User will be asked again until valid input is detected)	<pre>Return to main menu?(y/n)z Return to main menu?(y/n)x Return to main menu?(y/n)c Return to main menu?(y/n)v Return to main menu?(y/n)d</pre>
---	--	--	-------------------------------	--	---

Testing review

At first, my testing was only partially successful. 2 out of the 9 tests were unable to be run, since I still faced the problem of the ELSE statement making my code misbehave. However, all the others worked perfectly fine. I was unable to find a solution to the else statement, so I left it in my code, but in triple quotation marks since it broke it when it was implemented. After more trial and error however (after I had written the testing review), I was able to find a solution as a replacement for the ELSE statement, meaning I had to change one of the tests. It was changed from testing the ELSE statement to testing the newly implemented WHILE loop, that fixed all the issues I had with the ELSE statement.

5. Evaluation

I was required to design, develop, test and evaluate a program to identify keywords in a query typed in by the user and provide a linked solution to common problems related to a mobile device from a selection stored in a text file or database. I had to identify appropriate keywords that can be linked to general advice related problems, whilst accounting for variations in the form of the user input. My solution does solve this problem, and also satisfies everything in the analysis section. The whole for loop in the main_menu() function takes care of identifying keywords, and the "solutions.txt" file I created that stores all the solutions satisfies the requirement to have solutions stored in a txt file. Variation in user input has been accounted for, since the loop only works to pick out keywords amongst the user's input, no matter how much they write. The results in the testing section, show that the code passed the tests, which were specifically based off of the success criteria and test plan. I think that my solution is efficient. For example, the use of def functions makes the code easier to read and identify where mistakes could be made. The use of the .readlines()[] function makes sure that I only have to use one txt file instead of many. However, I just think that the code could've been shorter, since a lot of it was repetitive, especially in the parts where it had to open the specific txt file to read the solution. My pseudocode and flowchart represent my code, and I didn't have to make any changes because I did agile programming instead of the waterfall lifecycle, because I found it easier to make the flowchart and pseudocode after the code had been made. To make my code more efficient, I had to make changes such as adding more functions like the extra return to menu function, as well as new techniques, such as a for loop.

Limitations

I face only one technical issue right at the end of my development. It was only minor however, my computer simply restarted by itself. Fortunately, due to autosave on Microsoft Word, nothing was lost. The hardware used was certainly sufficient enough to produce python code, since it was in good condition (not broken or anything) and the computer(s) I used were fast enough to comfortably run python and other programs (like Microsoft Word and Google Chrome). When looking over my code to tidy it up, I realized that I couldn't use the `\n` function to tidy up the solutions, because they were stored in a txt file, and python commands didn't work there, so for some of the solutions, some words overran onto the next line. Though it's not a serious issue, it's still something worth correcting, as it improves the overall quality of the code.

Further development

I think to improve my code further; I would organize it so that the text would be displayed neater so that it would look better. Although I did find a solution to replace the ELSE statement with a while loop, in theory, it would only work if the user's input was less than 999 characters. While it is unlikely that anyone would actually exceed this limit, it would be better if I could find a solution that wasn't limited to a specific number, just for efficiency purposes. I would also edit it so that if a keyword was entered from two different possible options, it would ask them which one they meant instead of just going to the first one.

6. Conclusion

I was required to create a troubleshooting program for mobile phones that will be able to identify keywords entered by the user to give them the appropriate help they need. To solve this problem, I created a program in python which was designed to assist the user with their phone problems. The way it worked was that when the user entered their query, the program would open a txt file which had stored solutions in it. Based on the user input, it would read a specific line which (hopefully) would contain the user's solution. To solve the problem, I used a combination of my own knowledge, lessons and advice I had received from my teachers and research on the internet. The majority of the code was fairly straightforward, but connecting it was where the difficult part was. Fortunately, I was able to find a suitable solution that would be able to help the user with basic phone issues. It is easy to use and your query is unrestricted, since it detects keywords in a sentence, adding to the efficiency of the code.

7. References

The one website I went to was stackoverflow.com. However, I never actually got a definitive solution from any one of the forums. By looking at some of the solutions, I eventually came up with my own. For example, when I was trying to figure out how to replace the else statement, I was on stackoverflow.com looking at this solution, which gave me an idea of making up my own, which happened to work.

4 Answers

active oldest votes

Do it like this:

13

```
for eachId in listOfIds:
    successful = False
    while not successful:
        response = makeRequest(eachId)
        if response == 'market is closed':
            time.sleep(24*60*60) #sleep for one day
        else:
            successful = True
```

The title of your question is the clue. *Repeating* is achieved by iteration, and in this case you can do it simply with a nested `while`.

share improve this answer

answered Sep 3 '11 at 15:24

David Heffeman
414k • 25 • 559 • 910

thanks for help, nested while ahhh how this not came in to my mind :p – Aamir Adnan Sep 3 '11 at 15:28

add a comment

Get the weekly newsletter! In it, you'll get:

- The week's top questions and answers
- Important community announcements
- Questions that need answers

[Sign up for the newsletter](#)

[see an example newsletter](#)

Linked

1 Don't go to next iteration in for loop

Related

1029 Why is using "for...in" with array iteration a bad idea?

1352 Accessing the index in Python for loops

111 Is it possible to implement a Python for range loop without an iterator variable?

486 for loop to iterate over enum in Java?

Since I already had the knowledge, there wasn't much research to do on the internet. I had the skills necessary to solve this task, it was just a matter of implementing. Throughout the course of this task, I didn't use any new techniques I hadn't been taught already.