Girls' Programming Network

Password Cracker

In this workbook, you will learn how to encode plaintext using a hash function and compare it with a stored passphrase for authentication!

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

This workbook and related materials were created by tutors at:

Sydney and Perth

Girls' Programming Network

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

Writers

Alex McCulloch
Renee Noble
Caitlin Shaw
Taylah Griffiths

Testers

Sheree Pudney
Manou Rosenberg

Part 0: Setting up

Intro to Python

Task 0.1: Making a python file

Open the start menu, and type 'IDLE'. Select IDLE.

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 'functions.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
```

Use the F5 key to run your code (you can also go to the Run menu and select 'Run Module'). Click OK when it asks you to Save.
Your code does nothing yet!

☑ CHECKPOINT ☑

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

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

Part 1: Welcome to Passphrases

Task 1.1: Welcome to Passphrases

A passphrase is a sentence that has meaning for you and therefore easier to remember than a password.

One example of a passphrase is: `"The ship sails at midnight"`

We use passphrases rather than passwords as they are longer than passwords and therefore more secure.

Let's make a variable called `correct` that stores a passphrase. This can be any sentence you like!

Hint

To create variable called `favourite` and store a string in it:

```
favourite = "Chocolate"
```

Task 1.2: What is the passphrase?

Let's guess what the passphrase is!

Use `input` to ask the user for their guess. Store their answer in a variable called `guess` so we can use it in our code!

```
What is the passphrase?
```

Hint

To find out someone's favourite ice-cream and store it in a variable called `favourite`

```
favourite = input("What is your favourite ice-cream? ")
```

Task 1.3: Let's see!

Now that we know the user's guess, let's `print` out the correct passphrase and the guess.

For example, here is what your code might look like when you run it:

```
What is the passphrase? My guess passphrase
The ship sails at midnight
My guess passphrase
```

Hint

Remember to use the `guess` variable that you made in Task 1.2!

To print `Hello` we would use this code: `print("Hello")`

☑ CHECKPOINT ☑

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

- ☐ Create a variable storing the passphrase
- ☐ Ask for the passphrase
- ☐ Print the correct passphrase
- ☐ Print the guessed passphrase
- ☐ Try running your code!

Part 2: Is the guess correct?

If Statements

Task 2.1: Check if they have guessed correctly!

Use an `if` statement to tell the user whether they have made the right guess.

You should welcome them if they got it right:

```
What is the passphrase? The ship sails at midnight
Welcome to the club!
```

Hint

In the if statement, compare the user's guess with the passphrase you chose. Don't forget to use == .

To check if someone guessed my favourite fruit

```
guess= "apple"
if guess == "banana":
   print("I love bananas!")
```

Task 2.2: And if they got it wrong!

Under your if statement, add an else statement to tell the user when they made the wrong guess.

You should tell them to go away if they have guessed wrong, like below:

```
What is the passphrase? At midnight the ship sails
Go away!
```

Hint

This is what an if and else statement looks like!

```
guess= "apple"
if guess == "banana":
   print("I love bananas!")
else:
   print("I don't like that fruit")
```

Task 2.3: Stop printing

Now that we have our `if` and `else` statements, we don't need to print out the `correct` and `guess` variables anymore. You can delete those two `print` lines, or you can comment them out.

Hint

To comment out a line of code you can add a # like this:

```
# print("something")
```

☑ CHECKPOINT ☑

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

- ☐ Welcome them if they got the passphrase correct
- ☐ Tell them to go away if they are wrong
- ☐ Run your code and test different guesses
- ☐ Your code doesn't print out the guess or correct passphrases

Part 3: What is Hashing?

Task 3.1: Hash a word by hand (no code for this part!)

First hash function

Replace each letter with its place in the alphabet:

G P N

Now add the numbers together:

Every time we follow this process for the acronym 'GPN', we will get the same number!

Now try hashing this word:

P N G

=

What number did you get? Is this a good thing? What happened here is called a collision!

Second hash function

Now try again but this time multiply the letter's place in the alphabet by its place in the word:

G P N

=

P N G

=

What do you notice?

Hint

You can use the table below to help find what number in the alphabet a letter is:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Task 3.2: Hash your name

Follow the same process as the second hash function and try to hash your name!

★ Bonus 1.4: Does Method 2 always work? ★

Can you find a word that collides with GPN using our second hash function?

Hint

Collision is when 2 different words are hashed to the same number.

☑ CHECKPOINT ☑

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

- ☐ Found the hash of GPN and PNG for both methods
- ☐ Found the hash value of your name

Part 4: Let's hash our code!

Task 4.1: Import the hash library

First we need to import the python library that has pre-made hashing functions - this makes our life easier as we can use code that has been written by other people!

At the very top of your code add the following line:

```
import hashlib
```

This tells our code to look for and use the hashlib library.

Task 4.2: Encode our passphrase

After we set the `correct` variable, create a new variable called `correct_encoded` and set it to encode `correct` using the hashlib library.

Hint

To encode a variable, you use the following code (replace variable_name with the variable you want to use):

```
name_encoded = name.encode()
```

Task 4.3: Time to hash the passphrase!

Create a new variable called `correct_hashed`. Hash the `correct_encoded` variable and store it in `correct_hashed`.

Hint

To hash a variable, you use the following code (replace variable_name with the variable you want to use):

```
name_hashed = hashlib.md5(name_encoded).digest()
```

Remember that hashlib is the library, md5 is the hashing algorithm and digest is what shows us what the hash is.

Task 4.4: Print the hashed passphrase

Now that we have hashed the passphrase, let's `print` the `correct_hashed` so we can see what it looks like!

Once you run your code, copy the printed passphrase to a text file or add it as a comment in your code to save it for use in the next part.

Hint

The hash should look something like this:

```
b'\xcc\xd6R\x16\xb9\x1bP~lK\x01\x0e\x063\x10\xec'
```

Remember that you can save code as a comment like this:

```
#this is a comment
```

☑ CHECKPOINT ☑

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

- ☐ Encoded your passphrase
- ☐ Hashed your passphrase
- ☐ Printed the hashed passphrase
- ☐ Run your code!
- ☐ Copied the printed hash to a text file or comment to use later

Part 5: Making our code secure.

At the moment if someone looks at our code they can see the passphrase written there - that isn't very secure!
To fix this we will store the hash of our passphrase only so that if someone sees our code they can't read the passphrase.

Task 5.1: Replace the string with a hash

Delete the variable `correct` - replace it with a variable called `correct_hashed`.

Store the hash you copied in the previous part to this variable.

Hint

Remember that the hash should look something like this:

```
correct_hashed = b'\xcc\xd6R\x16\xb9\x1bP~lK\x01\x0e\x063\x10\xec'
```

Task 5.2: Delete extra code

Now that we have saved our hashed passphrase, we can delete the code we wrote in the last part.

Delete the line where we create the `correct_encoded` variable and the line where we create the `correct_hashed` variable and the line where we `print` the hashed value.

Task 5.3: Encode the guess

Create a new variable called `guess_encoded`. Store the encoded value of our `guess`!

Hint

If you've forgotten how to do this, have another look at Part 4!

Task 5.4: Hash the guess

Create a new variable called `guess_hashed`. Store the hashed value of our `guess_encoded`.

Hint

If you've forgotten how to do this, have another look at Part 4!

Task 5.5: Compare the hashes

Change your if statement to compare the `guess_hashed` variable and `correct_hashed` variable instead of the `guess` and `hash` variables. Make sure the if statement comes after all the hash code!

☑ CHECKPOINT ☑

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

- ☐ Remove the `correct` variable
- ☐ Encode the guess and store it in the variable `guess_encoded`
- ☐ Hash the guess and store it in the variable `guess_hashed`
- ☐ Change your `if` statement to compare hashes instead of strings
- ☐ Run your code!

Extension 6: Let's get Cracking!

Here is a list of the 10 most common passwords. However, we only have the hashes and forgot to write down what the plain password is! In this part, you will use your python program from parts 0 to 5 to figure what the plain text for each hash is.

Plain text	Username	Hash
	James	b'\x81\xdc\x9b\xdbR\xd0M\xc2\x006\xdb\xd81>\xd0U'
	Robert	b"\xad\xffD\xc5\x10/\xca'\x9f\xceuY\xab\xf6o\xee"
	John	b'%\xf9\xe7\x942;E8\x85\xf5\x18\x1f\x1bbM\x0b'
	Joseph	b'\xd5\xaa\x17)\xc8\xc2S\xe5\xd9\x17\xa5&HU\xea\xb8'
	Andrew	b'\xd0v>\xda\xa9\xd9\xbd*\x95\x16(\x0e\x90D\xd8\x85'
	Ryan	b'\n\xcfE9\xa1K:\xa2}\xee\xb4\xcb\xdfn\x98\x9f'
	Brandon	b'\x1b\xbd\x88d`\x82p\x15\xe5\xd6\x05\xedD%"Q'
	Jason	b'vA\x9cXs\r\x9f5\xdez\xc58\xc2\xfdg7'
	Sarah	b'[\xad\xca\xf7\x89\xd3\xd1\xd0\x97\x94\xd8\xf0!\xf4\x0f\x0e'
	Amber	b"_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99"

Each of these hashes will match one of these plain text passwords:

monkey	11111111	qazwsx	ashley
password	freedom	michael	starwars
	123456789	1234	

Task 6.1: What is the password?

Go back to the website for today's workshop. In your room folder, you should be able to find a text file with the list of the hashes provided above for you to copy and paste into your python program for convenience.

For each hash given above, see if you can use the code you made today to work out the hash of each of the possible plain passwords and match them up!