Welcome to GPN

Thank you to our Sponsors!

Platinum Sponsor:

ATLASSIAN

amazon

Gold Sponsor:

Commonwealth Bank

Who are the tutors?

Who are you?

Log on

Log on and jump on the GPN website

girlsprogramming.network/workshop

Click on your node location

Click on your room.

From this page you can see:

- These **slides** (to take a look back or go on ahead).
- A link to your **workbook** in EdStem
- Other helpful bits to use through the day!

Tell us you’re here!

Click on the

Start of Day Survey

and fill it in now!

Today’s project!

Markov Chains!

What is a Markov Chain?

A Markov chain is a simple Artificial Intelligence!

Let’s play a game with some cups to help explain it

Let’s play the cups game!

Let’s generate some text in the style of

Green Eggs & Ham by Dr Seuss

Do you like green eggs and ham?

I do not like them, Sam-I-am.

I do not like green eggs and ham.

Would you like them here or there?

I would not like them here or there.

I would not like them anywhere.

Let’s play the cups game!

- Each cup is **labelled** with a word from Green Eggs and Ham

- Each cup **contains** the words that follow the "label" word in Green Eggs and Ham

We’re going to write some text by randomly choosing a next word based on the word before it

Let's play the cups game!

Read the outside of your cup!

If someone shouts the word on the outside of your cup:

1. Pick a piece of paper from inside your cup
2. Shout out the word on the piece of paper
3. Put the piece of paper back in your cup

A tutor will write the words called out on the board

Today we'll be making Markov Chains!

Markov chains are exactly what we just did with the cups!

Today we'll make the computer do it to make some crazy stories!!

Here's one we made from some Shakespeare!

```
doth stay! All days when I compare thee to unseeing eyes
be blessed made By chance, or eyes can see, For all the
top of happy show thee in dark directed. Then thou, whose
shadow shadows doth stay! All days when I compare thee in
your self in inward worth nor outward fair, Can make
bright, How would thy shade Through heavy sleep on the eye
of life repair, Which this, Time's pencil, or my pupil
pen, Neither in the living day, When in eternal lines of
that fair from fair thou grow'st, So should the lines to a
summer's day?
```

Imagine if you used one of these to do your homework!!

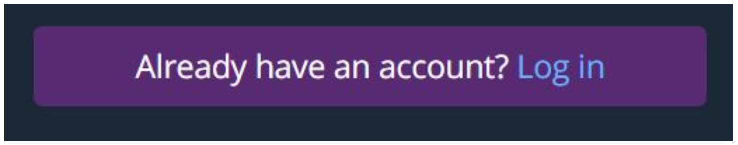
Introduction to Edstem

Log on

Click on your **Workbook** link to take you into EdStem

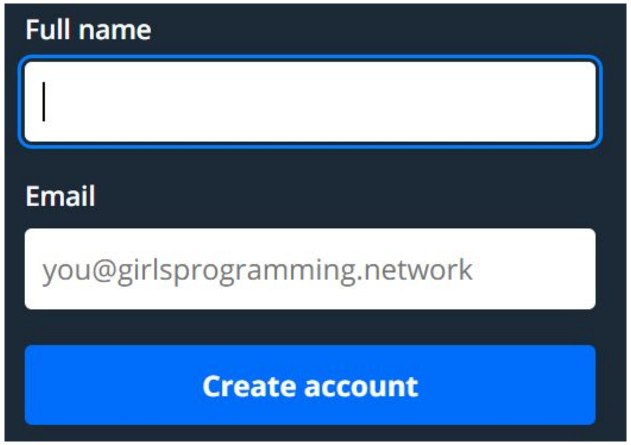
Signing up to Edstem

Log in if you already have a an "Edstem" account from a past GPN

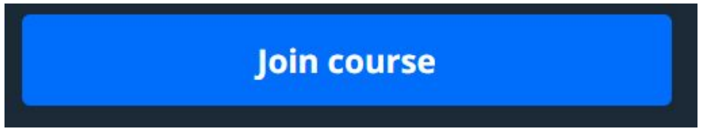


If you haven't got an account, let's make one:

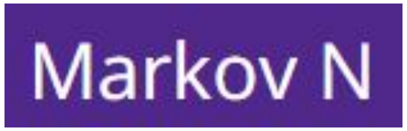
1. Type in your Full Name
2. Type in your personal email
3. Click Create Account
4. Go to your email and verify your new account
5. Create a password



Click Join Course



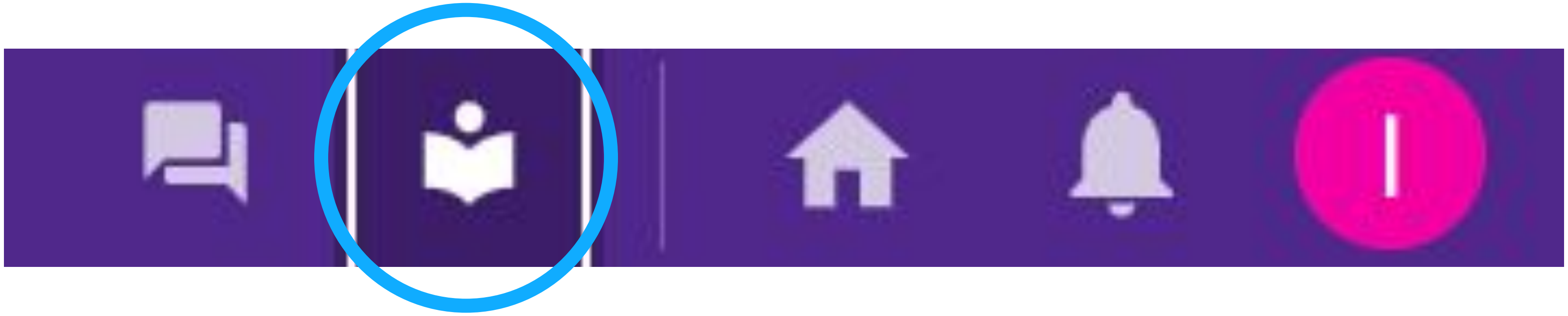
The name of your course will be at the top : →



If you don't have access to your email account, ask a tutor for a GPN Edstem login

Getting to the lessons

1. Once you are in the course, you'll be taken to a discussion page.
2. Click the button for the lessons page (top right - looks like a book)



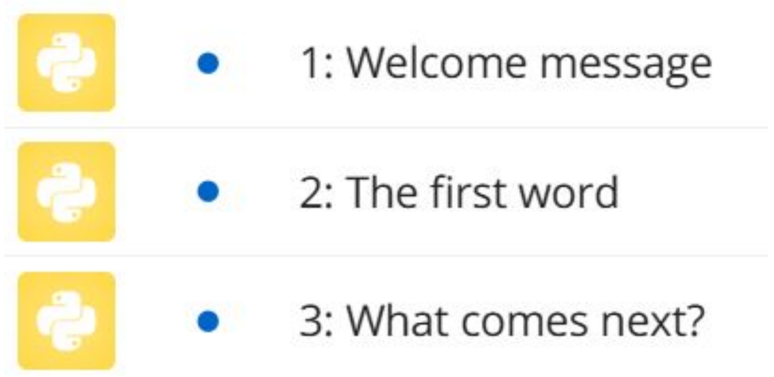
The set up of the workbook

The main page:

1. Heading at the top that tells you the project you are in

2. List of "Chapters" called something like **1:Welcome Message**
   They have an icon that looks like this:

3. To complete your project, work through the chapters one at a time



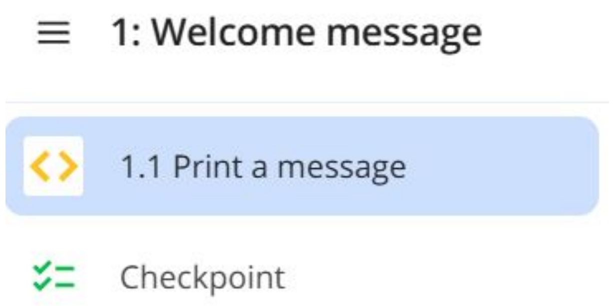
1: Welcome message
2: The first word
3: What comes next?

Inside a Chapter

Inside a Chapter there are two main types of pages:

- **Lessons** where you will do your coding.
  - They have this icon: <>
- **Checkpoints** ✓ Checkpoint

  Each chapter has a checkpoint to complete to move to the next chapter. Make sure you scroll down to see all the questions in a checkpoint.



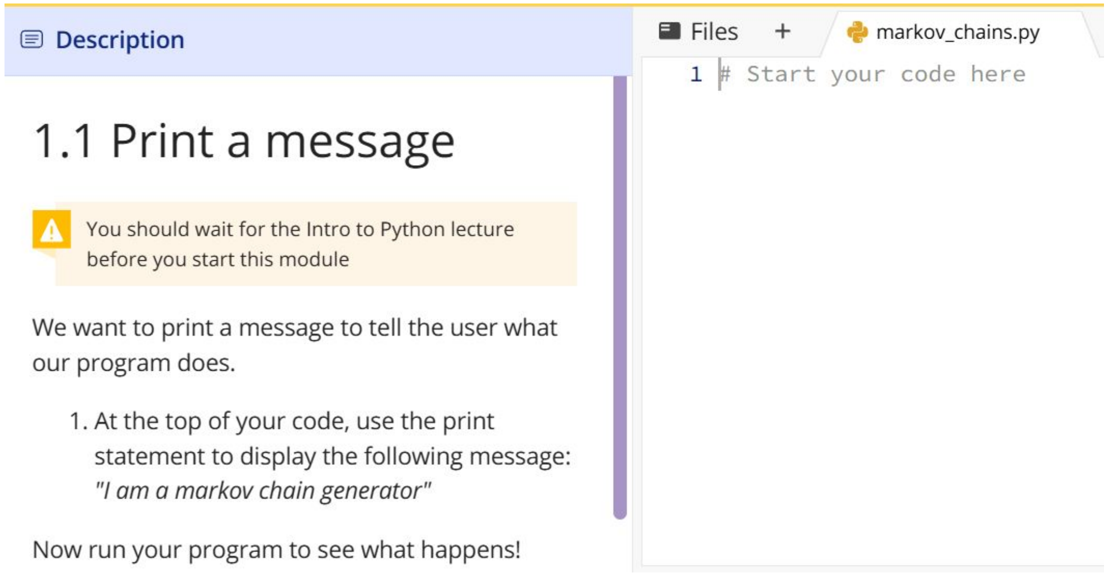
There may also be **Bonus Lessons** to try if you want to or if you are waiting for the next lecture

How to do the work

In each Lesson there is:

1. A section on the left with instructions
2. A section on the right for your code

You will need to **copy your code from the last lesson**, then follow the instructions to change your code



Description

Files + markov_chains.py

```
# Start your code here
```

1.1 Print a message

You should wait for the Intro to Python lecture before you start this module

We want to print a message to tell the user what our program does.

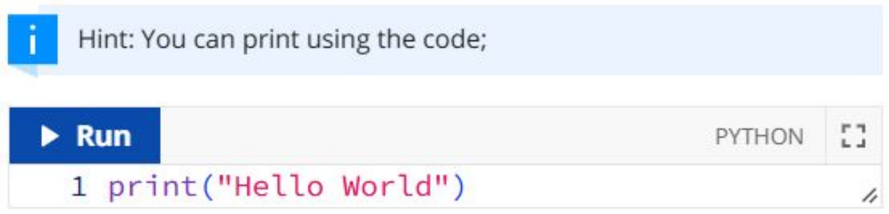
1. At the top of your code, use the print statement to display the following message: *"I am a markov chain generator"*

Now run your program to see what happens!

There are also Hints and Code Blocks to help you

Hints

Sometimes in a lesson, there's some code we want you to do that might be a bit tricky, to help you out we've added some hints. They look like this:



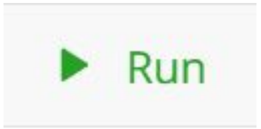
Hint: You can print using the code;

```python
print("Hello World")
```

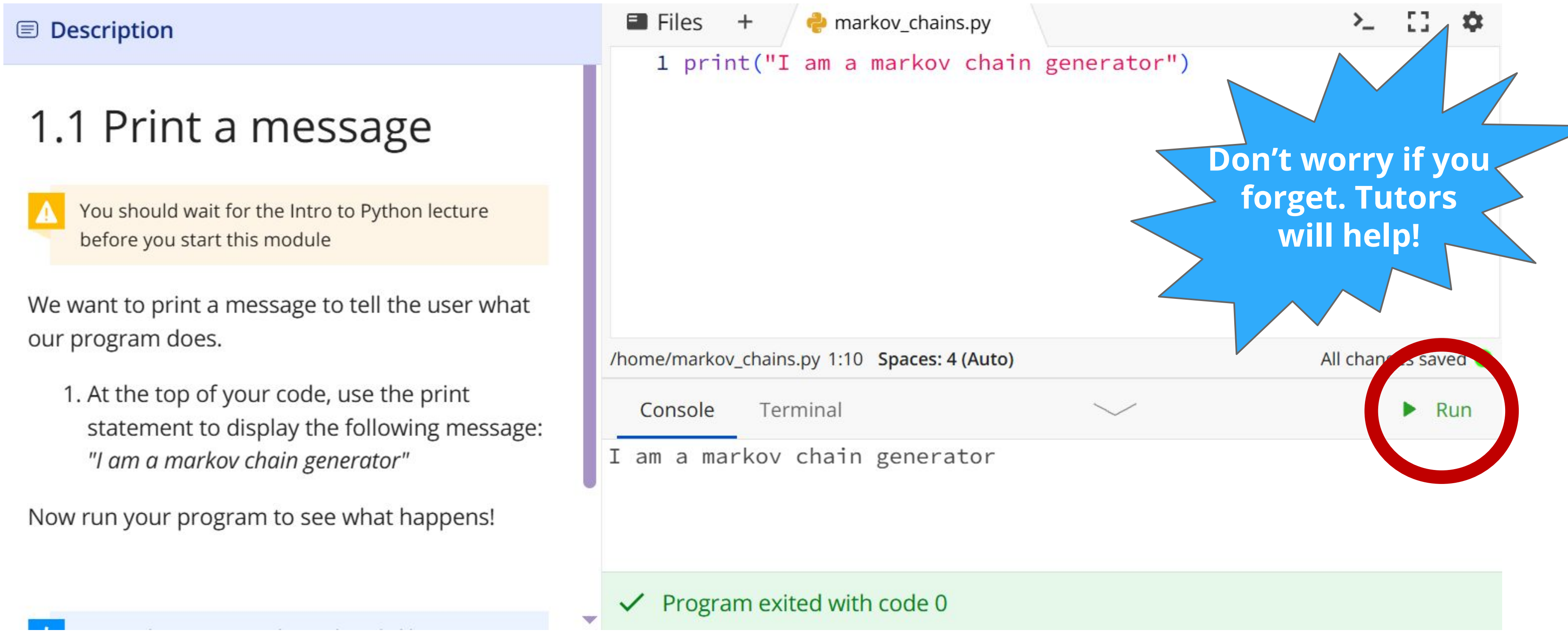
If you press the blue run button it will show you what that code does, you can even change the code to see if/how it changes.

These are **<u>just hints</u>** make sure you're not copying the hint into your code as it will likely end up breaking. They are just to show you the kinds of things you can do.

Running your code...

Click  in the bottom right hand corner

Your code will run and any output will display in the Console



1.1 Print a message

You should wait for the Intro to Python lecture before you start this module

We want to print a message to tell the user what our program does.

1. At the top of your code, use the print statement to display the following message: *"I am a markov chain generator"*

Now run your program to see what happens!

```
print("I am a markov chain generator")
```

/home/markov_chains.py 1:10 **Spaces: 4 (Auto)** All changes saved

```
I am a markov chain generator
```

✓ Program exited with code 0

Some shortcuts...

There are a couple things you can do to make copying your code from one page to another easier.

1. **Ctrl + A**	Pressing these keys together will select all the text on a page
2. **Ctrl + C**	Pressing these keys together will copy anything that's selected
3. **Ctrl + V**	Pressing these keys together will paste anything you've copied

On Macs use Command (⌘) instead of Ctrl

Project time!

You now know all about the EdStem!

You should now sign up and join our EdStem class.

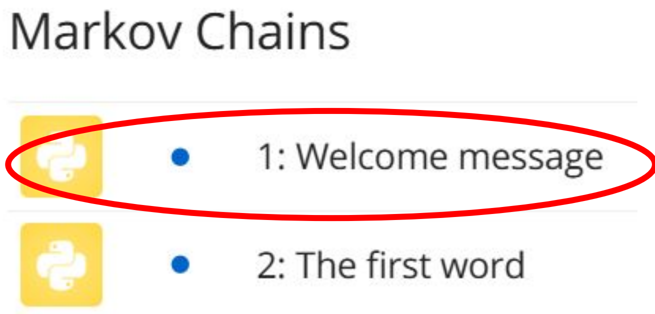
Remember the tutors will be around to help!

Intro to Python

Let's get coding!

Let's make a mistake!

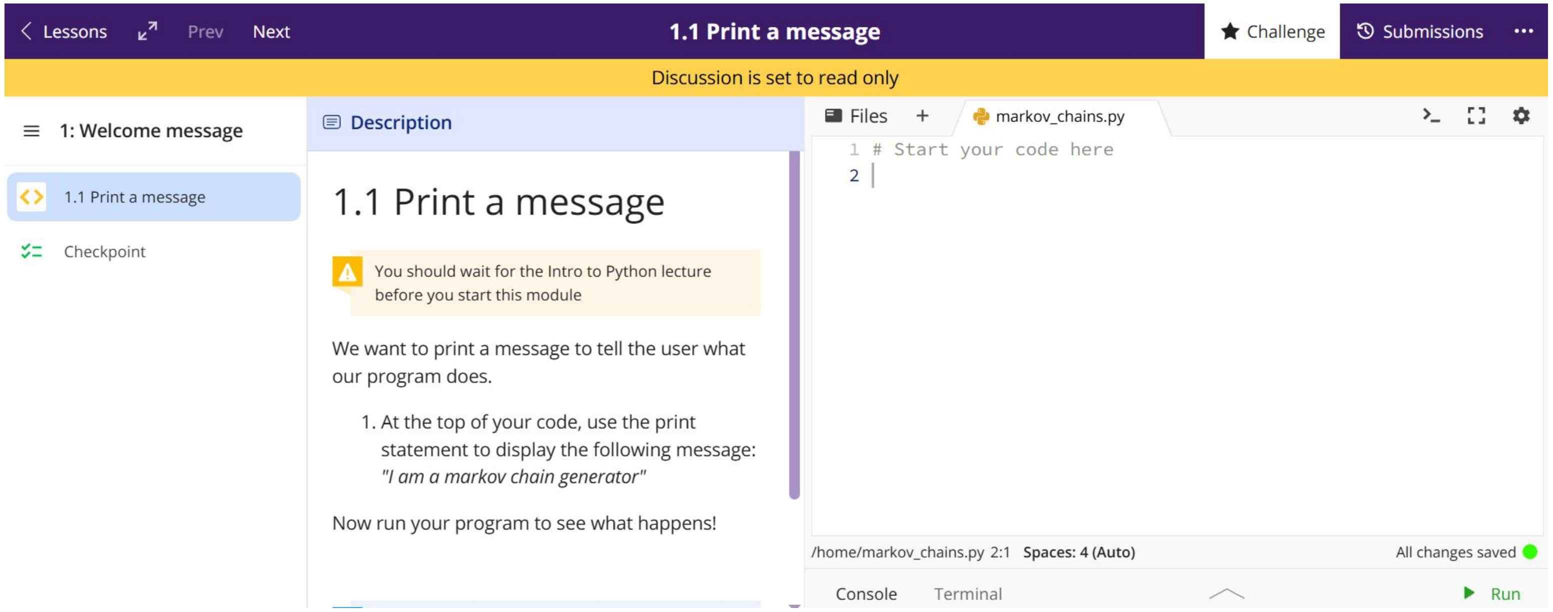
Click on Chapter 1 **'Welcome message'**


Markov Chains

1: Welcome message

2: The first word

The first lesson '1.1 Print a message' will open. It looks like this



1.1 Print a message

You should wait for the Intro to Python lecture before you start this module

We want to print a message to tell the user what our program does.

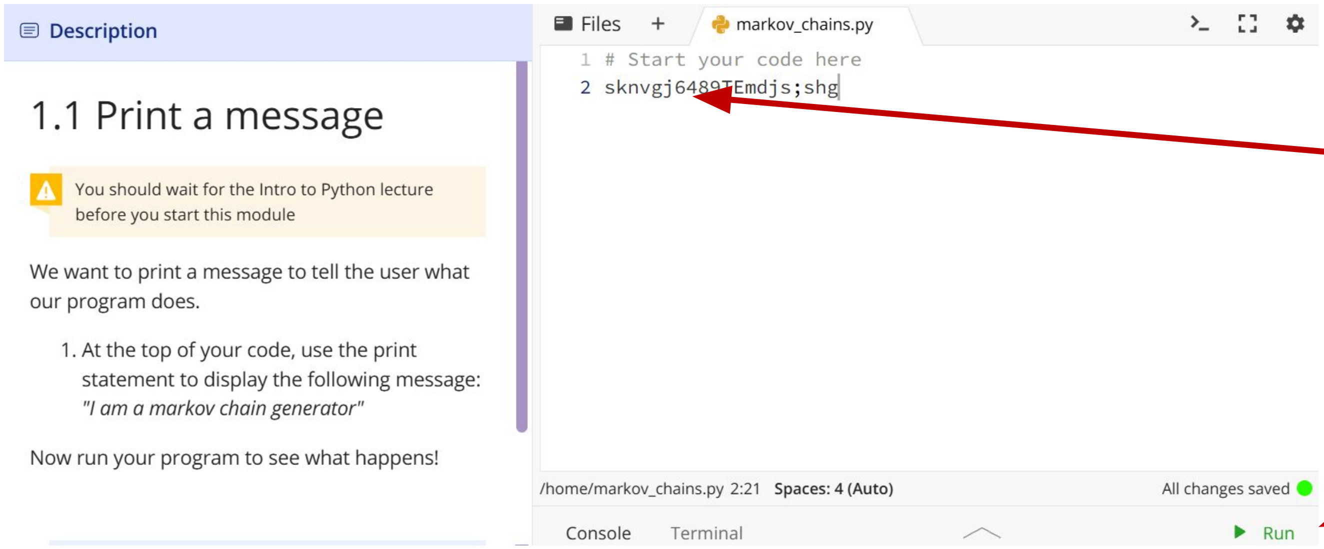
1. At the top of your code, use the print statement to display the following message: *"I am a markov chain generator"*

Now run your program to see what happens!

```
# Start your code here

```

Let's make a mistake!



Description

1.1 Print a message

You should wait for the Intro to Python lecture before you start this module

We want to print a message to tell the user what our program does.

1. At the top of your code, use the print statement to display the following message: *"I am a markov chain generator"*

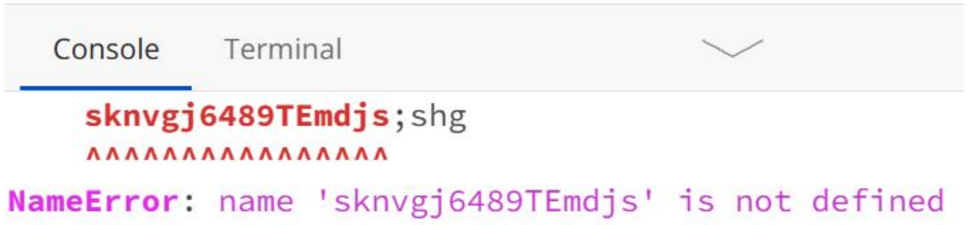
Now run your program to see what happens!

```
# Start your code here
sknvgj6489TEmdjs;shg
```

Type by **button mashing** the keyboard here - type anything you want

Click Run here to run your code!

Did you get a big ugly error message?



```
sknvgj6489TEmdjs;shg
^^^^^^^^^^^^^^^^
NameError: name 'sknvgj6489TEmdjs' is not defined
```

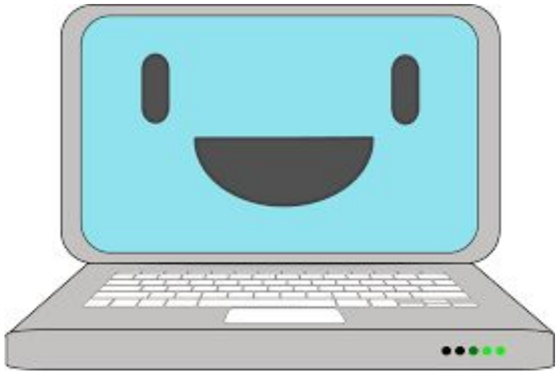
Mistakes are great!

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!



KeyError: 'Hairy Potter'

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

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

Write some code!!

This is the first bit of code we will do. What do you think it does?

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

Write some code!!

This is the first bit of code we will do. What do you think it does?

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

It prints the words “hello world” onto the screen!

Reusing variables

We can replace values in variables:

```
animal = "dog"
print("My favourite animal is a " + animal)
animal = "cat"
print("My favourite animal is a " + animal)
animal = animal + "dog"
print("My favourite animal is a " + animal)
```

What will this output?

Reusing variables

We can replace values in variables:

```python
animal = "dog"
print("My favourite animal is a " + animal)
animal = "cat"
print("My favourite animal is a " + animal)
animal = animal + "dog"
print("My favourite animal is a " + animal)
```

What will this output?

```
My favourite animal is a dog
My favourite animal is a cat
My favourite animal is a catdog
```

Asking a question!

It’s more fun when we get to interact with the computer!

Let’s get the computer to ask us a question!

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

This is what happens …

```
What is your name? Maddie
Hello Maddie
```

1. Computer prints ‘What is your name?’
2. Computer waits for you to type in your name
3. Computer prints ‘Hello Maddie’

Breaking it down

Big Tip : Put a space at the end of the question so it won't be squished together with your answer - it looks nicer!

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

SPACE 🙂

```
What is your name? Maddie
Hello Maddie
```

NO SPACE 🙁

```
What is your name?Maddie
Hello Maddie
```

Adding a comment!

Sometimes we want to write things in code that the computer doesn't look at! We use **comments** for that!

Use comments to write a note or explanation of our code

Comments make code easier for humans to understand

```
# This code was written by Sheree
```

We can make code into a comment if we don't want it to run (but don't want to delete it!)

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

Project time!

You now know all about printing, variables and input!

Let’s put what we learnt into our project

Try to do Lessons 1 & 2

Don't forget to copy your code when you move to a new Lesson!

The tutors will be around to help!

If Statements and Lists

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

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")
```

<u>What do you think happens?</u>

```
>>>
```

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")
```

<u>What do you think happens?</u>

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

Conditions

How about a different number???

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

<u>What do you think happens?</u>

```
>>>
```

Conditions

How about a different number???

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

What do you think happens?

```
>>>
```

Nothing!

If statements

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

What happens?

If statements

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

What happens?

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

If statements

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

What happens?

```
>>> GPN is awesom
```

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 :(
```

Elif statements

elif
Means we can give specific instructions for other words

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

What happens?

Elif statements

elif
Means we can give specific instructions for other words

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

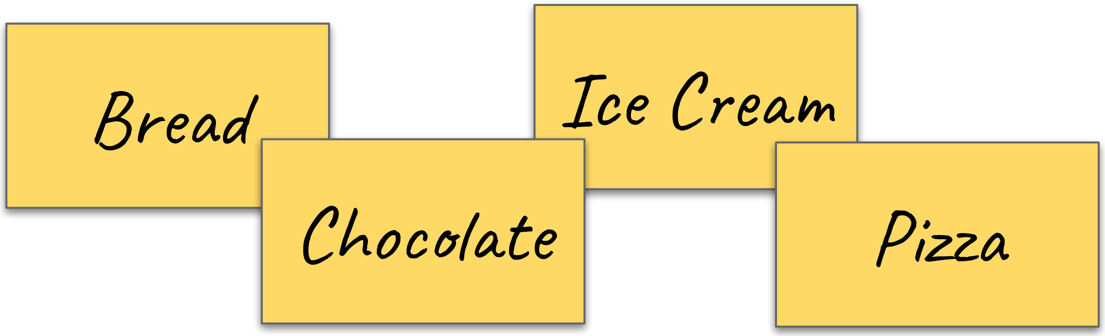
What happens?

```
>>> YUMM Chocolate!
```

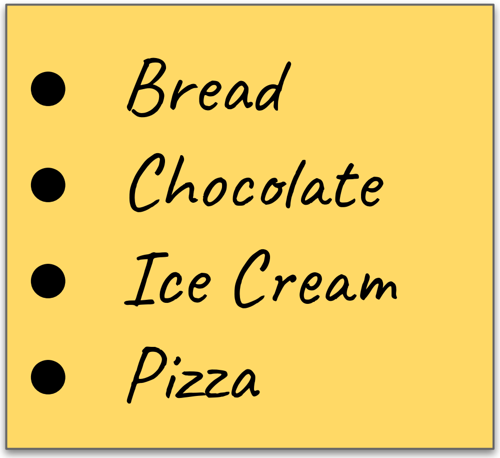
Lists

When we go shopping, we write down what we want to buy!

But we don't store it on lots of little pieces of paper!



We put it in one big shopping list!



Lists

It would be annoying to store it separately when we code too

```
>>> shopping_item1 = "Bread"
>>> shopping_item2 = "Chocolate"
>>> shopping_item3 = "Ice Cream"
>>> shopping_item4 = "Pizza"
```

So much repetition!

Instead we use a python list!

```
shopping_list = ["Bread", "Chocolate", "Ice Cream", "Pizza"]
```

List anatomy

Stored in the variable shopping_list

```
shopping_list = ["Bread", "Chocolate", "Ice Cream", "Pizza"]
```

List anatomy

Stored in the variable shopping_list

Made up of different items (these are strings)

```
shopping_list = ["Bread", "Chocolate", "Ice Cream", "Pizza"]
```

List anatomy

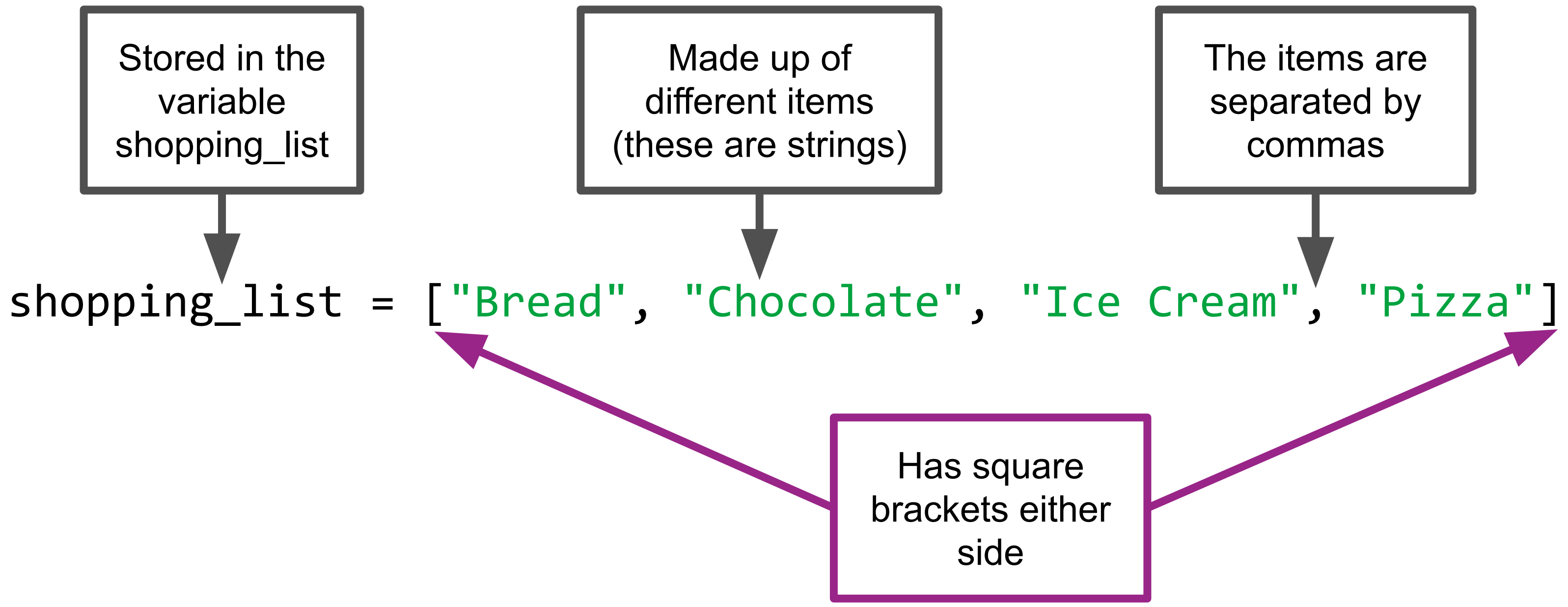
Stored in the variable shopping_list

Made up of different items (these are strings)

The items are separated by commas

```
shopping_list = ["Bread", "Chocolate", "Ice Cream", "Pizza"]
```

List anatomy


Stored in the variable shopping_list
Made up of different items (these are strings)
The items are separated by commas
Has square brackets either side

```
shopping_list = ["Bread", "Chocolate", "Ice Cream", "Pizza"]
```

Project Time!

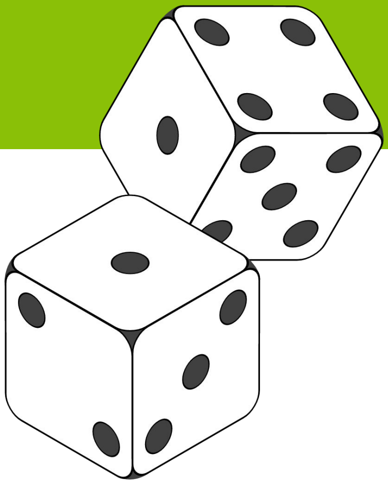
You now know all about `if` and **lists**!

See `if` you can do Lesson 3

The tutors will be around to help!

Random!

That's so random!



There's lots of things in life that are up to chance or random!

We want the computer to be random sometimes!

Python lets us **import** common bits of code people use! We're going to use the **random** module!

Using the random module

Let’s choose something randomly from a list!

This is like drawing something out of a hat in a raffle!

Try this!

1. **Import the random module!**

```
>>> import random
```

2. **Copy the shopping list into IDLE**

```
>>> shopping_list = ["eggs", "bread", "apples", "milk"]
```

3. **Choose randomly! Try it a few times!**

```
>>> random.choice(shopping_list)
```

Using the random module

You can also assign your random choice to a variable

```
>>> import random
>>> shopping_list = ["eggs", "bread", "apples", "milk"]
>>> random_food = random.choice(shopping_list)
>>> print(random_food)
```

Project Time!

Raaaaaaaaaandom! Can you handle that?

Let’s try use it in our project!

Try to do Lesson 4

The tutors will be around to

For Loops

For Loops

For loops allow you to do something a certain number of times.

We use them when we know exactly how many times we want to do something!

For Loops

```
number = 10
for i in range(number):
    #Do something
```

For Loops

```
number = 10
for i in range(number):
    #Do something
```

The for word tells python we want to use a loop

For Loops

This i is a temporary variable which will count how many times we have looped.

```
number = 10
for i in range(number):
    #Do something
```

The for word tells python we want to use a loop

For Loops

This `i` is a temporary variable which will count how many times we have looped.

```
number = 10
for i in range(number):
    #Do something
```

The for word tells python we want to use a loop

This part says we want to loop `number` amount of times (in this case, 10)

For Loops

This `i` is a temporary variable which will count how many times we have looped.

```
number = 10
for i in range(number):
    #Do something
```

The for word tells python we want to use a loop

The code indented in the loop is what will happen every time.

This part says we want to loop `number` amount of times (in this case, 10)

Looping how many times?

We can loop through a list:

```
friends = 4
for i in range(friends):
    print(“Hello friend!”)
```

What’s going to happen?

Looping how many times?

We can loop through a list:

```
friends = 4
for i in range(friends):
    print(“Hello friend!”)
```

What’s going to happen?

```
>>> Hello friend!
>>> Hello friend!
>>> Hello friend!
>>> Hello friend!
```

Looping how many times?

We can loop through a list:

```
friends = 4
for i in range(friends):
    print(“Hello friend!”)
```

What’s going to happen?

```
>>> Hello friend!
>>> Hello friend!
>>> Hello friend!
>>> Hello friend!
```

We do what’s in the for loop as many times as what is in the “range”

Project Time!

Now you know how to use a for loop!

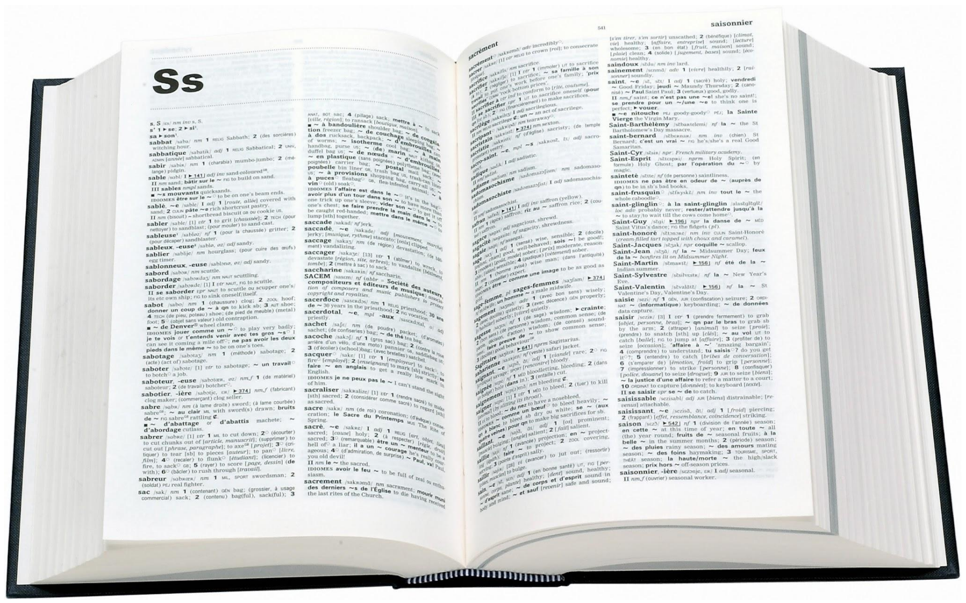
Try to do Lesson 5

...if you are up `for` it!

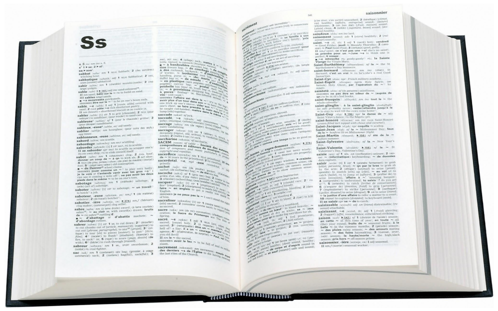
The tutors will be around to help!

Dictionaries

Dictionaries!



Dictionaries!

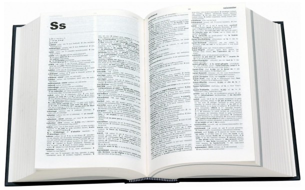


You know dictionaries!

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 <u>key → value</u> pattern!

Dictionaries anatomy!

This is a python dictionary!

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

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

Dictionaries anatomy!

This is a python dictionary!

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

Stored in the variable phone_book

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

Dictionaries anatomy!

This is a python dictionary!

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

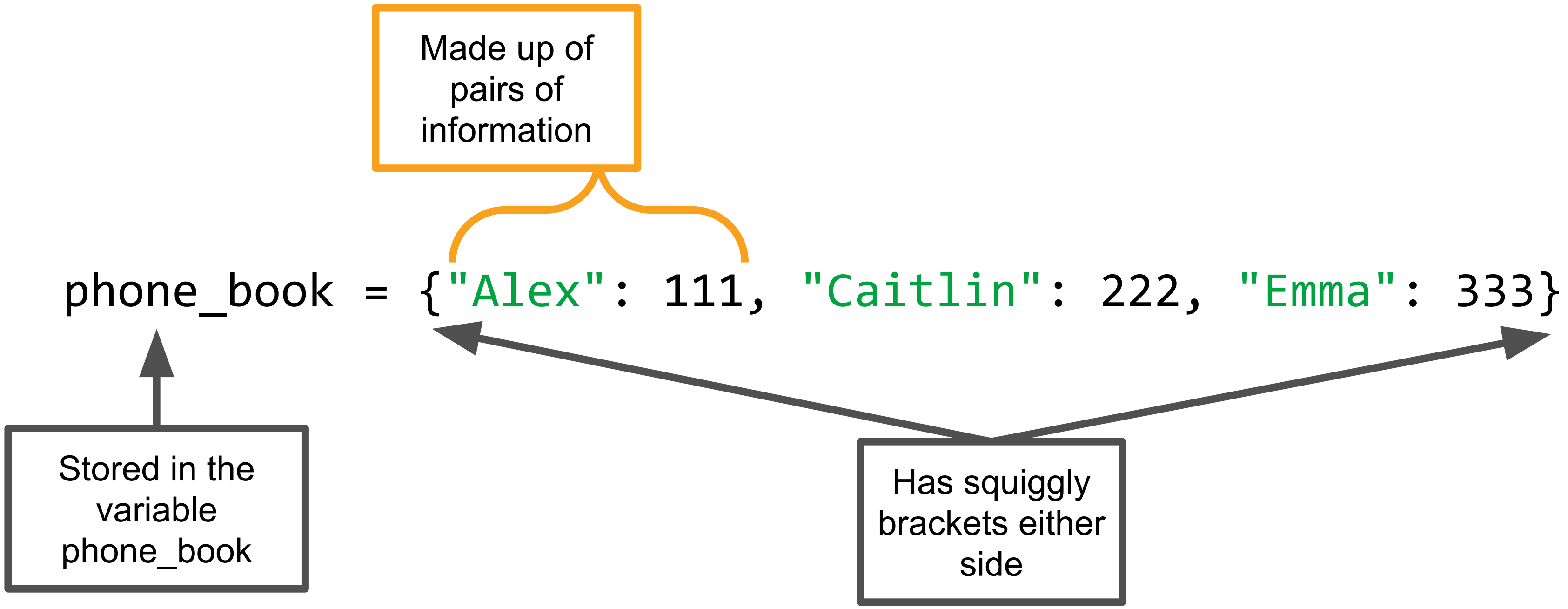
Stored in the variable phone_book

Has squiggly brackets either side

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

Dictionaries anatomy!

This is a python dictionary!

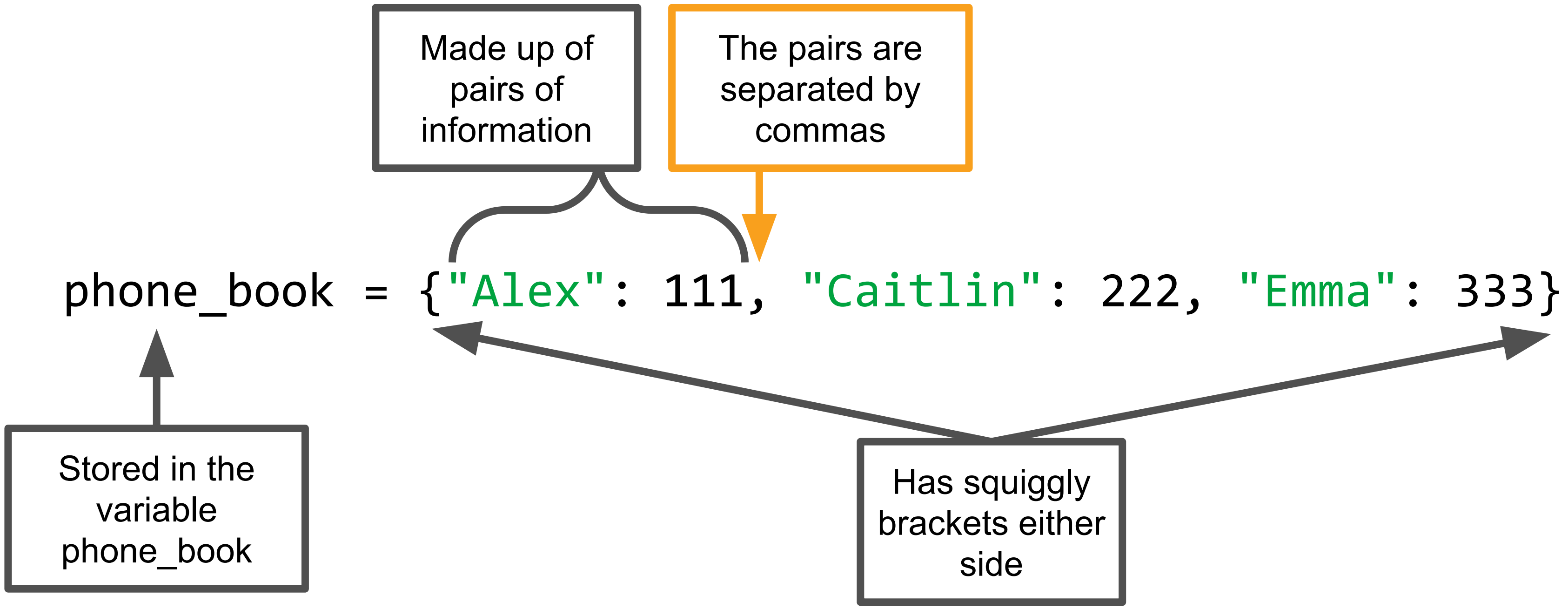

Made up of pairs of information
Stored in the variable phone_book
Has squiggly brackets either side

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

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

Dictionaries anatomy!

This is a python dictionary!

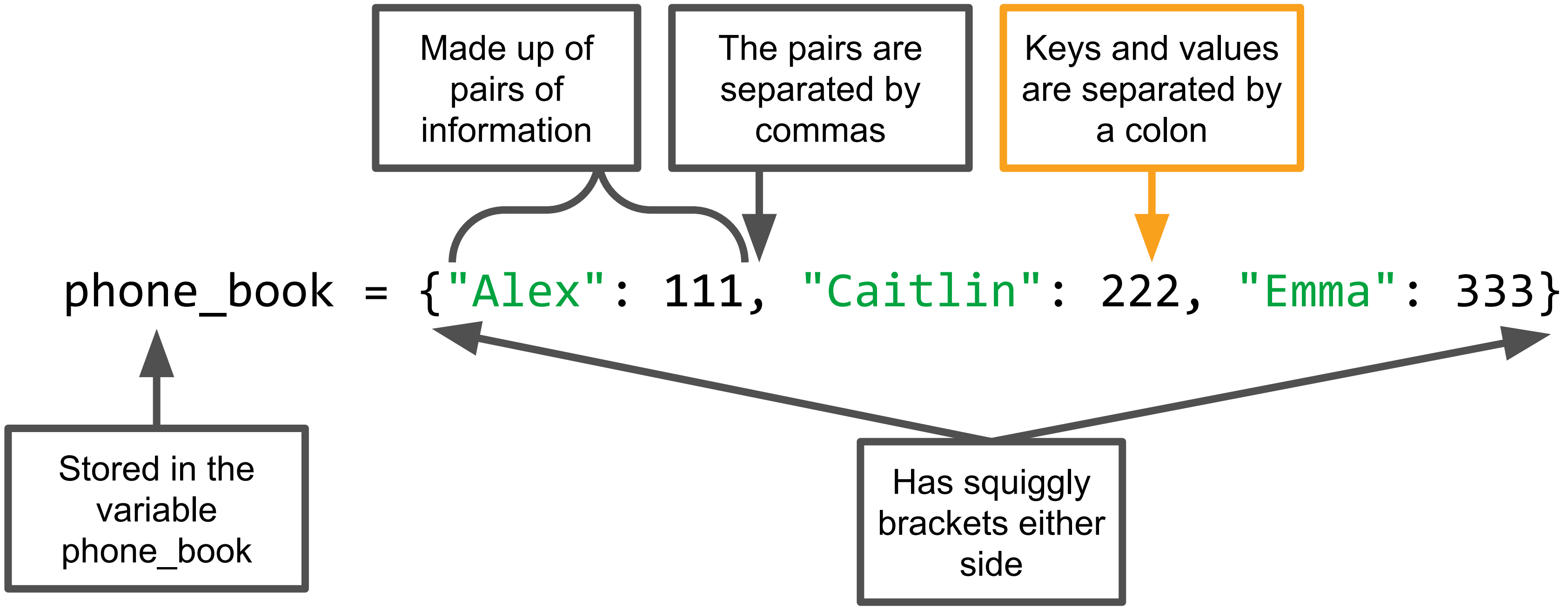

Made up of pairs of information
The pairs are separated by commas
Stored in the variable phone_book
Has squiggly brackets either side

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

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

Dictionaries anatomy!

This is a python dictionary!



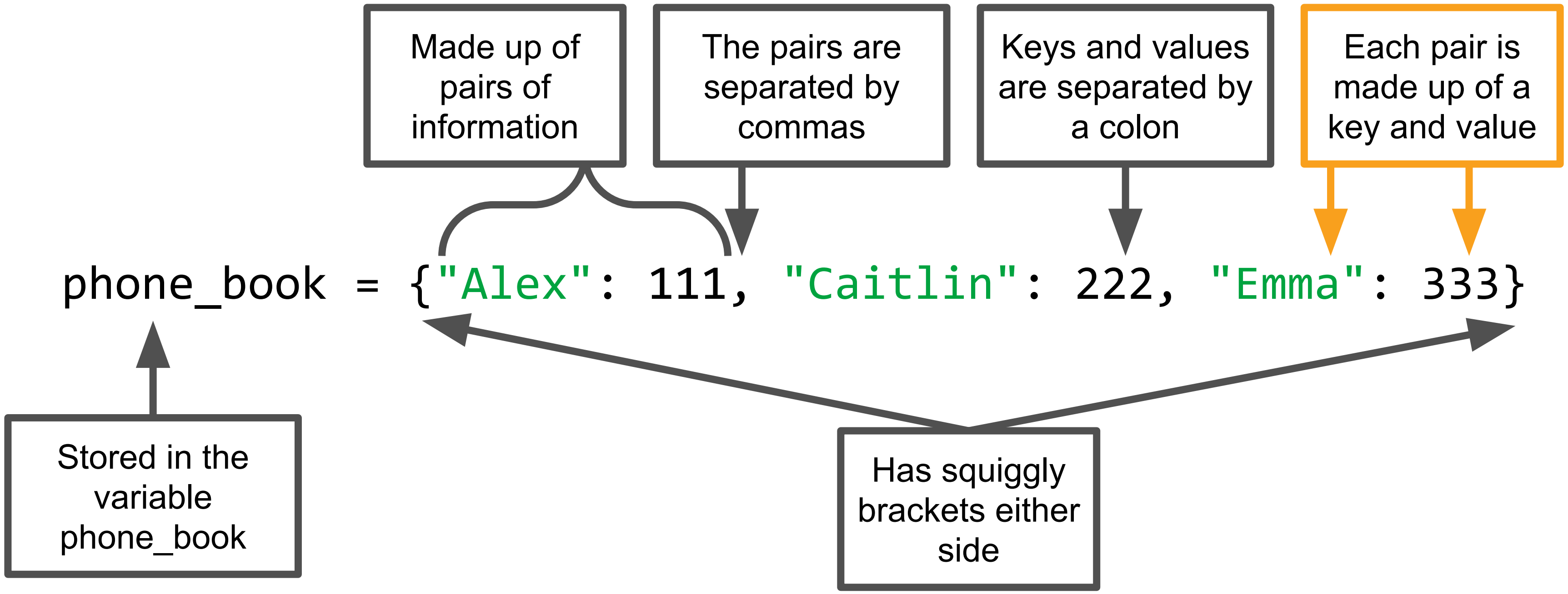
Made up of pairs of information
The pairs are separated by commas
Keys and values are separated by a colon
Stored in the variable phone_book
Has squiggly brackets either side

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

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

Dictionaries anatomy!

This is a python dictionary!



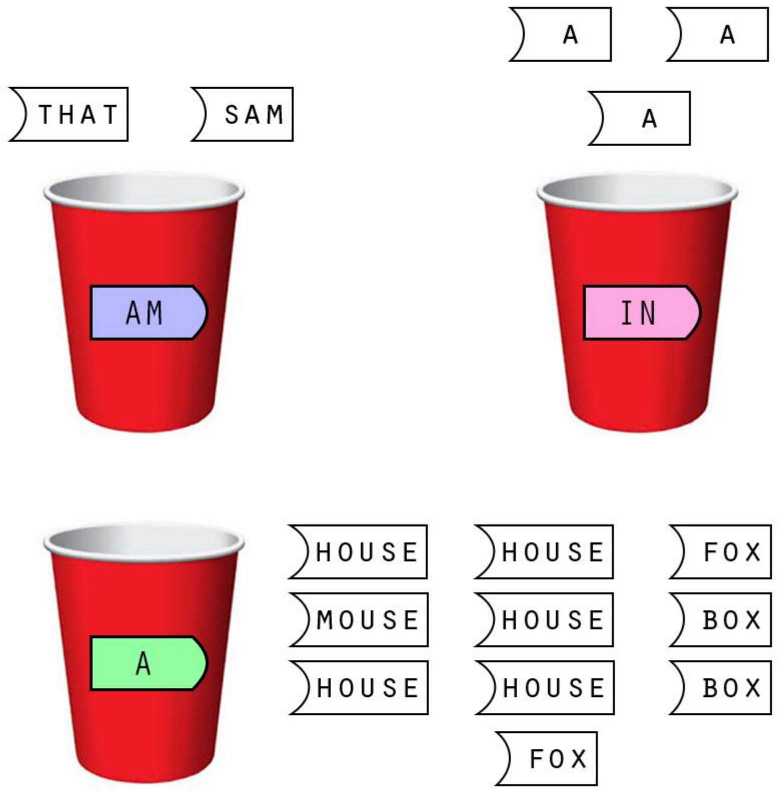
Made up of pairs of information
The pairs are separated by commas
Keys and values are separated by a colon
Each pair is made up of a key and value
Stored in the variable phone_book
Has squiggly brackets either side

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

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

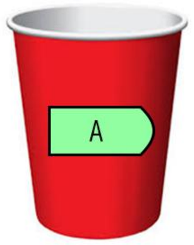
Cups!!

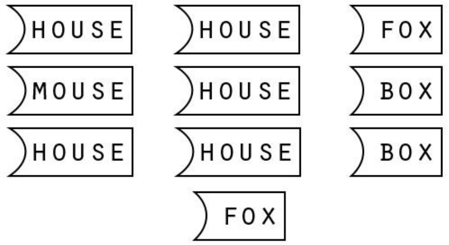
Remember the cups activity from the start of the day?



A Single Cup!

The word "A" **can be followed by** **Any of these words**



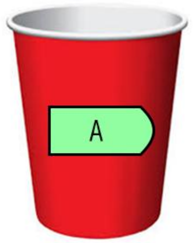


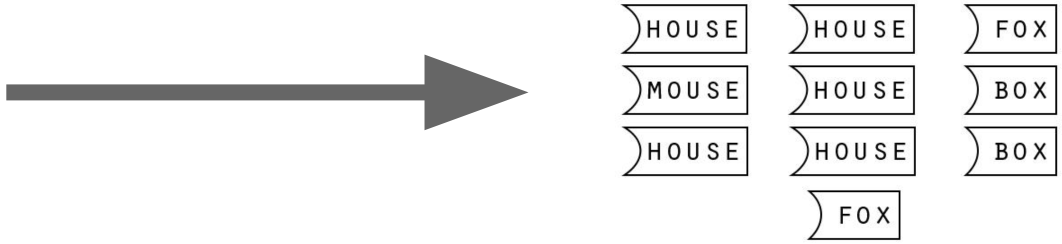
Key

Value

A Single Cup!

The word "A" **can be followed by** **Any of these words**



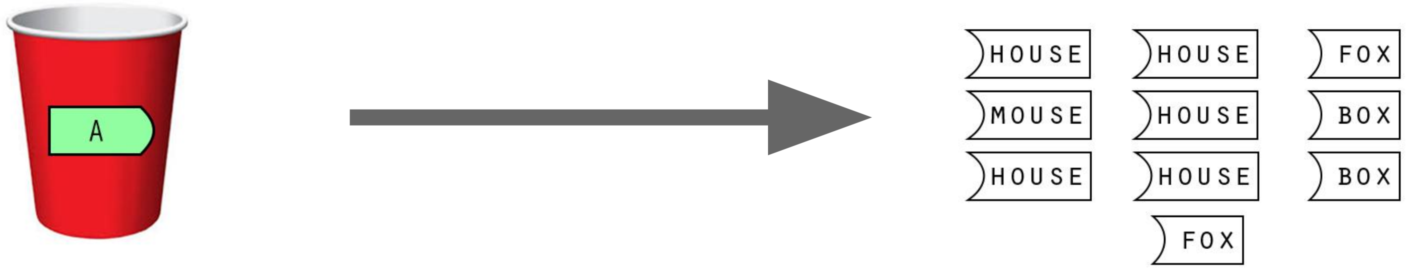


We can store the slips of paper as a python list!

```
['house', 'mouse', 'house',
'mouse', 'box', 'fox', 'box',
'fox', 'house', 'mouse']
```

A Single Cup!

The word "A" can be followed by **Any of these words**



We want to look up the word "a" and get back the list!

```
{'a'  :  ['house', 'mouse', 'house',
          'mouse', 'box', 'fox', 'box',
          'fox', 'house', 'mouse']}
```

A Single Cup!

So we get a Dictionary with a List value!

```
{'a' :    ['house', 'mouse', 'house',
           'mouse', 'box', 'fox', 'box',}
           'fox', 'house', 'mouse']
```

Key

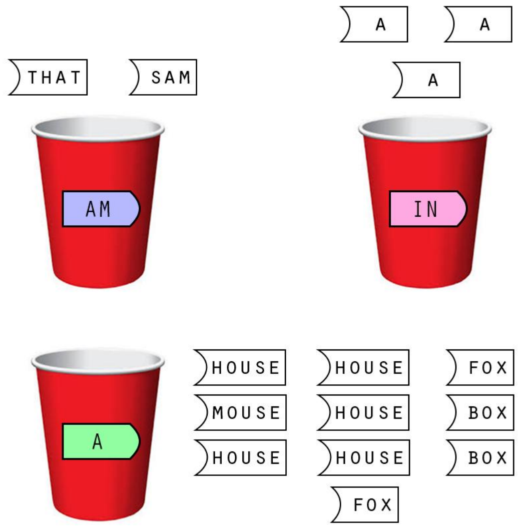
Value

If you look up “A” you get back a list of all the words that can follow “a”

Cups → Dictionary with lists!

Here's what it looks like for a few more cups!

```
cups = {'am': ['Sam', 'That'],
        'In': ['a', 'a', 'a'],
        'a' : ['house', 'mouse',
               'house', 'mouse',
               'box', 'fox', 'box',
               'fox', 'house',
               'Mouse']
       ....}
```

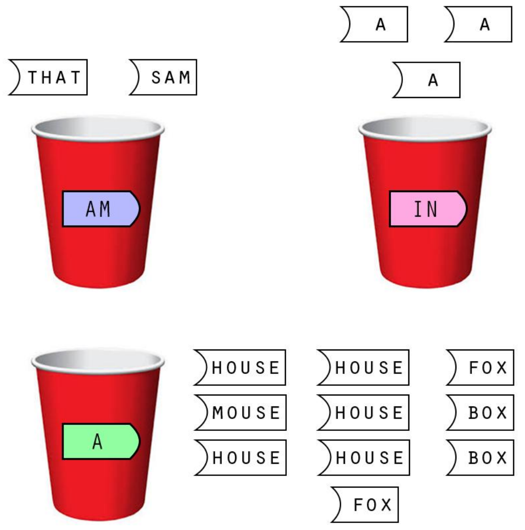


You can get the whole cup dictionary from today's website!

Is it there?

We can check if something is a key in a dictionary like this:

```
cups = {'am': ['Sam', 'That'],
        'In': ['a', 'a', 'a'],
        'a' : ['house', 'mouse',
               'house', 'mouse',
               'box', 'fox', 'box',
               'fox', 'house',
               'Mouse']
       ....}
```



```
if current_word in cups:
```

Project time!

You now know all about lists and dictionaries!

Let's put what we learnt into our project

Try to do Lessons 6 & 7

The tutors will be around to help!

More Dictionaries and Lists!

Getting words from sample text

In order to be able to read in lots of text we need to be able to turn sentences into a list of words.

We can do this by using .split() on our text!

Using split

```
text = “a really cool sentence”
words = text.split()
print(words)
```

What do you think words will be?

Using split

```
text = “a really cool sentence”
words = text.split()
print(words)
```

What do you think words will be?

```
[“a”, “really”, “cool”, “sentence”]
```

More things you can do with lists!

There's lots of cool things we can do with lists! Like:

Getting the length of a list

```
words = [“a”, “really”, “cool”, “sentence”]
print(len(words))
```

Adding new items to a list

```
words.append(“yay”)
print(words)
```

More things you can do with lists!

There's lots of cool things we can do with lists! Like:

Getting the length of a list

```
words = [“a”, “really”, “cool”, “sentence”]
print(len(words))
 4
```

Adding new items to a list

```
words.append(“yay”)
print(words)
```

More things you can do with lists!

There's lots of cool things we can do with lists! Like:

Getting the length of a list

```
words = [“a”, “really”, “cool”, “sentence”]
print(len(words))
```

```
 4
```

Adding new items to a list

```
words.append(“yay”)
print(words)
```

```
 [“a”, “really”, “cool”, “sentence”, “yay”]
```

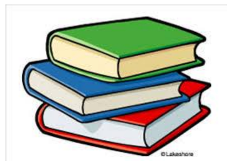
Accessing Lists!

This favourites `list` holds four strings in order:

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

We can count out the items using index numbers!

0 **1** **2** **3**



Remember: Indices start from zero!

Accessing Lists

We access the items in a `list` with an index such as `[0]`:

```
>>> faves[0]
'books'
```

What code do you need to access the second item in the list?

Accessing Lists

We access the items in a `list` with an index such as `[0]`:

```
>>> faves[0]
'books'
```

What code do you need to access the second item in the list?

```
>>> faves[1]
'butterfly'
```

0 **[1]** 2 3

Going Negative

Negative indices count backwards from the end of the `list`:

```
>>> faves[-1]
'skateboard'
```

What would `faves[-2]` return?

Going Negative

Negative indices count backwards from the end of the `list`:

```
>>> faves[-1]
'skateboard'
```

What would `faves[-2]` return?

```
>>> faves[-2]
'chocolate'
```

-4

-3

[-2]

-1

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 our dictionaries!

We've seen how to use dictionaries - but how do we update existing ones? Let's have a look at a phone book example!

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

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

```
>>> phone_book["Rowena"] = 444
>>> phone_book
{ "Alex": 123, "Caitlin": 222, "Emma": 333,
"Rowena": 444 }
```

Lists in dictionaries!

We've been using lists as the values of our dictionary like this:

- Let's make some sports teams:

```
>>> team_members = {
        "Sydney": ["Pauline", "Srishti", "Amara"],
        "Perth": ["Crischell", "Ash", "Taylah"]
    }
```

- What happens if you do:

```
>>> team_members["Sydney"]
```

- What if we did this?

```
>>> team_members["Perth"].append("Priya")
```

Lists in dictionaries!

We've been using lists as the values of our dictionary like this:

- Let's make some sports teams:

```
>>> team_members = {
        "Sydney": ["Pauline", "Srishti", "Amara"],
        "Perth": ["Crischell", "Ash", "Taylah"]
    }
```

- What happens if you do:

```
>>> team_members["Sydney"]
 ["Pauline", "Srishti", "Amara"]
```

- What if we did this?

```
>>> team_members["Perth"].append("Priya")
```

Lists in dictionaries!

We've been using lists as the values of our dictionary like this:

- Let's make some sports teams:

```
>>> team_members = {
        "Sydney": ["Pauline", "Srishti", "Amara"],
        "Perth": ["Crischell", "Ash", "Taylah"]
    }
```

- What happens if you do:

```
>>> team_members["Sydney"]
 ["Pauline", "Srishti", "Amara"]
```

- What if we did this?

```
>>> team_members["Perth"].append("Priya")
 ["Pauline", "Srishti", "Amara", "Priya"]
```

Project Time!

Now you know even more about Lists and Dictionaries!

You can now try the second section of your workbook!

The tutors will be around to help!

Files

Opening files!

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!

```
f = open("test.txt")
```

You'll now be able to read the things in `f`

If your file is in the same location as your code you can just use the name!

A missing file causes an error

Here we try to open a file that doesn't exist:

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

You can read a whole file into a string

```
>>> f = open('haiku.txt')
>>> my_string = f.read()



>>> print(my_stirng)
Wanna go outside.
Oh NO! Help! I got outside!
Let me back inside!
```

```
haiku.txt

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

You can also read in one line at a time

You can use a for loop to only get 1 line at a time!

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

```
Wanna go outside.

Oh NO! Help! I got outside!

Let me back inside!
```

Why is there an extra blank line each time?

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'
```

x.strip() is safe as lines without newlines will be unaffected

Reading and stripping!

```python
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!

Using **with**!

This is a special trick for opening files!

```
with open("words.txt") as f:
  for line in f:
    print(line.strip())
```

It automatically closes your file for you!

It's good when you are writing files in python!

Project Time!

Now you can read some stuff

You can now try the third section of your workbook!

The tutors will be around to help!