Girls' Programming Network

Sassy Security Chatbot!

Keep your secrets safe from nosy siblings or friends with the Sassy Security Chatbot!

This project was created by GPN Australia for GPN sites all around Australia!

This workbook and related materials were created by tutors at:

Sydney, Canberra and Perth

Girls' Programming Network

If you see any of the following tutors don't forget to thank them!!

Writers

Courtney Ross

Testers

Annie Liu
Aadeeba Mou

Intro to Python

Part 0: Setting up

Task 0.1: Set Up the File

Open the start menu, and type 'IDLE'. Select the IDLE 3.X version.

1. Go to the file menu and select 'new file'. This opens a new window.
2. Go to the file menu, select 'Save as'.
3. Go to the Desktop and save the file as 'chatbot.py'.

Task 0.2: You've got a blank space, so write your name!

At the top of the file use a comment to write your name! Any line starting with # is a comment.

```
# This is a comment
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 1:

- ☐ You should have a file called chatbot.py
- ☐ Your file has your name at the top in a comment
- ☐ Run your file with F5 key and it does nothing!!

Part 1: User Identification

Task 1.1: Welcome User

It's time to welcome our user to the Sassy Security Chatbot!

1. `print` a welcome message.

For example:

```
Welcome to the Sassy Security Chatbot!
```

Task 1.2: Identify yourself!

The Sassy Security Chatbot needs to know who they're talking to.

1. Ask the user for their **`name`**. Store the answer in a **`variable`** called **`name`**.

Task 1.3: Personalised Welcome

Now we need to acknowledge the user.

1. `print` a personalised welcome message, using the user's **`name`**.

For example:

```
User Identified: Courtney
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 2:

- ☐ You have printed a welcome message.
- ☐ You have asked for the user's name.
- ☐ Your code prints a personalised welcome.

★ BONUS 1.4: Wait a second!

Let's make our chatbot a bit more human. We can make it wait a bit between responses, so it looks like it's thinking.

1. At the top of your file write:
   ```
   import time
   ```

2. Before any print statement, add:
   ```
   time.sleep(3)
   ```

This will make your program ‘sleep’ for 3 seconds!

Change the number to change how long the chatbot ‘thinks’. Don’t make it too long though!

Part 2: Secret Diary!

Task 2.1: Welcome to the Secret Diary

Our Sassy Security Chatbot protects our secret diary! Let's get access to it.

1. `print` a message, welcoming the user to the secret diary.

2. Ask the user what they would like to do to the secret diary. They can either (1) write to the diary, or (2) read the diary. Store the answer in a **`variable`**.

For example, your code might print:

```
You've gained access to the Secret Diary!
What would you like to do? (1) write in the diary (2) read the diary?
```

Task 2.2: Do you want to write?

Let's check to see if the user wanted to write to the diary.

1. Create an `if` statement that checks to see if the user entered `"1"` so they can write to the secret diary.

2. If they did, `print "You are now writing to the Secret Diary"`.

Hint: If Statements

You can check to see `if` it's raining like this:

```
if raining == "true":
    print("It's raining!")
```

Task 2.3: Open the diary!

So we can write to the diary, we need to open it in a mode called **append**. This mode will create a file if it doesn't already exist and open it in write mode.

1. Inside your `if` statement, add this line of code:
   ```
   f = open("secret_diary.txt", "a")
   ```

The `"secret_diary.txt"` is the name of the file that we're going to create and write to. This file will be created in the same location as your **chatbot.py** file. The `"a"` means to open the file in append mode.

Task 2.4: Write to the diary!

Now that we have the diary open, we can ask the user what they'd like to write and add it to the diary!

1. Ask the user what they'd like to write to the diary. Store the answer in a variable called **newline**.

2. Write the line in the diary with the following code:
   ```
   f.write(newline)
   ```

Task 2.5: Close the diary!

It's really important that we close the text file when we're finished with it!

1. Close the diary with the following code:
   ```
   f.close()
   ```

2. Check to see that it worked by opening the file **secret_diary.txt** in your favourite text editor! Do you see a new line of text in there?

Task 2.6: Read the diary!

Now that we can write to the diary, it's time to read it!

1. Add an `elif` to your if statement, that checks to see `if` the user selected option `"2"`.

2. If the user selected to read the diary, **open** the diary again! Use `"r"` instead of `"a"` to open the diary in read mode.

Task 2.7: Read the diary!

It's time to show the user the contents of the diary!

1. `print` out the contents of the diary to the user. `f.read()` allows you to read the diary.

2. Once we've `print`ed everything out, `close` the diary again because we're finished with it!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 3:

☐ You have asked the user what they would like to do to the diary.

☐ If the user selects option 1, they can write a new line of text to the diary.

☐ If the user selects option 2, they can read the diary.

Part 3: Layers of Security

Task 3.1: First Layer of Security

It's way too easy to get to the secret diary! We have no security! Let's add some in with a password and a decoy password.

1. Add a variable called **`password`**. Store something in there!

2. Add another variable called **`decoy_password`**. Store something different in there.

Task 3.2: What's the secret password?

Let's ask the user if they know what the secret password is!

1. After the user is identified, but before the user has access to the secret diary, ask the user what the secret password is. Store their answer in a **`variable`**.

Task 3.3: Guessing Game

The user might not get the password right on the first try. They might have done a typo! Let's give them some chances to get the password right.

1. Create a `while` loop, that keeps going while the user's answer does not match the **`password`** or **`decoy_password`**.

2. Inside the `while` loop, ask the user again for the password! Store the user's answer in the same **`variable`** as before.

Hint: While Loops

You can keep asking questions in a `while` loop like this:

```
while answer != "4":
      answer = input("What number am I thinking of? ")
```

Hint: Chaining Conditions!

You can use `and` to check multiple conditions:

```
if today == "Monday" and holidays == False:
    print("It's time to go to school!")
```

Task 3.4: Maths questions!

Let's add in one more layer of security for the moment.

1. Ask the user a maths question, such as `"What is 2 + 2? "`. Store the user's answer in a variable.

2. Create a variable called **`correct_answer`**. Store the correct answer to the maths question in there.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 4:

☐ You have a password and a decoy_password.

☐ You keep asking the user for the password until they enter the password or the decoy_password.

☐ You ask the user a maths question, and have the answer!

★ BONUS 3.5: Preventing Brute Force

Right now, we keep asking the user what the **`password`** is until they enter either the **`decoy_password`** or the real **`password`**! This means that the user can keep guessing until they get it right.

1. Create a **`variable`** called **`attempts`** just above the **`while`** loop. Set it to **`0`**.

2. Add another condition to your **`while`** loop, so it only runs while **`attempts`** is less than **`3`**.

3. Inside your **`while`** loop, add **`1`** to **`attempts`** each time!

Now the user will only get 3 attempts to get the password right, before being locked out!

Part 4: Determining Access

Task 4.1: All tests passed

We need to determine what level of access our user gained now.

1. Under the maths question, create an `if` statement that checks to see if the user entered the real **`password`** and got the maths question right.

2. If both conditions are True, set the **`access_type`** to `"allowed"`.

Task 4.2: No decoys allowed here!

Someone has the fake password! Let's redirect them.

1. Add an `elif` to the `if` statement that checks to see if the **`decoy_password`** was entered.

2. If the **`decoy_password`** was entered, set the **`access_type`** to `"decoy"`.

Task 4.3: Access Denied!

If they didn't get anything right, deny the user access!

1. Add an `else` to the `if-elif` statement. Inside it, set the **`access_type`** to `"denied"`.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 5:

☐ You have an if-elif-else statement that sets the access_type.

☐ The user is either allowed, decoy or denied access to the secret diary!

Part 5: Revealing Information!

Task 5.1: Access Level: Allowed

Let's change our code so that the user is only given access to the secret diary if they have permission!

1. Above the code that asks the user what they would like to do with the secret diary, add an `if` statement that checks to see if `access_type` is `"allowed"`.

2. *Indent* all the code for writing and reading the secret diary so it is inside the `if` statement.

Task 5.2: Access Level: Decoy

Our user might have gotten the decoy password! Let's give them some fake information to distract them.

1. Add an `elif` to the `if` statement you just created. Have the elif check to see if `access_type` is `"decoy"`.

2. Inside the `elif` statement, give out some fake information!

For example, your code might print:

```
Here is your 'secret' information ;p
https://www.youtube.com/watch?v=dQw4w9WgXcQ
```

Task 5.3: Access Level: Denied

Lastly, we need to tell any users that didn't get anything correct to go away!

1. Add an `else` to your `if-elif` statement. Tell the user to go away, because they are denied access!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 6:

☐ The Access Level: Allowed gives the user access to the secret diary.

☐ The Access Level: Decoy gives the user access to some fake information.

☐ The Access Level: Denied tells the user to go away!

Part 6: More Security!

Task 6.1: As many questions as we like

We're going to create a function, that takes a **`question`** and **`answer`** and checks to see if the user got it right! We can use functions over and over again to ask different questions.

1. Create a function called **`intelligence_questions`**.

Hint: Functions

Below is an example function

- It is called `check_equal`
- It passes **`num1`** and **`num2`** as parameters

```
def check_equal(num1, num2):
```

Task 6.2: Question Time!

Now we need to ask the user the question!

1. Ask the user the question. Store the user's answer in a variable called **`user_answer`**.

Task 6.3: Ding Ding Ding!

Let's see if the user got it right!

1. Create an `if` statement that checks to see if the **`user_answer`** is equal to **`answer`**. If it is, get the function to `return True`.

2. If the user didn't get the question right, have the function `return False`.

Hint: Returning

Below is a function that adds two numbers together.

- It is called `check_equal`
- It passes **`num1`** and **`num2`** as parameters
- It checks if **`num1`** is equal to **`num2`**
- It `returns True` if **`num1`** is equal to **`num2`**, and `False` if it isn't.

```
def check_equal(num1, num2):
    if num1 == num2:
        return True
    else:
        return False
```

Task 6.4: Calling all questions

We need to call our function, so that we can use it.

1. Call the **`intelligence_questions`** function, and store the returned result in a variable called **`int_answers`**. Make sure you pass in a question and an answer!

Hint: Calling functions

You can call the `check_equal` function like this, and store the returned result in a variable:

```
equal = check_equal("3", "4")
```

Task 6.5: Updating access

We also need to update the `if` statement that sets the access level so that the extra security is used.

1. Add another condition to the `if` statement that sets **`access_type`** to `"allowed"`, that checks to see if **`int_answer`** is equal to `True`.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to the Extensions:

☐ You have a function called intelligence_questions

☐ Your function asked a question, and checks to see if the user got it right.

☐ You can check if any additional intelligence questions were answered correctly as well.

★ BONUS 6.6: Try, Try again!

Just like we did with the password, we might want to let the user have several attempts at the question!

1. Just above the `if` statement, create a **`variable`** called **`attempts`**. Set it to `0`.

2. Below the **`attempts`** variable, but above the `if` statement, create a `while` loop that keeps going while **`attempts`** is less than `3`.

3. Change the **`else`** part of your if statement. Remove the `return False` statement, and instead, ask the **`question`** again. Increase **`attempts`** by `1`.

4. Outside the `while` loop, at the end of the function, add a `return False` statement.

Now the user will get 3 attempts to answer the question correctly!

Extension 7: What's your style?

Task 7.1: Give it a fun theme!

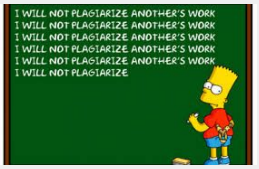
In the instructions, we've based it on keeping a secret diary safe from our little siblings! **But you can add your own theme!!**

Think about a theme you want to add to your chatbot! Some ideas are below!!

A spy theme!!

The chatbot protects the secret identities and speaks like a secret agent.

A secret club for a school project!!



Keep copycats at bay and teach them a lesson for trying to steal your ideas! Teach cheaters a lesson!

Pokemon Go!!

Use your chatbot to store the secret locations of rare pokemon!

A fight between Good and Evil!

Every team has to store their secret info somewhere whether they are in Dumbledore's Army, The Rebel Alliance or the Powerpuff Girls!

Task 7.2: S-S-S-Style!

Use your theme!

1. Change, or add in more print statements to reflect your chosen theme!
2. Change your questions and answers so they match the theme as well!

Extensions 8: Make it Funnier!

Task 8.1: More security, more!

Think about jokes and tricks to add to your chatbot. Some ideas are below!

A Magic 8 Ball!

Trick imposters into thinking that your secret is a magic 8 ball. Distract the imposter with you magic 8 ball!

```
Ask the magic 8 ball for
advice: Will it rain today?
>>> Yes
Ask another question: Should I
go shopping?
>>> Maybe
Ask another question: Will it
rain today?
>>> Ask again later
```

ASCII Art

Add some ASCII art for fun!!!

```
Here is the super secret info
...
              _=,_
           o_/6 /#\\
           \__ |##/
            ='|--\\
              /   #'-.
              \#|_   _'-. /
               |/ \\_( # |"
              C/ ,--___/
Woof...
```

Task 8.2: Magic 8 Ball

Let's create a Magic 8 Ball!

1. At the top of your code, `import random`. This will let us choose things randomly!
2. Create a `list` called `answers`. Add in some Magic 8 answers to this list, like `"Yes"`, `"No"`, or `"Maybe"`. Add as many answers to this list as you like!
3. Use `input` to ask the user what question they'd like the answer to.
4. Use `random.choice` to select an answer from the answers list. Tell the user!
5. Add a `while` loop, so your code keeps asking questions and giving random answers.

Hint: Lists

You can create a list of your favourite foods like this:

```
myFave = ["pizza", "chocolate", "nutella", "lemon"]
```

Hint: Random choice

You can use `random.choice` to select a random number between 1 and 3 like this:

```
mynum = random.choice([1, 2, 3])
```

Task 8.3: ASCII art

Let's show some ASCII art to our secret agents!

1. Pick some ASCII art! You can choose from http://www.ascii-art.de/ascii/, or make your own!

2. Copy and paste it into your program. You will need to `print` your ASCII art.

Note: If your picture doesn't print out exactly as you expect you might have quotes inside it that are breaking your print. Edit the picture, or choose another!

Hint:

You can use triple quotes to `print` things on multiple lines using one `print` statement.

```
print("""This is
A really really
Long answer""")
```

Extension 9: Secret Diary Function!

Task 9.1: Hide that Diary!

Functions are really useful for tidying code! It makes our code easier to read, and easier to use.

1. Create a function called `secret_diary` at the top of your code. This function takes no parameters.

Task 9.2: Change places!

We need to move all the code used for the secret diary into the function.

1. Move all the code that you wrote in part 2 to the `secret_diary` function.
2. Make sure you delete it from the `if` statement, so it doesn't get called twice!

Task 9.3: Using the diary

Now we need to call the function!

1. Call your function from inside the `if` statement that checks to see if `access_type` is equal to `"allowed"`.
2. Make sure your code still works the way you expect! There should be no change to the behaviour of the code.

Extension 10: Random Maths questions!

Task 10.1: That's so random!

Let's generate some random maths questions for security, so they change every time! We'll use the random module, and create another function!

1. If you haven't already, at the top of your code, `import random`.
2. Create a function called `maths_questions`. This function takes no parameters.

Task 9.2: Number Generator!

Now we need to generate the numbers for the maths question! Let's get the answer at the same time.

1. Randomly choose a number between `1` and `10`, and store it in a variable called `num1`.
2. Randomly choose another number between `1` and `10`, and store it in a variable called `num2`.
3. Add the two numbers together, and store it in a variable called `answer`.

Hint: Random Choice

You can select randomly from a list using random.choice:

```
dinner = random.choice(["pizza", "chocolate", "nutella", "lemon"])
```

Hint: Generating lists of numbers

To create a list of numbers from 5 to 19, you can use `range`:

```
myList = list(range(5, 20))
```

Task 9.3: Question Creation

Now we need to create the question, so we can ask it.

1. Assemble a question that asks what happens when you add num1 and num2 together. Store it in a variable called question. Make sure that the user can see what is stored in num1 and num2!

Hint: Converting numbers to strings

You can convert integers into strings, so we can print them:

```
mynum = 4
print(str(mynum))
```

And you can convert strings into integers, so we can compare them!

```
mystring = "4"
mystring = int(mystring)
```

Task 9.4: Ask again!

We already have a function that asks questions and check the answer! Let's use that to ask our question.

1. Call the function **`intelligence_questions`**. Pass in the **`question`** and **`answer`**.

2. Store the result from **`intelligence_questions`** in a variable called **`int_answer`**.

3. At the bottom of the **`maths_question`** function, **`return int_answer`**.

Task 9.5: Who you gonna call?

Let's start using our function!

1. Call the function **`maths_questions`**. Store the result in a variable called **`maths_answer`**.

2. Add another condition to the **`if`** statement that sets **`access_type`** to **`"allowed"`**, that checks to see if **`maths_answer`** is equal to **`True`**.