

Girls’ Programming Network

# *Password Cracker*

| *In this workbook, you will be getting salty with your hashed passwords!*  **TUTORS ONLY** |
| --- |

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

**This workbook and related materials were created by tutors at:**

Sydney, Perth and Canberra



Girls’ Programming Network

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

| **Writers** | **Testers** |
| --- | --- |
| Alex McCulloch  Renee Noble  Caitlin Shaw  Taylah Griffiths | Manou Rosenberg |

# Part 0: Setting up

We are going to be looking at how to make hashes more secure by "salting" our passwords.

| Task 0.1: Back to the beginning |
| --- |
| Let's get started by opening the file we worked on in the first workbook. |

| Task 0.2: Getting the text files. |
| --- |
| Make sure you download the file called “salty-accounts.txt” and add it to the folder we made in the previous workbook. |

| CHECKPOINT |
| --- |
| **If you can tick all of these off you can go to Part 1:**  ☐ You should have the file from the first workbook open (functions.py).  ☐ You have a folder called rainbow\_tables  ☐ You have 3 .txt files in your folder common\_passwords, accounts and salty-accounts |

# 

# 

# Part 1: Creating Salted Hashed Passwords

The idea of adding a salt is primarily to make it harder for rainbow tables to guess your passwords, even if you have a common one. To add a salt, we select a string to be the “salt” and add it to the end of a password before encoding

| Task 1.1: Adding salt |
| --- |
| Add two variables below the import statement:   1. One called saltwith a value of “salty” 2. The other is called correctwith a value of “The ship sails at midnight” and salt *(both strings added together)*. |

| Task 1.2: Encoding correct |
| --- |
| Using the same encode method we used for guess, encode the variable correct, calling it correct\_encoded. |

| Task 1.3: Hashing correct |
| --- |
| Using the same hash and digest method we used for guess, hash and digest the variable correct, calling it correct\_salted\_hash. |

| CHECKPOINT |
| --- |
| **If you can tick all of these off you can go to Part 2:**  ☐ Imported hashlib  ☐ Created salt and correct variables  ☐ Encoded correct  ☐ Hashed and digested correct  ☐ Try running your code! |

# 

| **TUTOR TIPS** |
| --- |
| Their code should look like this:  import hashlib  salt = "salty"  correct = "The ship sails at midnight" + salt  correct\_encoded = correct.encode()  correct\_salted\_hashed = hashlib.md5(correct\_encoded).digest()  correct\_hashed = b'\xcc\xd6R\x16\xb9\x1bP~lK\x01\x0e\x063\x10\xec'  guess = input("What is the passphrase? ")  guess\_encoded = guess.encode()  guess\_hashed = hashlib.md5(guess\_encoded).digest()  if guess\_hashed == correct\_hashed:  print("Welcome to the club!")  else:  print("Go away!") |

# 

# Part 2: Saving Salted Hash

| Task 2.1: Changing the saved hash |
| --- |
| Print out the value of correct\_salted\_hash and copy it into the correct\_hashed variable, replacing the old hash value. Then you can delete the print statement you just wrote. |

| Task 2.2: Removing Lines |
| --- |
| In this task we will be deleting some lines of code - we are deleting these so we do not have our correct password visible to anyone who could read our code!  Remove the lines from the last pass that helped you make the salted hash. Make sure you don’t delete the salt variable. |

| Task 2.3: Comparing salty things |
| --- |
| To compare the guess with the correct answer, we need to add the salt to the guess as well!  Before you encode the guess, add the salt to it. |

| CHECKPOINT |
| --- |
| **If you can tick all of these off you can go to Part 3:**  ☐ Changed the variable name and value  ☐ Removed the lines of code  ☐ Added salt to the guess as well  ☐ Run your code! |

# 

# 

| **TUTOR TIPS** |
| --- |
| Their code should look like this:  import hashlib  salt = "salty"  correct\_hashed = b'\xae\xbc\xe9f\xd1%Q\x8a\xf2\xfd\xba\xb2\x922`\xa5'  guess = input("What is the passphrase? ")  guess = guess + salt  guess\_encoded = guess.encode()  guess\_hashed = hashlib.md5(guess\_encoded).digest()  if guess\_hashed == correct\_hashed:  print("Welcome to the club!")  else:  print("Go away!") |

# 

# Part 3: Printing Salt!

We're going to try and guess the salt that is being used by coming up with every possible salt (to make this easier we know that the salt is a number between 1000 and 9999 but in the real world it would be very long and include letters and symbols) and trying to see if we can crack a password with each salt - if we can crack one, then we know that's the salt that is being used (again, to make this easier we're telling you that `password` is definitely being used by at least one user).

| Task 3.1: Loop through salt |
| --- |
| Create a new file for this code. Call it **salty\_guesser.py** In this new file, loop through the numbers in the range of 1000-9999 and print out each one. |
| *Hint* |
| To print out number in the range of 1-10 you would write:  for number in range(1, 10):  print(number) |

| CHECKPOINT |
| --- |
| **If you can tick all of these off you can go to Part 4:**  ☐ Create a new file that contains a for loop  ☐ Print out each posible salt  ☐ Run your code! |

# 

| **TUTOR TIPS** |
| --- |
| Their code should look like this:  for salt in range(1000, 9999):  print(salt) |

# 

# 

# Part 4: Saving a file to list!

This part should be done **above the for loop** created in the previous part.

| Task 4.1: Creating a list |
| --- |
| Create an empty list called salted\_passwords. |

| Task 4.2: Open file and create a for loop |
| --- |
| Open the file called *salty-accounts.txt* in a for loop so you can read each line. |
| *Hint* |
| If you’ve forgotten how to do this, have a look at your code from Workbook 2!Tasks 4.3 - 4.6 will be written underneath this for loop. |

| Task 4.3: Strip the whitespace from each line |
| --- |
| Use the .strip() method to remove the whitespace from each line. |

sa

| Task 4.4: Split each line at the comma |
| --- |
| Next, we want to use the .split() method to split the account name away from the account’s hashed password at the comma, store in a list called account. |

| Task 4.5: Add a password\_hash variable |
| --- |
| Put the second value from account into a variable called password\_hash. |

| Task 4.6: Append to salted\_passwords |
| --- |
| Append password\_hash to the list called salted\_passwords. |
| *Hint* |
| To append something to a list we can use code like this: pets = [“Emmy”, “Saphira”, “BiBi”]  pets.append(“Artemis”) |

| CHECKPOINT |
| --- |
| **If you can tick all of these off you can go to Part 5:**  ☐ Create an empty list  ☐Create a for loop  **Inside the for loop you:**  ☐ Stripped each line of whitespace  ☐Split each line at the comma  ☐Put each password\_hash into a variable  ☐Appended password\_hash to salted\_passwords |

# 

| **TUTOR TIPS** |
| --- |
| Their code should look like this:  salted\_passwords = []  for line in open("salty-accounts.txt"):  line = line.strip()  account = line.split(",")  password\_hash = account[1]  salted\_passwords.append(password\_hash)    for salt in range(1000, 9999):  print(salt) |

# Part 5: Checking hash is salt

This entirety of the task is done underneath the for loop that is looping through possible salts

| Task 5.1: Creating possible\_salted |
| --- |
| Create a variable called *possible\_salted* with a value of “password” and add the salt number to the end of “password” (make sure you change it to a string!) |
| *Hint* |
| To change a number into a string, use this code:  age = 17  birthday = “Happy Birthday! You are “ + str(age) + “ years old!” |

| Task 5.2: Creating possible\_encoded |
| --- |
| Create a variable called possible\_encodedwith a value of possible\_saltedencoded. |
| *Hint* |
| Remember to import hashlib in this file! |

| Task 5.3: Hash and digest possible\_encoded |
| --- |
| Create a variable called possible\_hashed with a value of possible\_encodedhashed and digested. Then make this variable a string! |

| Task 5.4: Check if *possible\_hashed* in *salted\_passwords* |
| --- |
| Use an if statement to check if possible\_hashedis in salted\_passwords.  Under that if statement, print salt, then break from the loop. If it isn’t in salted\_passwords then we know this isn’t the salt, and the loop can keep going |

| CHECKPOINT |
| --- |
| **If you can tick all of these off you can go to Part 6:**  ☐ Created *possible\_salted*  ☐ Created *possible\_encoded*  ☐ Created *possible\_hashed*  ☐ Created if statement  ☐ Printed *salt* |

# 

| **TUTOR TIPS** |
| --- |
| Their code should look like this:  import hashlib  salted\_passwords = []  for line in open("salty-accounts.txt"):  line = line.strip()  account = line.split(",")  password\_hash = account[1]  salted\_passwords.append(password\_hash)    for salt in range(1000, 9999):  possible\_salted = "password" + str(salt)  possible\_encoded = possible\_salted.encode()  possible\_hashed = str(hashlib.md5(possible\_encoded).digest())  if possible\_hashed in salted\_passwords:  print(salt)  break |

# 

# 

# Part 6: Adding to Rainbow Table

Adding the salt to our old rainbow table code. This part should all be written in our rainbow table python file from Workbook 2

| Task 6.1: Create salt |
| --- |
| At the top of the file, create a variable called salt with the number value that you found in the last part as a string. |

| Task 6.2: Add salt |
| --- |
| Inside the first for loop, before we encode each password add the salt to the password |

| Task 6.3: Changing the file |
| --- |
| In the second for loop, instead of opening “accounts.txt”, open “salty-accounts.txt”. |

| CHECKPOINT |
| --- |
| **If you can tick all of these off you can go to Part 7:**  ☐ Added the salt to each password  ☐ Using salty-accounts instead of the normal accounts file  ☐ Run your code! |

# 

# 

| **TUTOR TIPS** |
| --- |
| Their code should look like this:  import hashlib  rainbow = {}  salt = "7549"  for line in open("common-passwords.txt"):  password = line.strip()  password = password + salt  password\_encoded = password.encode()  password\_hash = hashlib.md5(password\_encoded)  password\_hash = str(password\_hash.digest())  rainbow[password\_hash] = password  for line in open("salty-accounts.txt"):  line = line.strip()  account = line.split(",")  name = account[0]  password\_hash = account[1]  if password\_hash in rainbow:  print(name)  print(rainbow[password\_hash]) |

# 

# 7. Extension: Finding Secrets

| Task 7.1: Secrets! |
| --- |
| Using the accounts and passwords you found before, go to the following link to find secrets on the website!  <https://girls-programming-network.github.io/meme-exchange/> |

# 