

Hashing Functions

Welcome to the labs!



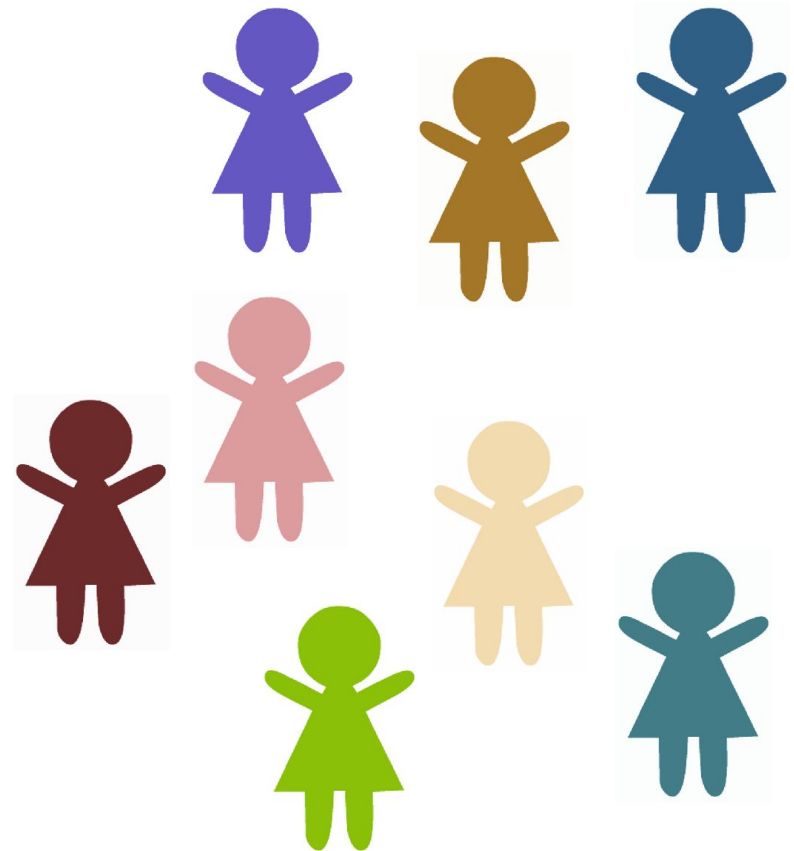
Who are the tutors?

Who are you?

People Bingo

Rules: 1) Read each square. 2) Find a new friend who can complete any of the squares. 3) One friend can fill in **one square only**. 4) The first person to fill in **every square** wins!

Can fluently speak more than one language 🗣️	Has been to GPN before ❤️	Has visited another country ✈️	Can play chess ♟️	Has more than 1 pet 🐶
Favourite food is Pizza 🍕	Knows what a Kumquat is ?	Has made their own pasta 🍝	Has eaten Dragon fruit 🍉	Was born in another country 🌍
Knows how to Juggle 🤹	Likes to play Tetris 🎮	Is an only child 👤	Likes to paint 🎨	Knows a magic trick 🪄
Plays an instrument 🎵	Can ride a skateboard 🛹	Likes to read 📖	Has the same favourite colour as you 🌈	Has the same eye colour as you 👁️
Has a part time or casual job 💰	Is in the same school year as you 🚌	Knows the second verse of the national anthem 🇺🇸	Has created their own coding project 💻	Plays sport on the weekend ⚽



Tell us you're here!

Click on the
Start of Day Survey
and fill it in now!

Password Cracker!

Today's project!



Using the workbook!

The workbooks will help you put your project together!

Each **Part** of the workbook is made of tasks!

Tasks - The parts of your project

Follow the tasks **in order** to make the project!

Hints - Helpers for your tasks!

Stuck on a task, we might have given you a hint to help you **figure it out**!

The hints have **unrelated** examples, or tips. **Don't copy and paste** in the code, you'll end up with something **CRAZY**!

Task 6.2: Add a blah to your code!

This has instructions on how to do a part of the project

1. **Start by doing this part**
2. **Then you can do this part**

Task 6.1: Make the thing do blah!

Make your project do blah

Hint

A clue, an example or some extra information to help you **figure out** the answer.

```
print('This example is not part of the project' )
```

Using the workbook!

The workbooks will help you put your project together!

Check off before you move on from a **Part!** Do some bonuses while you wait!

Checklist - Am I done yet?

Make sure you can tick off every box in this section before you go to the next Part.

Lecture Markers

This tells you you'll find out how to do things for this section during the names lecture.

Bonus Activities

Stuck waiting at a lecture marker? Try a purple bonus. They add extra functionality to your project along the way.



CHECKPOINT



If you can tick all of these off you're ready to move the next part!

- ☐ Your program does blah
- ☐ Your program does blob



★ BONUS 4.3: Do some extra!

Something to try if you have spare time before the next lecture!

If Statements



Conditions

Sometimes, we want our code to make **choices**.

This means **certain parts** will only run if **certain conditions** are met.



If it's raining take an umbrella

Yep it's raining

..... take an umbrella

Conditions can either be **True or False**.

Computers call true/false values **booleans**.

If Statements

We use **if/elif/else statements** to:

- check which conditions are **True/False**
- **pick** which parts of the code to run

What is **"elif"**?

elif is short for **"else if"**

```
word = "Chocolate"
if word == "GPN":
    print("GPN is awesome!")
elif word == "Chocolate":
    print("YUMMM Chocolate!")
else:
    print("The word isn't GPN :(")
```

What happens??

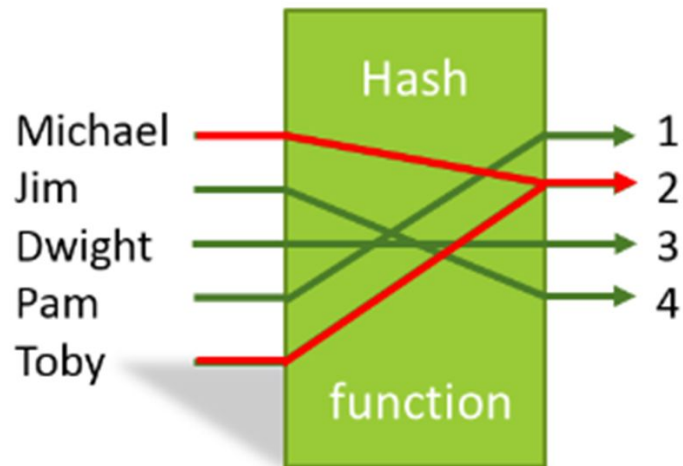
Project Time!

You now know all about **if** and **else**!

Let's put what we learnt into our project
Try to do Parts 0 - 2!

The tutors will be around to help!

Hashing



What is Hashing?

Hashing is the process of converting any piece of data (called a *key*) into another *value*.

A **hash function** is used to generate the new value according to a mathematical algorithm.

The result of a hash function is known as a **hash value**.


Reference from: <https://www.educative.io/edpresso/what-is-hashing>

What is Hashing?

How does it work?

We take a readable word or phrase (this is called plaintext) like this:

password

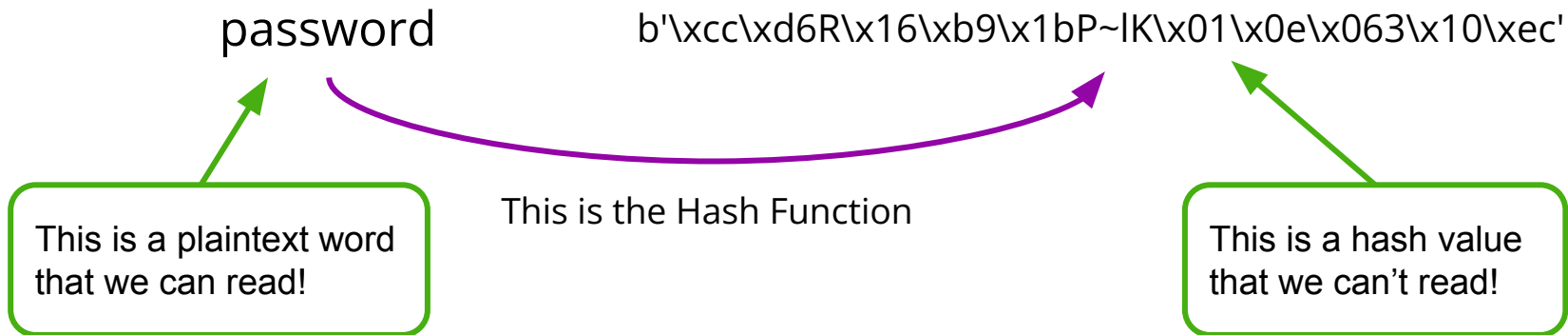


This is a plaintext word
that we can read!

What is Hashing?

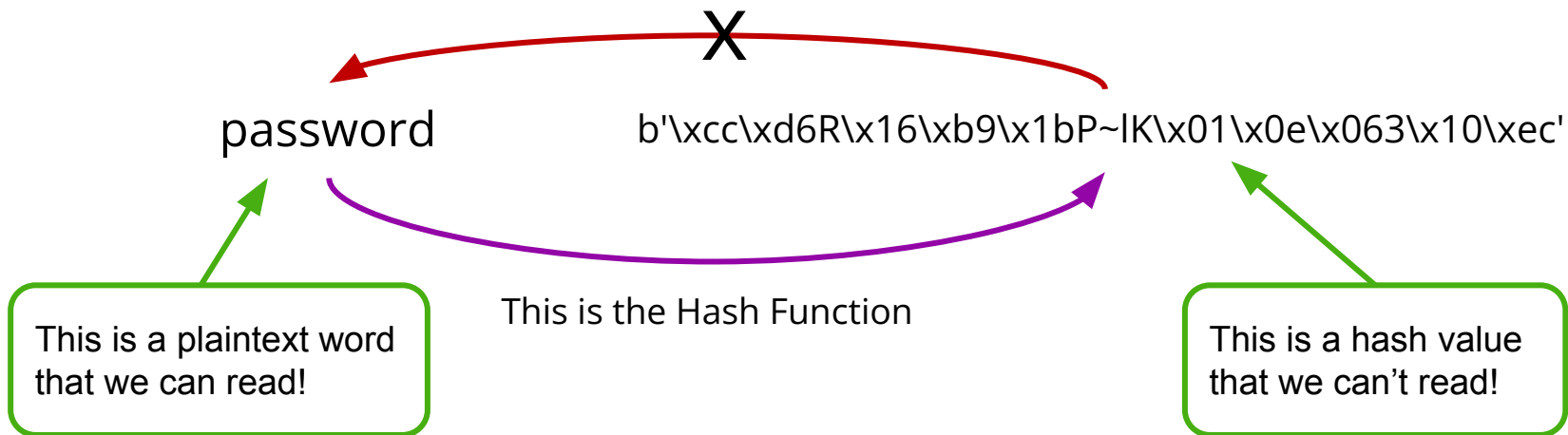
How does it work?

We take a readable word or phrase (this is called plaintext) like this:



And we use a “Hash function” to turn it into something we can’t read!

What is Hashing?



The coolest thing about a Hash function is that you can only go **one way**, so you can't work out what the plaintext word was if you only have the hash value - this makes it secure!

Hashing in Python

Firstly to use the Python code we need to import the hashing library!

We can do this by writing:

```
import hashlib
```

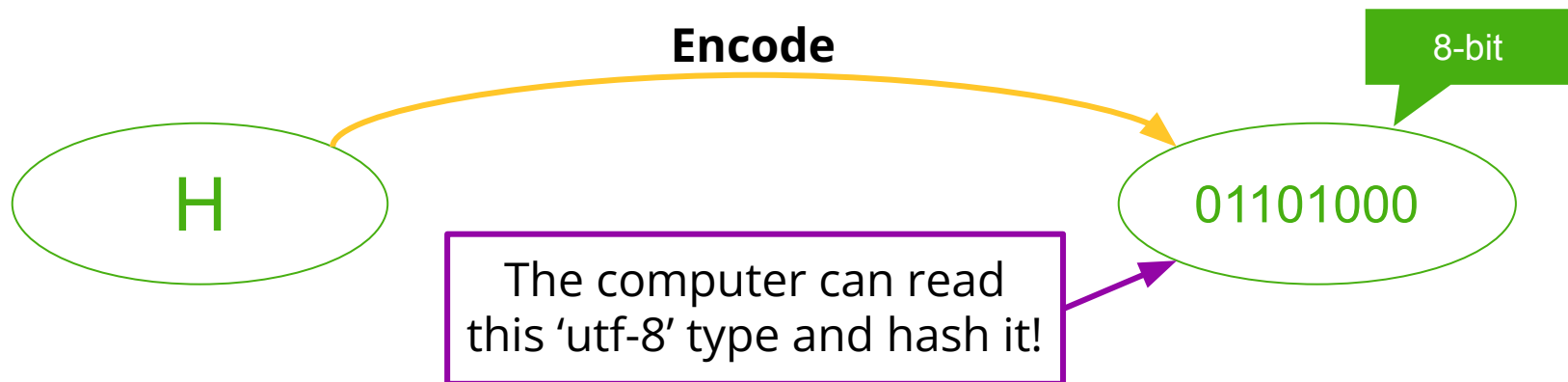
at the top of our code!

Encoding!

Next we need to **prepare** our data for hashing -
by **encoding** it!

```
my_string = "hello"
```

```
my_string_encoded = my_string.encode()
```



Hashing!

We had to encode the string as only very specific data types like “utf-8” can be hashed.

Now we can actually hash our value!

To hash a value we can use the .md5 function written:

```
my_string_hashed = hashlib.md5(my_string_encoded)
```

Digest!

After hashing our variable we want to turn it into a value we can use, so we use the `.digest()` method, written:

```
my_string_hashed = hashlib.md5(my_string_encoded).digest()  
print(my_string_hashed)
```

Result:

```
b' ]A@*\xbcK*v\xb9q\x9d\x91\x10\x17\xc5\x92 '
```

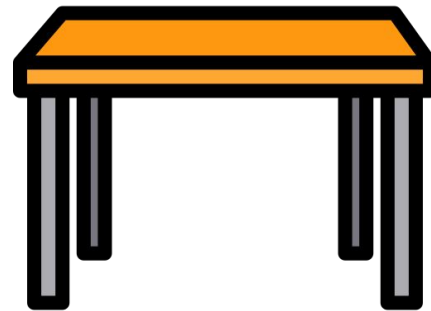
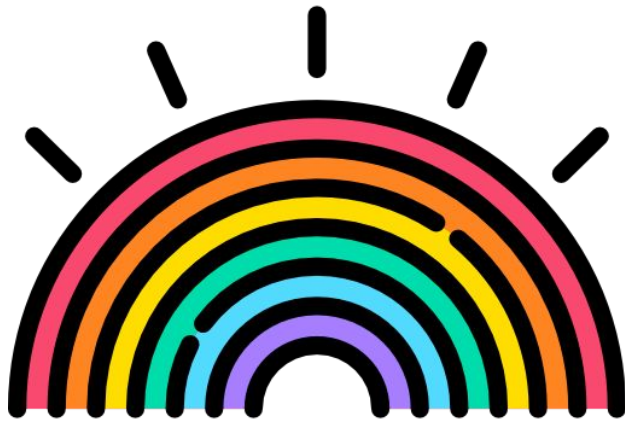
Project Time!

Hashing!

Let's put what we learnt into our project
Try to do Parts 3 - 5!

The tutors will be around to help!

Rainbow Tables



What is a Rainbow Table?

How can we figure out a password?

Remember that we **can not** work backwards from a hash because they are **irreversible**

But every unique string gives a unique hash
(e.g. if you hash 'apple' it will always give the same hash)

What if we work **forwards** instead?

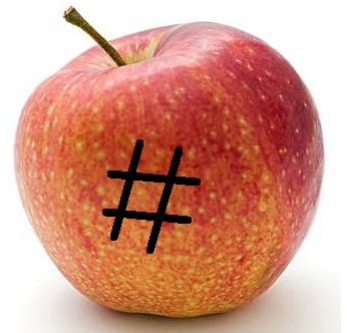
We could **guess** the password!

What is a Rainbow Table?

Plaintext password = 'apple'

hashed password =

b'\x1f8p\xbe'Oll\xb3\xe3\x1a\x0cg(\x95\x7f'



What is a Rainbow Table?

We can make **a lot** of guesses!

A **rainbow table** is a collection of a lot of possible passwords
and their hashes

It makes sense to use the **most common passwords** in the table
This will help you break into the most accounts!

Rainbow tables show you “**the entire spectrum of possibilities**”.

<https://stackoverflow.com/questions/5051608/why-is-it-called-rainbow-table>

Let's Crack a password

We've discovered a list of leaked password hashes.



Let's try and crack one of them with a rainbow table!

What's the plaintext of

b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'

Our rainbow table

Plaintext	Hashes
password	b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'
password123	b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98l\x1e8'
p@ssw0rd	b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'
1234567890	b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'
qwerty	b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\\xa4'
qwerty123	b'?\xc0\xa7\xac\xf0\x87\xf5l\xac+&k\xaf\x94\xb8\xb1'

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match?

Plaintext	Hashes
password	<code>b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'</code>
password123	<code>b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98\x1e8'</code>
p@ssw0rd	<code>b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'</code>
1234567890	<code>b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'</code>
qwerty	<code>b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'</code>
qwerty123	<code>b'?\xc0\xa7\xac\xf0\x87\xf5\xac+&k\xaf\x94\xb8\xb1'</code>

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match? No, let's try the next one

Plaintext	Hashes
password	<code>b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'</code>
password123	<code>b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98\x1e8'</code>
p@ssw0rd	<code>b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'</code>
1234567890	<code>b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'</code>
qwerty	<code>b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'</code>
qwerty123	<code>b'?\xc0\xa7\xac\xf0\x87\xf5\xac+&k\xaf\x94\xb8\xb1'</code>

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match?

Plaintext	Hashes
password	<code>b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'</code>
password123	<code>b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98l\x1e8'</code>
p@ssw0rd	<code>b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'</code>
1234567890	<code>b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'</code>
qwerty	<code>b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'</code>
qwerty123	<code>b'?\xc0\xa7\xac\xf0\x87\xf5l\xac+&k\xaf\x94\xb8\xb1'</code>

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match? No, let's try the next one

Plaintext	Hashes
password	<code>b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'</code>
password123	<code>b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98\x1e8'</code>
p@ssw0rd	<code>b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'</code>
1234567890	<code>b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'</code>
qwerty	<code>b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'</code>
qwerty123	<code>b'?\xc0\xa7\xac\xf0\x87\xf5\xac+&k\xaf\x94\xb8\xb1'</code>

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match?

Plaintext	Hashes
password	<code>b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'</code>
password123	<code>b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98\x1e8'</code>
p@ssw0rd	<code>b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'</code>
1234567890	<code>b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'</code>
qwerty	<code>b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'</code>
qwerty123	<code>b'?\xc0\xa7\xac\xf0\x87\xf5\xac+&k\xaf\x94\xb8\xb1'</code>

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match? No, let's try the next one.

Plaintext	Hashes
password	<code>b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'</code>
password123	<code>b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98\x1e8'</code>
p@ssw0rd	<code>b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'</code>
1234567890	<code>b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'</code>
qwerty	<code>b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'</code>
qwerty123	<code>b'?\xc0\xa7\xac\xf0\x87\xf5\xac+&k\xaf\x94\xb8\xb1'</code>

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match?

Plaintext	Hashes
password	b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'
password123	b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98l\x1e8'
p@ssw0rd	b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'
1234567890	b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'
qwerty	b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'
qwerty123	b'?\xc0\xa7\xac\xf0\x87\xf5l\xac+&k\xaf\x94\xb8\xb1'

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match? Yes! Now we can get the plaintext.

Plaintext	Hashes
password	b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'
password123	b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98l\x1e8'
p@ssw0rd	b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'
1234567890	b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'
qwerty	b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'
qwerty123	b'?\xc0\xa7\xac\xf0\x87\xf5l\xac+&k\xaf\x94\xb8\xb1'

Let's Crack a password

Hash to crack: `b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'`

Do these match? Yes! Now we can get the plaintext. It's 1234567890

Plaintext	Hashes
password	b'_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99'
password123	b'H,\x81\x1d\xa5\xd5\xb4\xbcml\x7f\xfa\x98l\x1e8'
p@ssw0rd	b'\x0f5\x97@\xbd\x1c\xda\x99O\x8bU3\x0c\x86\xd8E'
1234567890	b'\xe8\x07\xf1\xfc\xf8-\x13/\x9b\xb0\x18\xcag8\xa1\x9f'
qwerty	b'\xd8W\x8e\xdf\x84X\xce\x06\xfb\xc5\xbbv\xa5\x8c\\xa4'
qwerty123	b'?\xc0\xa7\xac\xf0\x87\xf5l\xac+&k\xaf\x94\xb8\xb1'

Choosing passwords

Storing hashes for **every single** possible password would use up **a lot of space**.

So hackers usually only create rainbow tables with the most common passwords.....

Which is why you shouldn't use them. That would make your account very easy to get into.

Data Breaches

Stealing passwords is something criminals are always trying to do.

These companies have had their users' passwords exposed online recently. How many of them do you use?

Facebook	(2019)	533 million users
Linkedin	(2021)	700 million users
Twitter	(2018)	330 million users
Uber	(2016)	57 million users
Twitch	(2021)	7 million
Zoom	(2020)	500,000

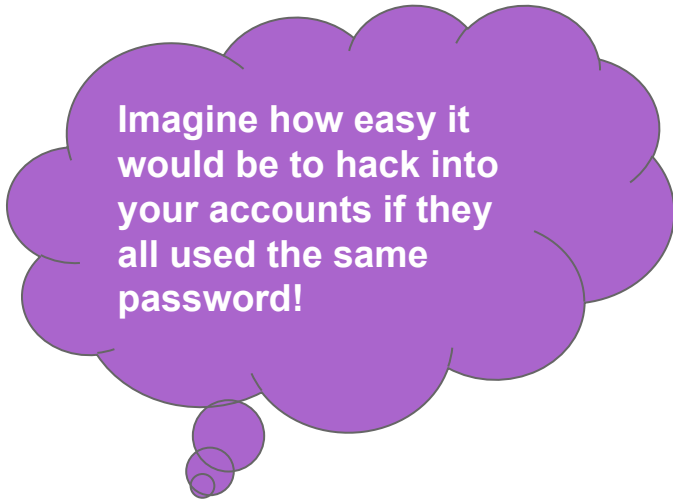
[src: https://www.upguard.com/blog/biggest-data-breaches](https://www.upguard.com/blog/biggest-data-breaches)

Data Breaches

Stealing passwords is something criminals are always trying to do.

These companies have had their users' passwords exposed online recently. How many of them do you use?

Facebook	(2019)	533 million users
Linkedin	(2021)	700 million users
Twitter	(2018)	330 million users
Uber	(2016)	57 million users
Twitch	(2021)	7 million
Zoom	(2020)	500,000



Imagine how easy it would be to hack into your accounts if they all used the same password!

[src: https://www.upguard.com/blog/biggest-data-breaches](https://www.upguard.com/blog/biggest-data-breaches)

Project Time!

Now that you've learnt all about rainbow tables!

Let's put what we learnt into our project

**Try to do Part 0
of workbook 2!**

The tutors will be around to help!

Files and For Loops



Opening files!

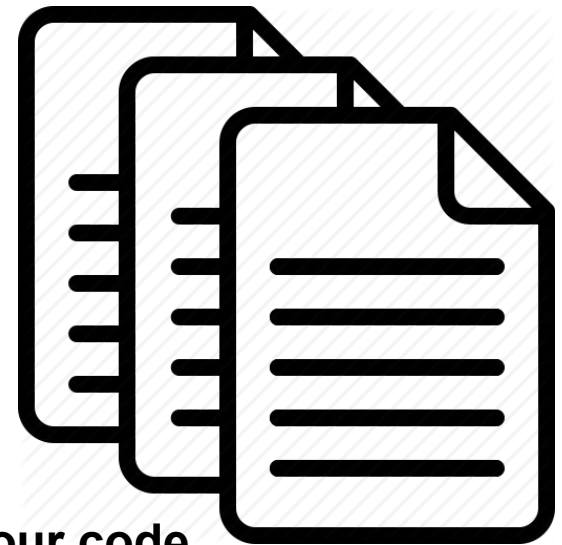
We can store information inside files.

Then we can use these files in our code!

To get access to the stuff inside a file in python, we need to **open** it!

(That doesn't mean clicking on the little icon)

```
for line in open("test.txt"):
    print(line)
```




Important: The file needs to be in the same place as your code

Opening Files

Let's break that code down a bit!

```
for line in open("test.txt"):
    print(line)
```



This bit is called a **for loop**, we'll learn about that in the next few slides!

Opening Files

Let's break that code down a bit!

```
for line in open("test.txt"):  
    print(line)
```

This bit is called a **for loop**, we'll learn about that in the next few slides!

Over here is where we tell our code that we want to open the file!

Opening Files

Let's break that code down a bit!

```
for line in open("test.txt"):  
    print(line)
```

This bit is called a **for loop**, we'll learn about that in the next few slides!

Over here is where we tell our code that we want to open the file!

Here is where we put the name of the file we want to open

Opening Files

Let's break that code down a bit!

```
for line in open("test.txt"):  
    print(line)
```

This bit is called a **for loop**, we'll learn about that in the next few slides!

Here is where we put the name of the file we want to open

Over here is where we tell our code that we want to open the file!

Finally, this is what we want to do with each line of the file.

For Loops

For loops allow you to do something for **each** item in a **group** of things

There are many real world examples, like:



**For each page in this book:
Read**



**For each chip in this bag of chips:
Eat**

Reading line by line

My cat wrote this Haiku for me and I want my program to print out each line:

Wanna go outside.
Oh NO! Help I got outside!
Let me back inside!

Reading line by line

My cat wrote this Haiku for me and I want my program to print out each line:

Wanna go outside.
Oh NO! Help I got outside!
Let me back inside!

We can use **for loops** to read individual lines of files


Reading line by line

My cat wrote this Haiku for me and I want my program to print out each line:

Wanna go outside.
Oh NO! Help I got outside!
Let me back inside!

We can use **for loops** to read individual lines of files

```
for line in open('haiku.txt'):  
    print(line)
```



What do you think
this will print?

Reading line by line

My cat wrote this Haiku for me and I want my program to print out each line:

Wanna go outside.
Oh NO! Help I got outside!
Let me back inside!

We can use **for loops** to read individual lines of files

```
for line in open('haiku.txt'):  
    print(line)
```

Wanna go outside.

Oh NO! Help! I got outside!

Let me back inside!

But there are extra lines?



Chomping off the newline

The newline character is represented by `\n`:

```
print('Hello\nWorld')
```

Hello

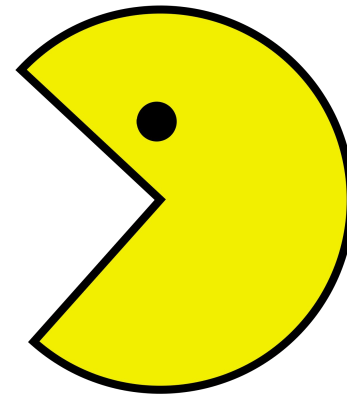
World

We can remove it from the lines we read with `.strip()`

```
x = 'abc\n'
```

```
x.strip()
```

'abc'



Reading and stripping!

```
for line in open('haiku.txt'):
    line = line.strip()
    print(line)
```

Wanna go outside.
Oh NO! Help! I got outside!
Let me back inside!

No extra lines!



A missing file causes an error

This is what happens if you try to open a file that doesn't exist:

```
open('missing.txt')
```

```
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
IOError: [Errno 2] No such file or  
directory: 'missing.txt'
```



Project Time!

Now you know how to use for loops and files!

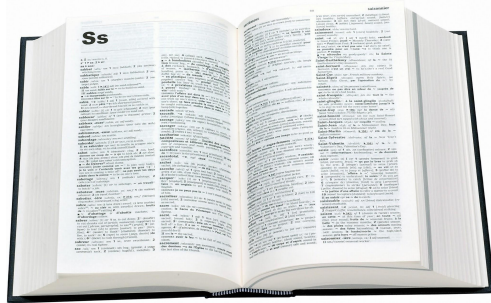
**Try to do Part 1
of workbook 2!!**

The tutors will be around to help!

Dictionaries



Dictionaries!

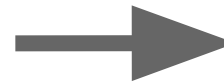
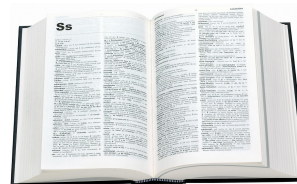


You know dictionaries!

**They're great at looking up thing
by a word, not a position in a list!**

Look up

Hello



Get back

***A greeting (salutation) said
when meeting someone or
acknowledging someone's
arrival or presence.***

Looking it up!

There are lots of times we want to look something up!



Competition registration

Team Name → List of team members



Phone Book

Name → Phone number



Vending Machine

Treat Name → Price

Looking it up!



Phone Book

Name → Phone number

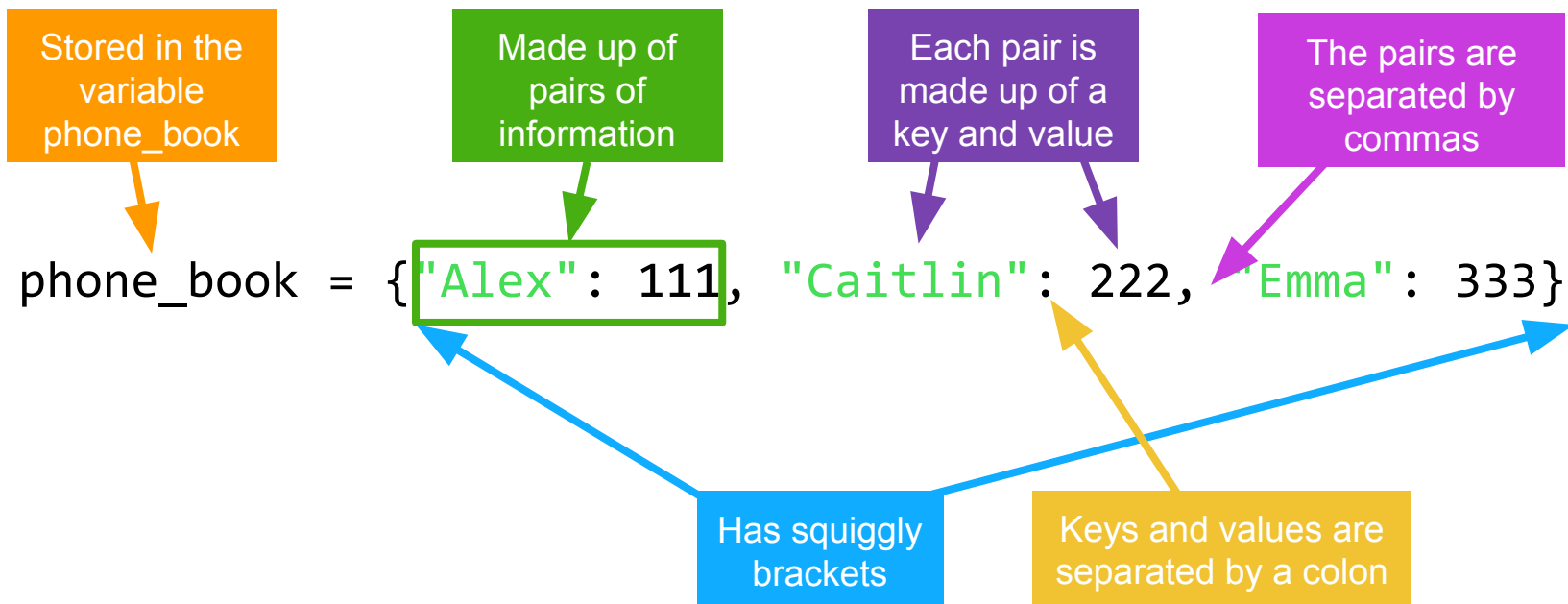
↑
Key

↑
Value

**We can use a dictionary for anything with a
key → value pattern!**

Dictionaries anatomy!

This is a python dictionary!



This dictionary has Alex, Caitlin and Emma's phone numbers

Playing with dictionaries!

Let's try using the phone book!

- Let's create the phonebook

```
>>> phone_book = {  
    "Alex": 111, "Caitlin": 222, "Emma": 333  
}
```

- Let's get Alex's number from the phonebook. What will it be?

```
>>> phone_book["Alex"]
```

Playing with dictionaries!

Let's try using the phone book!

- Let's create the phonebook

```
>>> phone_book = {  
    "Alex": 111, "Caitlin": 222, "Emma": 333  
}
```

- Let's get Alex's number from the phonebook. What will it be?

```
>>> phone_book["Alex"]  
111
```

Updating our dictionaries!

- Alex changed her number. Let's update it!

```
>>> phone_book["Alex"] = 123
```

```
>>> print(phone_book)
```

- We met Rowena! Let's add her to our phone book

```
>>> phone_book["Rowena"] = 444
```

```
>>> print(phone_book)
```


Updating our dictionaries!

- Alex changed her number. Let's update it!

```
>>> phone_book["Alex"] = 123
```

```
>>> print(phone_book)
```

```
{ "Alex": 123, "Caitlin": 222, "Emma": 333 }
```

- We met Rowena! Let's add her to our phone book

```
>>> phone_book["Rowena"] = 444
```

```
>>> print(phone_book)
```

Updating our dictionaries!

- Alex changed her number. Let's update it!

```
>>> phone_book["Alex"] = 123
```

```
>>> print(phone_book)
```

```
{ "Alex": 123, "Caitlin": 222, "Emma": 333 }
```

- We met Rowena! Let's add her to our phone book

```
>>> phone_book["Rowena"] = 444
```

```
>>> print(phone_book)
```

```
{ "Alex": 123, "Caitlin": 222, "Emma": 333,  
  "Rowena": 444 }
```

Project time!

You now know all about dictionaries!

Let's put what we learnt into our project
Try to do Part 2
of workbook 2!

The tutors will be around to help!

Python Lists



Lists can store multiple things

A **list** is an ordered group of related items, all stored in the same variable

```
>>> day1 = 'Monday'
>>> day2 = 'Tuesday'
>>> day3 = 'Wednesday'
>>> day4 = 'Thursday'
>>> day5 = 'Friday'
>>> day6 = 'Saturday'
>>> day7 = 'Sunday'
```

```
>>> days = ['Monday', 'Tuesday', 'Wednesday',
            'Thursday', 'Friday', 'Saturday', 'Sunday']
```

Your Favourite Things!

```
>>> faves = ['books', 'butterfly', 'chocolate',  
'skateboard']
```



Accessing Lists!

- The favourites **list** holds four strings in order.
- We can count out the items using index numbers!

0



-4

1



-3

2



-2

3



-1

- **Indices start from zero!**

You can put (almost) anything into a list

- You can have a list of **integers**

```
>>> primes = [1, 2, 3, 5, 11]
```

- You can have **lists** with mixed **integers** and **strings**

```
>>> mixture = [1, 'two', 3, 4, 'five']
```

- But this is almost **never** a good idea! You should be able to treat every element of the **list** the same way.

Falling off the edge

Python complains if you try to go **past the end** of a **list**

```
>>> faves = ['books', 'butterfly', 'chocolate', 'skateboard']  
>>> faves[4]
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
IndexError: list index out of range
```

Updating items!

We can also **update things** in a list:

```
>>> faves = ['books', 'butterfly', 'chocolate',  
'skateboard']
```

```
>>> faves[1]  
'Butterfly'
```

```
>>> faves[1] = 'new favourite'  
>>> faves[1]  
'new favourite'
```

Removing items!

- We can remove items from the list if they're no longer needed!
- What if we decided that we didn't like butterflies anymore?

```
>>> faves.remove('butterfly')
```

- What does this list look like now?



Looping over a list of ints

We can loop through a list:

```
numbers = [1, 2, 3, 4]
for i in numbers:
    print(i)
```

What's going to happen?

Looping over a list of ints

We can loop through a list:

```
numbers = [1, 2, 3, 4]
for i in numbers:
    print(i)
```

What's going to happen?

```
>>> 1
>>> 2
>>> 3
>>> 4
```

- Each item of the list takes a turn at being the variable `i`
- Do the body once for each item
- We're done when we run out of items!

Project time!

Lists!

Let's put what we learnt into our project
Try to do Parts 3 - 4
of workbook 2!

The tutors will be around to help!

Salting



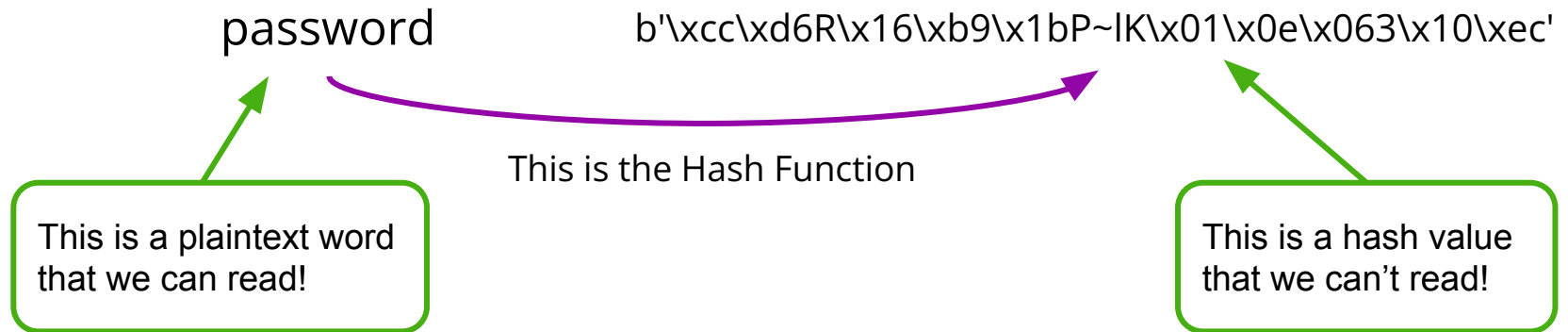
What is Salting?

Password **salting** is a technique to **protect passwords** stored in databases by **adding a string** of 32 or more characters and then hashing them.

Salting **prevents hackers** who breach an enterprise environment from **reverse-engineering passwords** and **stealing** them from the database.

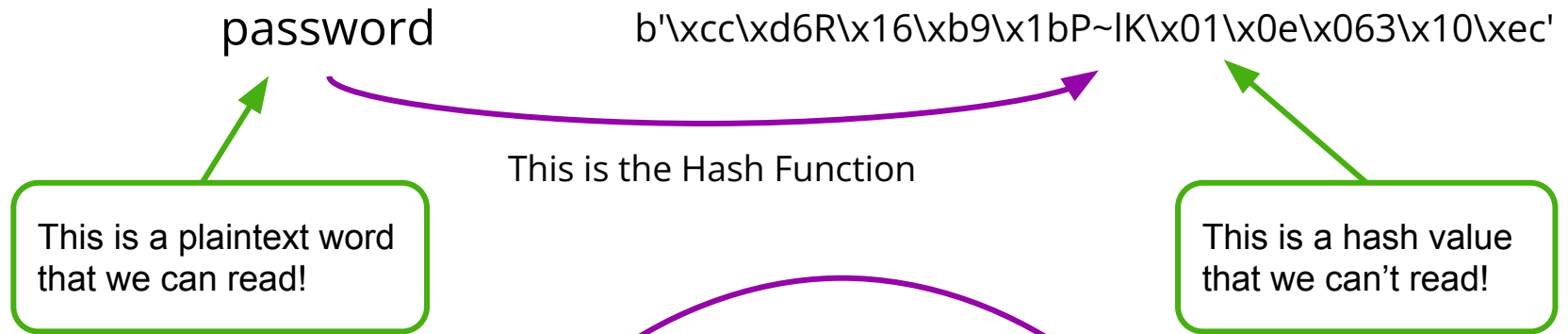
Normal Hashing

Usually when you hash a password it looks like this:

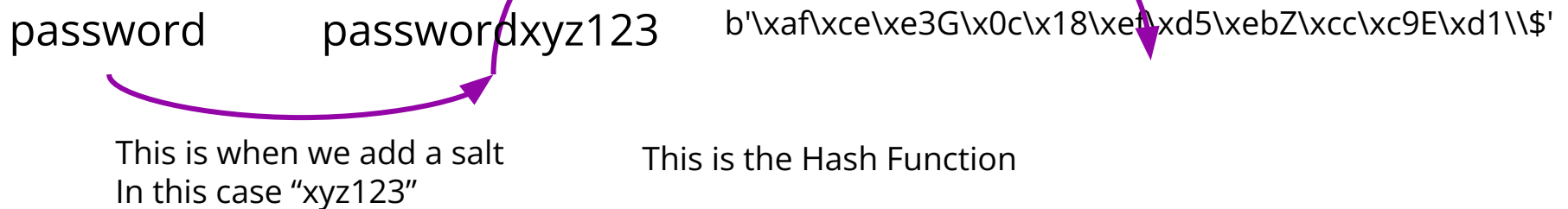


Normal Hashing

Usually when you hash a password it looks like this:




When we add a salt, it looks like this:



Why Salty?

What is the point of adding a salt?


Imagine we have a rainbow table, then we know the hash of a common password like this:

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

Why Salty?

What is the point of adding a salt?

Imagine we have a rainbow table, then we know the hash of a common password like this:

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

But if we add a salt to every password, then a hacker looking for common passwords won't be able to use their rainbow table unless they also know the salt.

passwordxyz123  b'\xaf\xce\xe3G\x0c\x18\xef\xd5\xebZ\xcc\xc9E\xd1\\\$'

Why Salty?

What is the point of adding a salt?

Imagine we have a rainbow table, then we know the hash of a common password like this:

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

But if we add a salt to every password, then a hacker looking for common passwords won't be able to use their rainbow table unless they also know the salt.

passwordxyz123 → b'\xaf\xce\xe3G\x0c\x18\xef\xd5\xebZ\xcc\xc9E\xd1\\\$'

These are not
the same hash!

Project time!

Salting!

Let's put what we learnt into our project
Try to do Parts 0 - 5
of workbook 3!

The tutors will be around to help!