

Welcome to the labs!



Thank you to our Sponsors!

Platinum Sponsor:

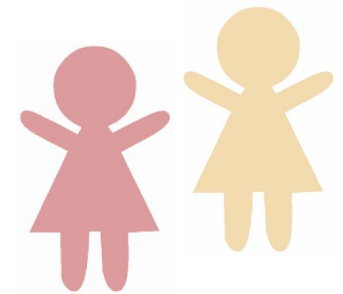
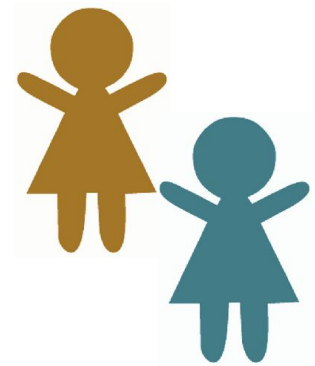


Who are the tutors?

Who are you?

Two Truths and a Lie

1. Get in a group of 3-5 people
2. Tell them three things about yourself:
 - a. Two of these things should be true
 - b. One of these things should be a lie!
3. The other group members have to guess which is the lie



Log on

Log on and jump on the GPN website

girlsprogramming.network/workshop

Click Content for your room. You can see:

- These **slides** (to take a look back or go on ahead).
- A digital copy of your **workbook**.
- Help bits of text you can **copy and paste!**

There's also links to places where you can do more programming!

Tell us you're here!

Click on the
End 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' )
```



Password Cracker!

Today's project is split into 2 main parts!

In workbook 1 we are going to build a program that can make passwords more secure using encoding, then compare an entered password with the actual password to see if it matches!

In workbook 2 we are going to learn about the most common passwords and how hackers utilise this to get into other peoples accounts, and how we can use our new knowledge to find other people's passwords!

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!



Intro to Programming

What is programming?



Programming is not a bunch of crazy numbers!

It's giving computers a set of instructions!



A Special Language

A language to talk to dogs!



Programming is a language to talk to computers

People are smart! Computers are dumb!

SALAD INSTRUCTIONS

Programming is like a recipe!

Computers do EXACTLY what you say, every time.

Which is great if you give them a good recipe!

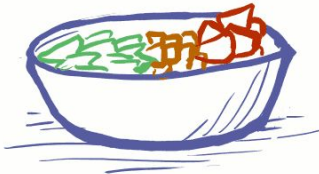
1) GET A LETTUCE HEAD, A CARROT, A TOMATO, A KNIFE, AND A BOWL



2) USE THE KNIFE TO CUT UP THE LETTUCE HEAD, CARROT, AND TOMATO



3) PUT THE LETTUCE, CARROT AND TOMATO IN THE BOWL



4) MIX THE CONTENTS OF THE BOWL



People are smart! Computers are dumb!

But if you get it out of order....

A computer wouldn't know this recipe was wrong!

SALAD INSTRUCTIONS

1) GET A LETTUCE HEAD, A CARROT, A TOMATO, A KNIFE, AND A BOWL



3) PUT THE LETTUCE, CARROT AND TOMATO IN THE BOWL



2) USE THE KNIFE TO CUT UP THE LETTUCE HEAD, CARROT, AND TOMATO



4) MIX THE CONTENTS OF THE BOWL



People are smart! Computers are dumb!

SALAD INSTRUCTIONS



Computers are bad at filling in the gaps!

A computer wouldn't know something was missing, it would just freak out!

Everyone/thing has strengths!



- Understand instructions despite:
 - Spelling mistakes
 - Typos
 - Confusing parts
- Solve problems
- Tell computers what to do
- Get smarter every day



- Does exactly what you tell it
- Does it the same every time
- Doesn't need to sleep!
- Will work for hours on end!
- Get smarter when you tell them how

Intro to Python

Let's get coding!

Where do we program? In Replit!

Go to replit.com

You need to sign up or sign in to start coding

- If you have a **Google** or **Apple account** it's easiest to use that.
- Or use an **email address** you are able to log into.
- If you don't have any of these, ask a tutor for one of our spare replit accounts to use today.

replit

Create a Replit account

Sign up for teachers

+ Create Account

Have an account? [Log In](#)
Trouble signing up? [Get help](#)

By continuing, you agree to Replit's [Terms of Service](#) and [Privacy Policy](#), and to receiving emails with updates.

Continue with Google

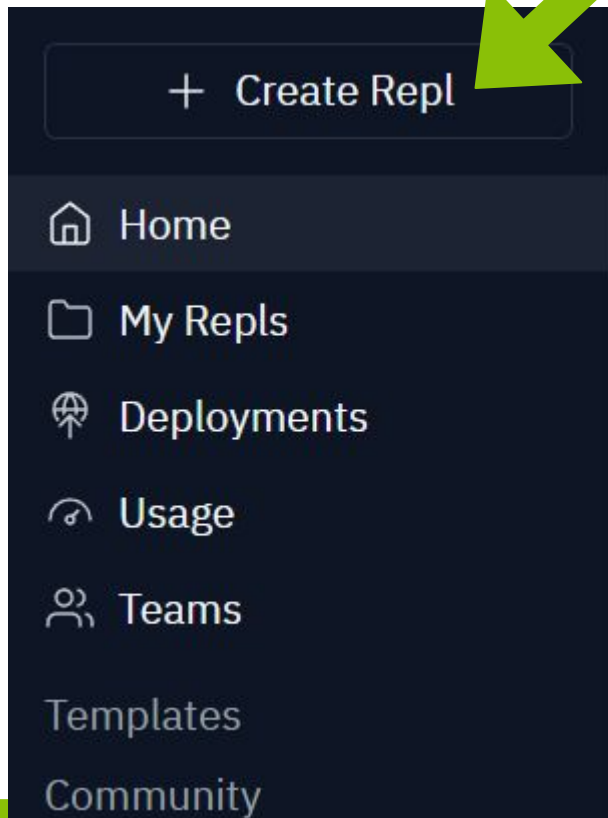
Continue with Github

Continue with Facebook

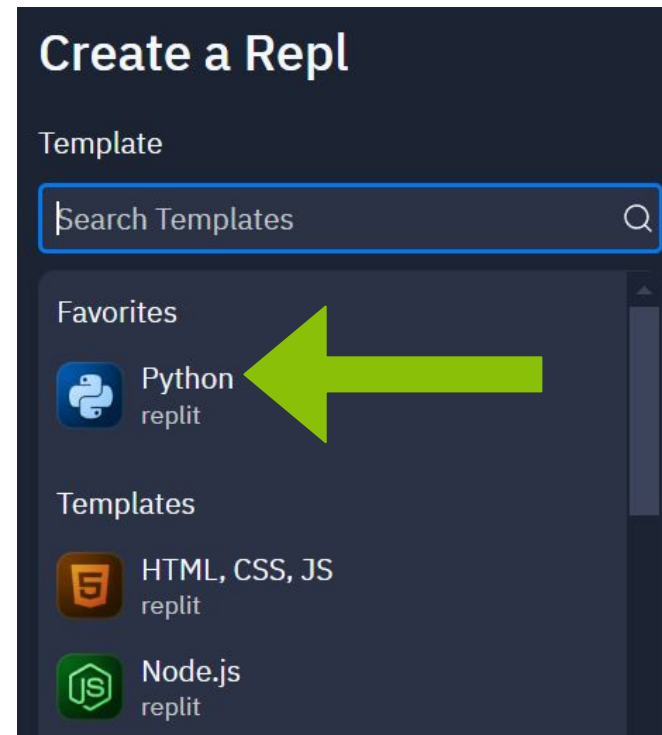
Continue with Apple

Creating our Repl It Project

Let's create a new project



Select Python for the project template

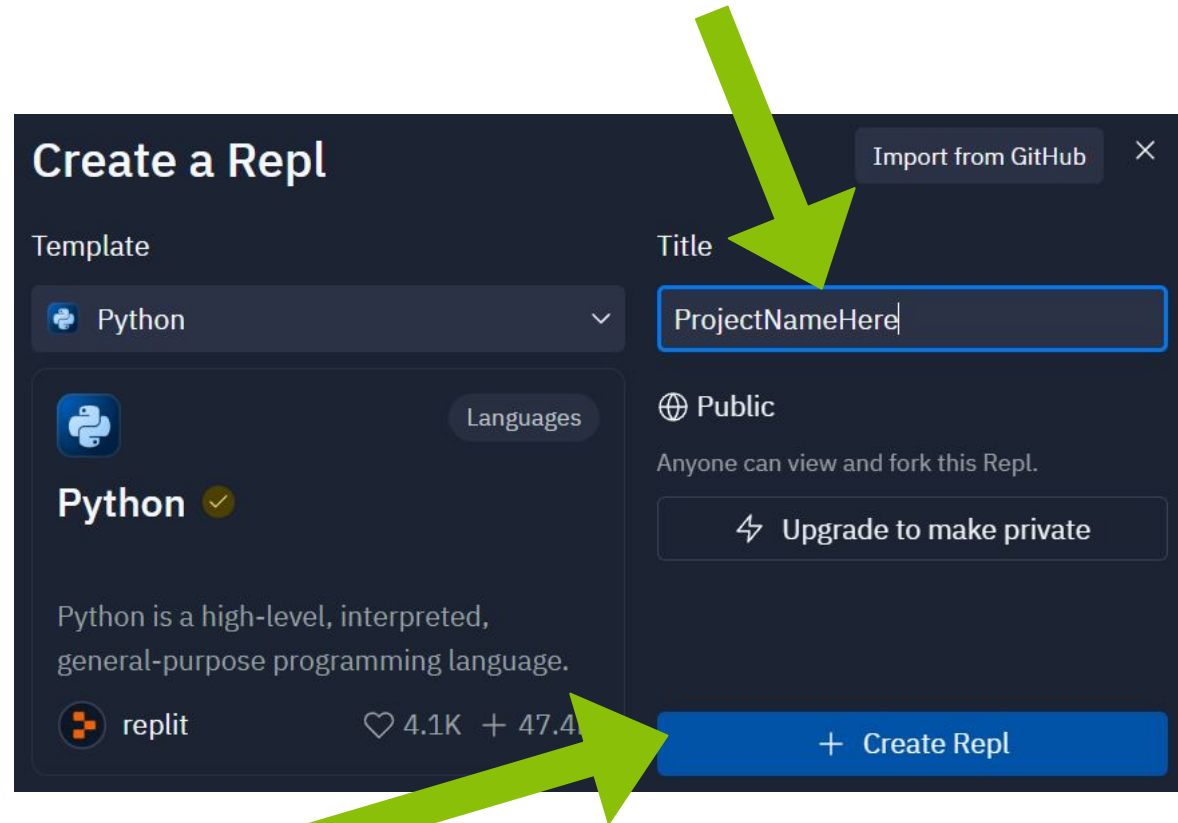


Creating our Repl It Project

**Don't forget to
give your
project a name!**

Name it after
today's project!

Click Create Repl



Setting our Repl It Project

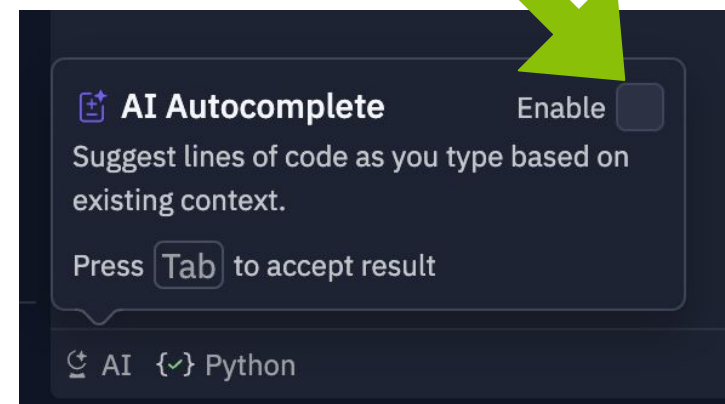
We can't learn if something else is doing all the work!

So we are going to disable AI Autocomplete for this project!



Click the small AI icon in the bottom left corner

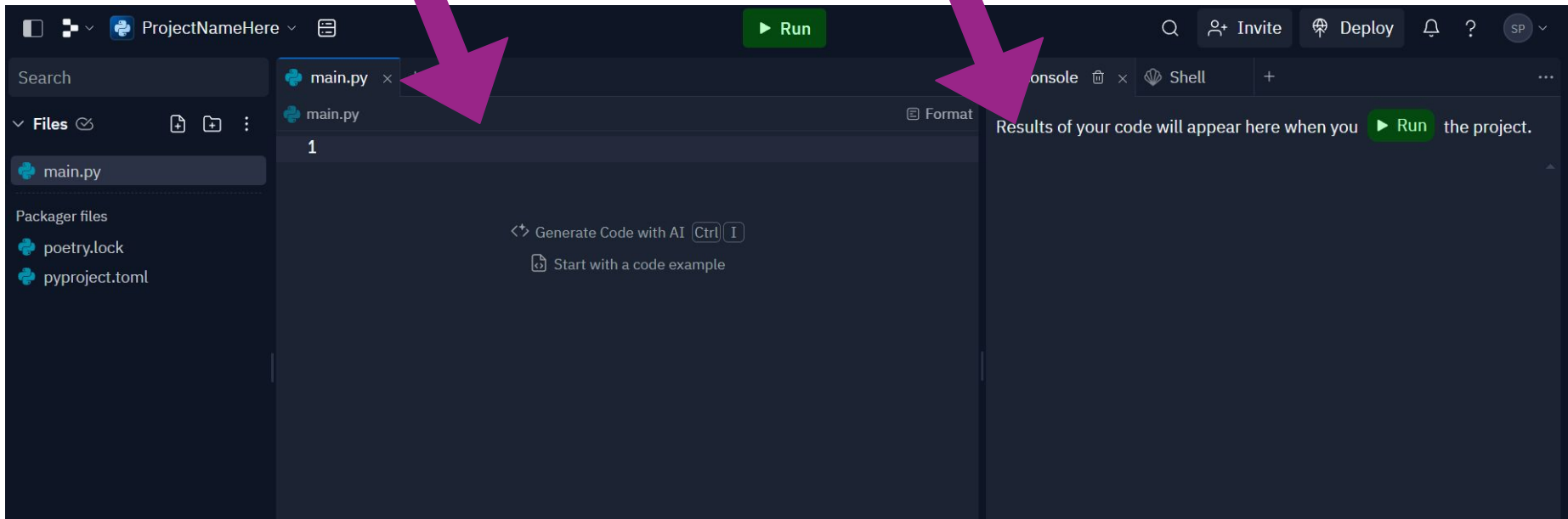
Then sure there is no tick in this box



We're ready to code!

We'll write our project here in main.py

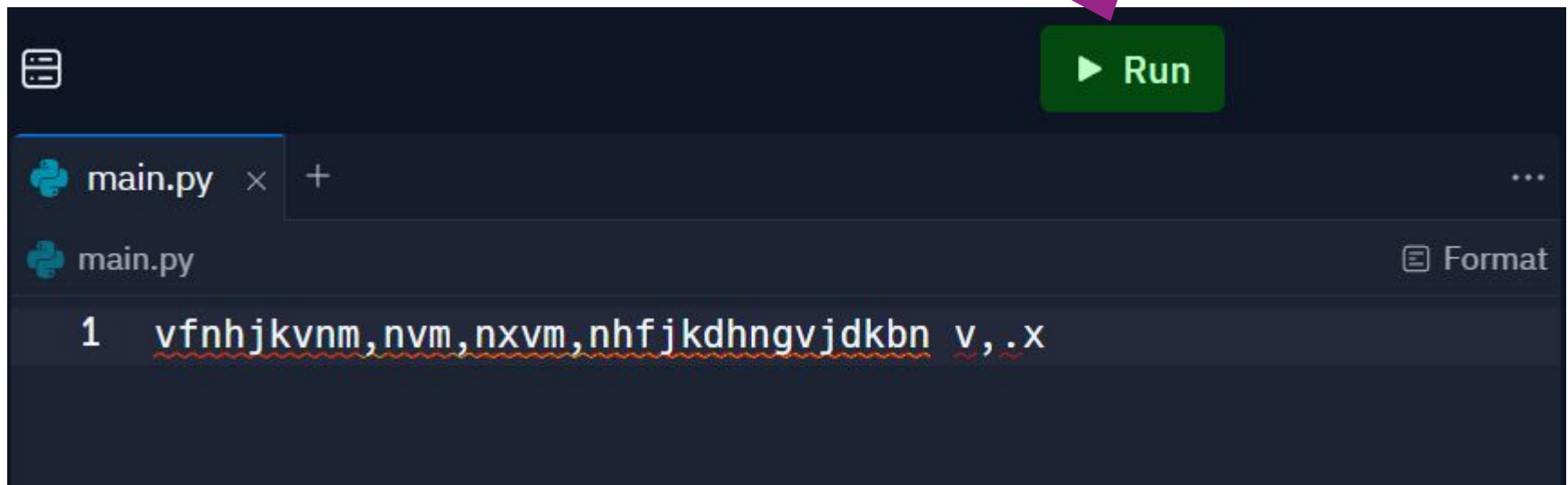
When you run your code, the results will display in the Console here



Run a test! Make a mistake!

Type by **button mashing** the keyboard!

Click Run



```
1 vfnhjkvnm,nvm,nxvm,nhfjkdhngvjdkbn v,.x
```

Did you get an error message in the Console?

Mistakes are great!

*SyntaxError:
Invalid Syntax*

Good work you made an error!

*ImportError:
No module
named humour*

- Programmers make A LOT of errors!
- Errors give us hints to find mistakes
- Run your code often to get the hints!!
- Mistakes won't break computers!



*AttributeError:
'NoneType' object
has no attribute
'foo'*

*TypeError: Can't
convert 'int' object
to str implicitly*

*KeyError:
'Hairy Potter'*



Write some code!!

Type this into the window
Then press enter!

```
print('hello world')
```

Did it print:

hello world

???

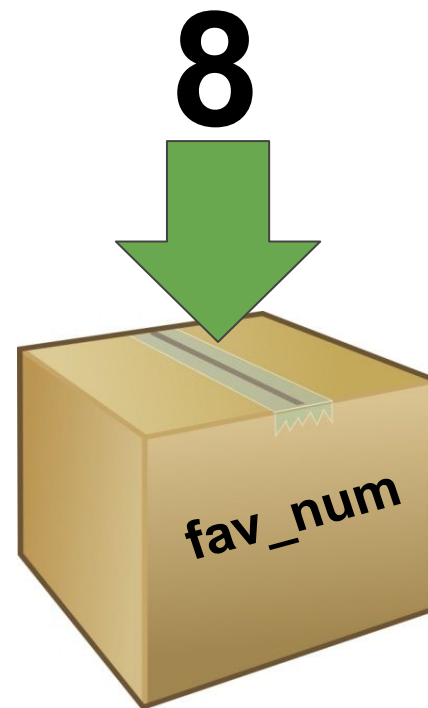
No Storing is Boring!

It's useful to be able to remember things for later!

Computers remember things in "**variables**"

Variables are like putting things into a **labeled cardboard box**.

Let's make our favourite number 8 today!



Math operators in Python

Before we dive into some examples, let's learn some math operators in Python!

| | |
|-----------------|----------|
| Plus | + |
| Minus | - |
| Multiply | * |
| Divide | / |

Variables

Instead of writing the number 8, we can write fav_num.



fav_num - 6

=> _

fav_num * 2

=> _

Variables

Instead of writing the number 8, we can write fav_num.



fav_num - 6

=> 2

fav_num * 2

=> __

Variables

Instead of writing the number 8, we can write fav_num.



fav_num - 6

=> 2

fav_num * 2

=> 16

Variables

Instead of writing the number 8, we can write fav_num.



fav_num - 6

=> 2

fav_num * 2

=> 16

**But writing 8 is
much shorter than
writing fav_num???**



No variables VS using variables

Imagine we want to change the operating number from 8 to 102:



4
Changes



1
Change

| | | |
|--------|---|----------|
| 8 - 6 | → | 102 - 6 |
| 8 * 2 | → | 102 * 2 |
| 8 + 21 | → | 102 + 21 |
| 8 / 2 | → | 102 / 2 |

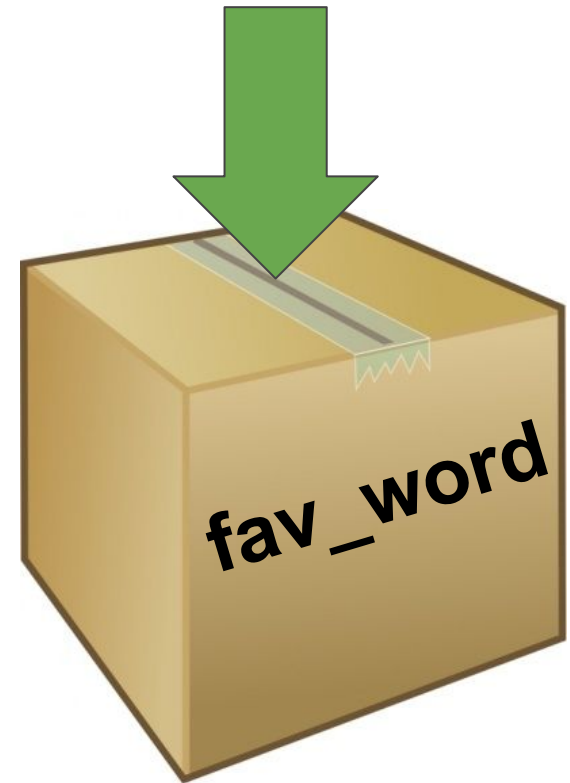
| | | |
|-----------------|---|-------------------|
| int fav_num = 8 | → | int fav_num = 102 |
| fav_num - 6 | → | fav_num - 6 |
| fav_num * 2 | → | fav_num * 2 |
| fav_num + 21 | → | fav_num + 21 |
| fav_num / 2 | → | fav_num / 2 |

Variables

Variables can store more than numbers

Try store a string in
`fav_word`

“Hello”



Variables

Instead of writing the string "Hello",
we can write fav_word:



fav_word + "World"

=> _____

fav_word * 2?

=> _____

Variables

Instead of writing the string “Hello”,
we can write fav_word:



fav_word + “World”
=> **“HelloWorld”**

fav_word * 2?
=> _____

Variables

Instead of writing the string “Hello”,
we can write fav_word:



fav_word + “World”
=> **“HelloWorld”**

fav_word * 2?
=> **“HelloHello”**

Asking a question!

it's more fun when we get to interact with the computer!

Let's learn about input!

```
>>> my_name = input('What is your name? ')
>>> print('Hello ' + my_name)
```

How input works!

Store the answer
in the variable
my_name

Writing input
tells the
computer to wait
for a response

This is the
question you
want printed to
the screen

```
>>> my_name = input('What is your name? ')  
>>> print('Hello ' + my_name)
```

We use the answer
that was stored in the
variable later!

Adding a comment!

Sometimes we want to write things in our file that the computer doesn't look at! **We can use "Comments" for that!**

Sometimes we want to write a note for people to read

```
# This code was written by Vivian
```

And sometimes we want to not run some code (but don't want to delete it!)

```
# print("Goodbye world!")
```

Project time!

Now you can give the computer variables!

Let's put what we learnt into our project

Try to do Part 0 - 1

The tutors will be around to help!

If Statements

Conditions!

Conditions let us make decision.
First we test if the condition is met!
Then maybe we'll do the thing



If it's raining take an umbrella

Yep it's raining

..... take an umbrella

Booleans (True and False)

Computers store whether a condition is met in the form of
True and **False**

To figure out if something is **True** or **False** we do a comparison

What do you think these are? True or False?

`5 < 10`

`5 != 5`

`"Dog" == "dog"`

`"D" in "Dog"`

Booleans (True and False)

Computers store whether a condition is met in the form of
True and **False**

To figure out if something is **True** or **False** we do a comparison

What do you think these are? True or False?

`5 < 10`

True

`"Dog" == "dog"`

`5 != 5`

`"D" in "Dog"`

Booleans (True and False)

Computers store whether a condition is met in the form of
True and **False**

To figure out if something is **True** or **False** we do a comparison

What do you think these are? True or False?

$5 < 10$

True

"Dog" == "dog"

$5 \neq 5$

False

"D" in "Dog"

Booleans (True and False)

Computers store whether a condition is met in the form of
True and **False**

To figure out if something is **True** or **False** we do a comparison

What do you think these are? True or False?

`5 < 10`

True

`"Dog" == "dog"`

False

`5 != 5`

False

`"D" in "Dog"`



Booleans (True and False)

Computers store whether a condition is met in the form of
True and **False**

To figure out if something is **True** or **False** we do a comparison

What do you think these are? True or False?

$5 < 10$

True

"Dog" == "dog"

False

$5 \neq 5$

False

"D" in "Dog"

True

Conditions

So to know whether to do something, they find out if it's **True!**

```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
```

Conditions

So to know whether to do something, they find out if it's **True!**

```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
```

That's the
condition!

Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if True:
    print("that's a small number")
```

Is it **True** that fave_num is less than 10?

- Well, fave_num is 5
- And it's **True** that 5 is less than 10
- So it is **True**!

Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if True:
    print("that's a small number")
```

The condition is True, and what happen?

Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if True:
    print("that's a small number")
```

The condition is True, we run the code!

```
>>> that's a small number
```

Conditions

How about a different number???

```
fave_num = 9000  
if fave_num < 10:  
    print("that's a small number")
```



That's the
condition!



Conditions

It's **False**!

```
fave_num = 9000
if False:
    print("that's a small number")
```


Conditions

It's **False**!

```
fave_num = 9000  
if False :  
    print("that's a small number")
```

The condition is False, and what happen?

Conditions

```
fave_num = 9000
if False:
    print("that's a small number")
```

The condition is False, and what happens?



Nothing!

If statements

```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
```

This line ...

... controls this line

If statements

Actually

```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
    print("and I like that")
    print("A LOT!!")
```

This line ...



... controls anything below it
that is indented like this!



If statements

```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
    print("and I like that")
    print("A LOT!!")
```

```
>>> that's a small number
>>> and I like that
>>> A LOT!!
```

If statements

```
word = "GPN"  
if word == "GPN":  
    print("GPN is awesome!")
```

What happens??

If statements

```
word = "GPN"  
if word == "GPN":  
    print("GPN is awesome!")
```

What happens??

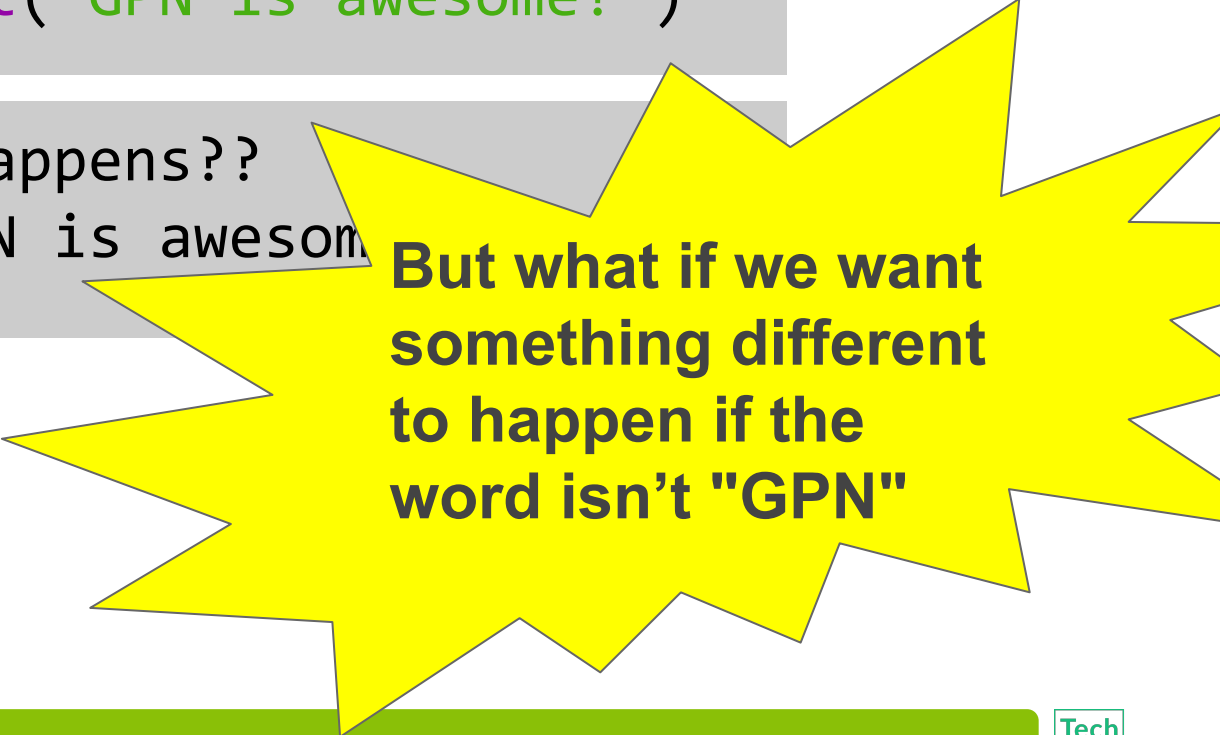
```
>>> GPN is awesome!
```

Else statements

```
word = "GPN"  
if word == "GPN":  
    print("GPN is awesome!")
```

What happens??

```
>>> GPN is awesome!
```



But what if we want something different to happen if the word isn't "GPN"

Else statements

else
statements
means something
still happens if
the **if** statement
was **False**

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

What happens??

Else statements

else
statements
means something
still happens if
the **if** statement
was **False**

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

What happens??

```
>>> The word isn't GPN :(
```

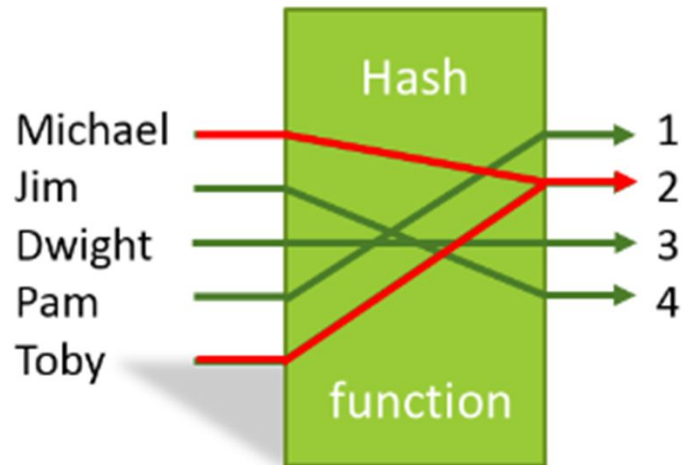
Project Time!

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

Let's put what we learnt into our project
Try to do Part 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?


Explain what hashing does?????

What is Hashing?

How does it work?

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

password

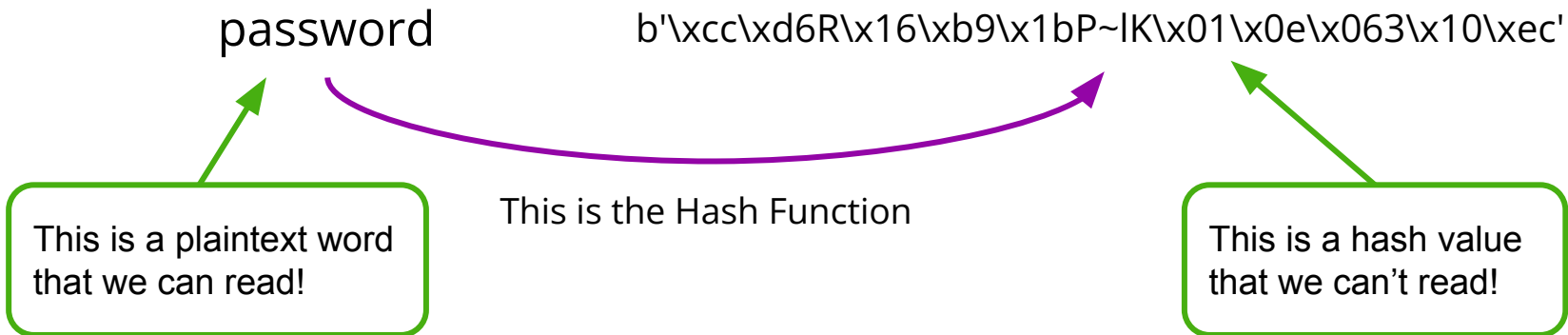


This is a plaintext word
that we can read!

What is Hashing?

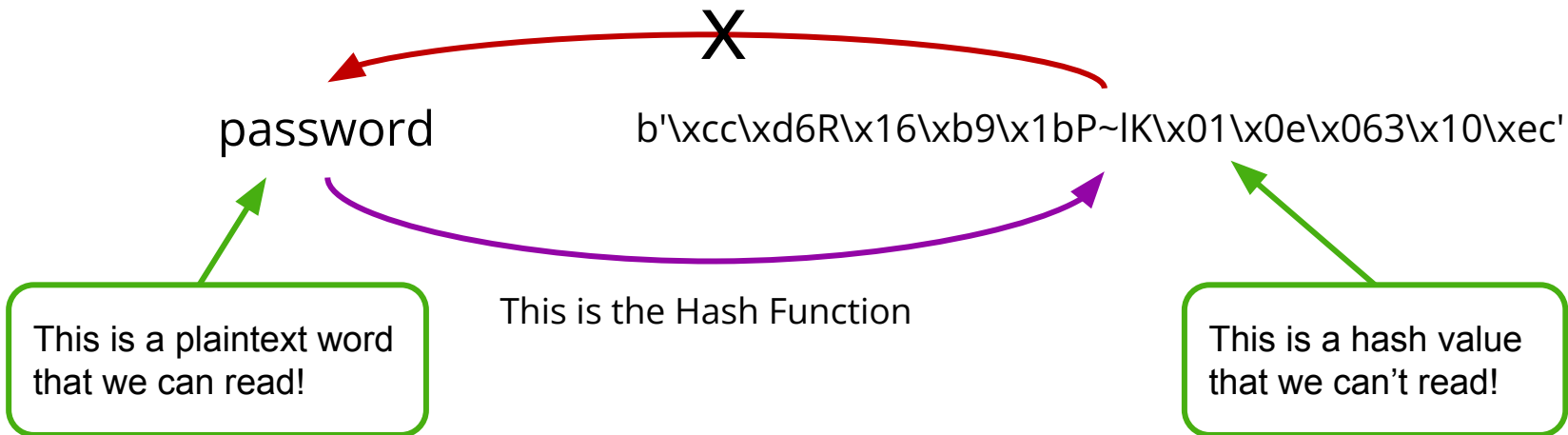
How does it work?

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
```

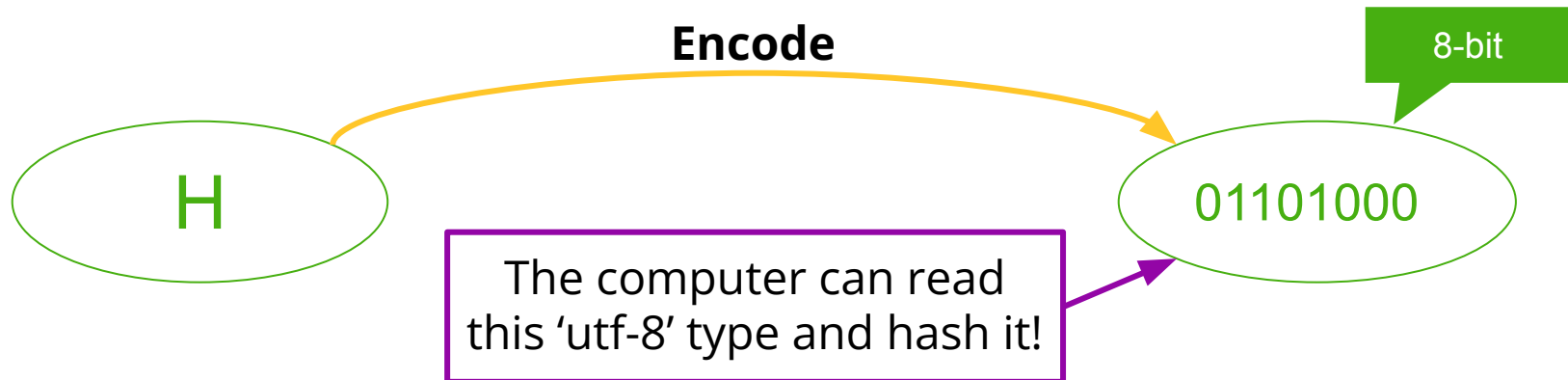
at the top of our code!

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 '
```

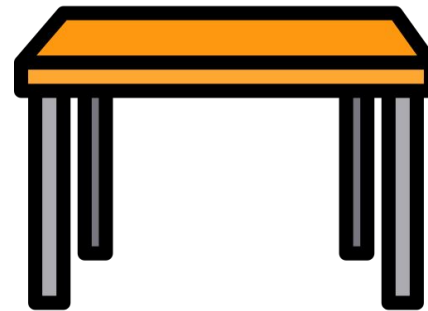
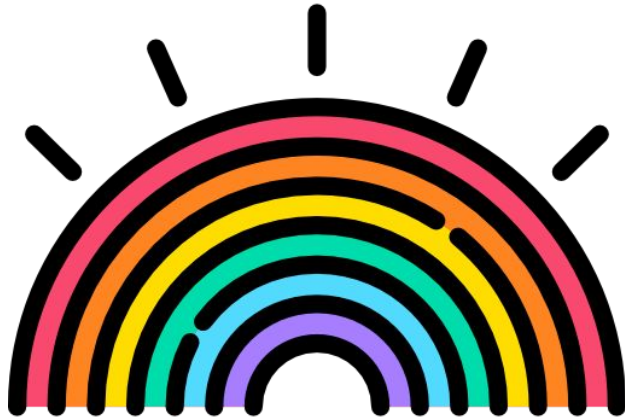
Project Time!

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?

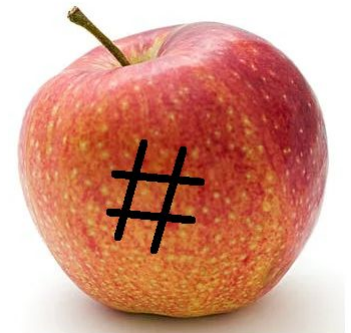
We could **guess** the password!

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\x98\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\xf5\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\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\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\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? Yes! Now we can get the plaintext.

| 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? Yes! Now we can get the plaintext. It's 1234567890

| 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> |

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 common passwords. 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 |

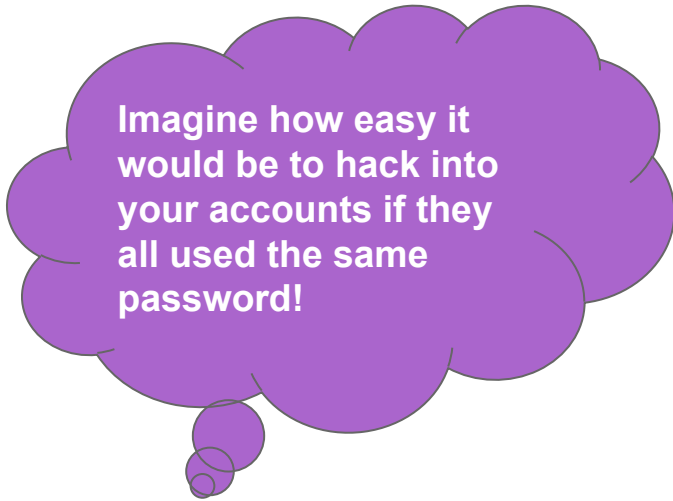
[src: https://www.upguard.com/blog/biggest-data-breaches](https://www.upguard.com/blog/biggest-data-breaches)

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!

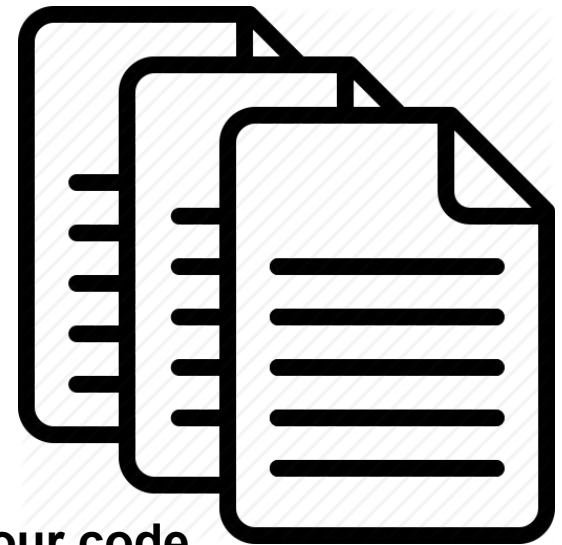
We can store information inside files.

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!

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!

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
```

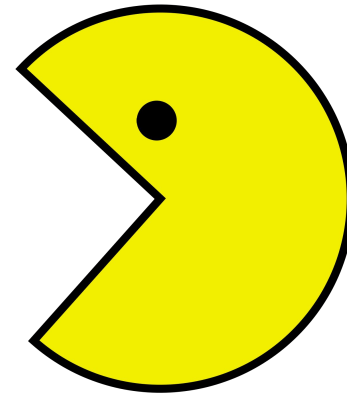
```
World
```

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!

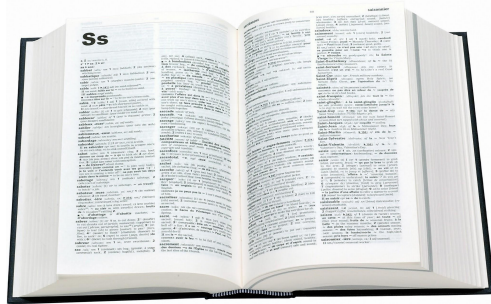
**Try to do Part 1
of workbook 2!!**

The tutors will be around to help!

Dictionaries



Dictionaries!

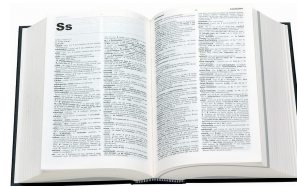


You know dictionaries!

**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

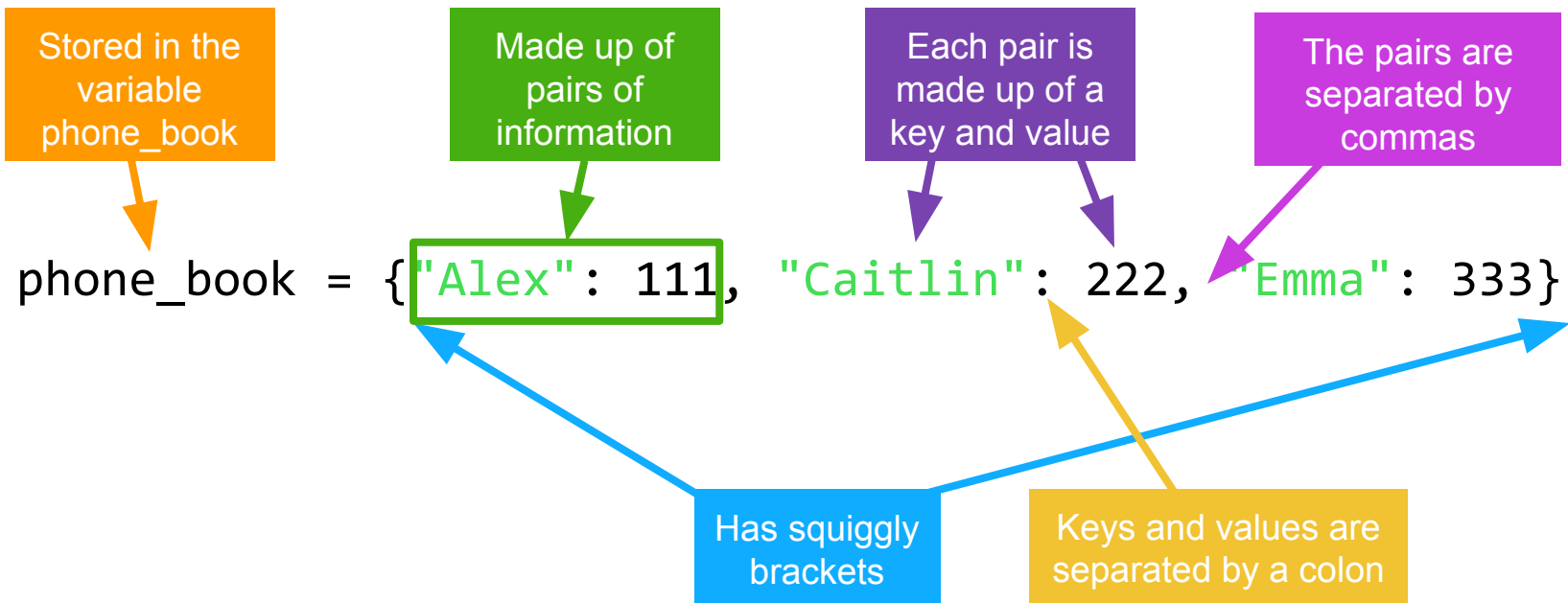
↑
Key

↑
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!

How does it work??

Everything in the list gets to have a turn at being the fruit variable




```
fruits = ['apple', 'banana', 'mango']  
▶ for fruit in fruits:  
    print('yummy ' + fruit)
```


Let's set fruit to to the **first** thing in the list!
fruit is now 'apple'!

How does it work??

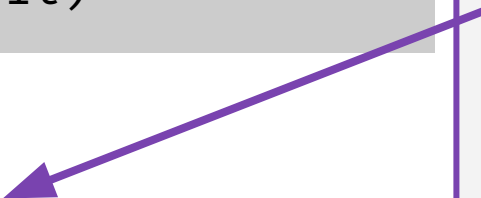
Everything in the list gets to have a turn at being the fruit variable



```
fruits = ['apple', 'banana', 'mango']  
for fruit in fruits:  
    print('yummy ' + fruit)
```



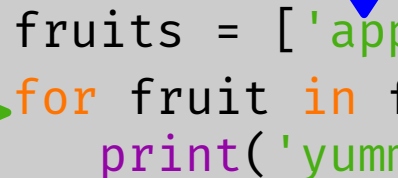
Let's set fruit to to the **first** thing in the list!
fruit is now 'apple'!
print('yummy ' + fruit)



```
>>> Yummy apple
```

How does it work??

Everything in the list gets to have a turn at being the fruit variable



```
fruits = ['apple', 'banana', 'mango']  
for fruit in fruits:  
    print('yummy ' + fruit)
```


```
>>> Yummy apple
```

Let's set fruit to to the **first** thing in the list!
fruit is now 'apple'!
print('yummy ' + fruit)
We're at the end of the loop body, back to the top!



How does it work??

Everything in the list gets to have a turn at being the fruit variable



```
fruits = ['apple', 'banana', 'mango']  
for fruit in fruits:  
    print('yummy ' + fruit)
```


```
>>> Yummy apple
```

Let's set fruit to to the **first** thing in the list!
fruit is now 'apple'!
print('yummy ' + fruit)
We're at the end of the loop body, back to the top!


Let's set fruit to to the **next** thing in the list!
fruit is now 'banana'!

How does it work??


Everything in the list gets to have a turn at being the fruit variable



```
fruits = ['apple', 'banana', 'mango']  
for fruit in fruits:  
    print('yummy ' + fruit)
```



```
>>> Yummy apple  
>>> Yummy banana
```

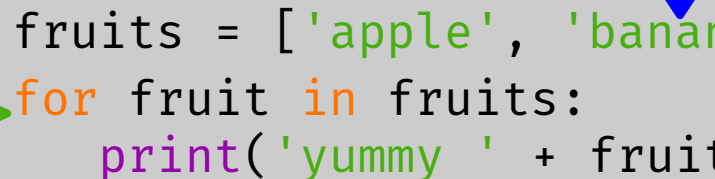


Let's set fruit to to the **first** thing in the list!
fruit is now 'apple'!
print('yummy ' + fruit)
We're at the end of the loop body, back to the top!

Let's set fruit to to the **next** thing in the list!
fruit is now 'banana'!
print('yummy ' + fruit)

How does it work??

Everything in the list gets to have a turn at being the fruit variable



```
fruits = ['apple', 'banana', 'mango']  
for fruit in fruits:  
    print('yummy ' + fruit)
```

```
>>> Yummy apple
```

```
>>> Yummy banana
```

Let's set fruit to to the **first** thing in the list!

fruit is now 'apple'!

```
print('yummy ' + fruit)
```

We're at the end of the loop body, back to the top!

Let's set fruit to to the **next** thing in the list!

fruit is now 'banana'!

```
print('yummy ' + fruit)
```

Out of body, back to the top!



How does it work??

Everything in the list gets to have a turn at being the fruit variable

```
fruits = ['apple', 'banana', 'mango']  
for fruit in fruits:  
    print('yummy ' + fruit)
```

```
>>> Yummy apple  
>>> Yummy banana  
>>> Yummy mango
```

Let's set fruit to to the **first** thing in the list!
fruit is now 'apple'!
print('yummy ' + fruit)
We're at the end of the loop body, back to the top!

Let's set fruit to to the **next** thing in the list!
fruit is now 'banana'!
print('yummy ' + fruit)
Out of body, back to the top!

Let's set fruit to to the **next** thing in the list!
fruit is now 'mango'!
print('yummy ' + fruit)

How does it work??

Everything in the list gets to have a turn at being the fruit variable

```
fruits = ['apple', 'banana', 'mango']  
for fruit in fruits:  
    print('yummy ' + fruit)
```

```
>>> Yummy apple  
>>> Yummy banana  
>>> Yummy mango
```



Let's set fruit to to the **first** thing in the list!
fruit is now 'apple'!
print('yummy ' + fruit)
We're at the end of the loop body, back to the top!

Let's set fruit to to the **next** thing in the list!
fruit is now 'banana'!
print('yummy ' + fruit)
Out of body, back to the top!

Let's set fruit to to the **next** thing in the list!
fruit is now 'mango'!
print('yummy ' + fruit)

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!

Tell us what you think!

Click on the
End of Day Form
and fill it in now!