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:

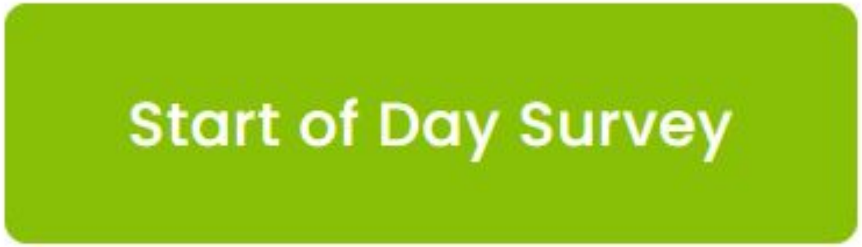
- These **slides** (to take a look back or go on ahead).
- A link to your **workbook** in EdStem
- Other helpful bits to use through the day!

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!!

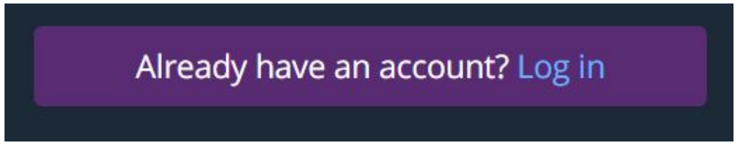
Introduction to Edstem

Log on

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

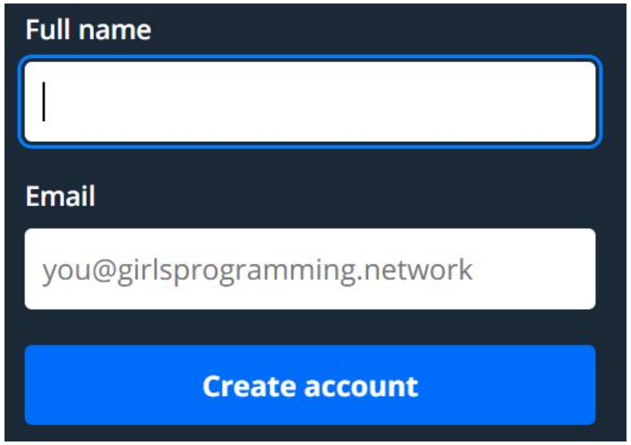
Signing up to Edstem

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

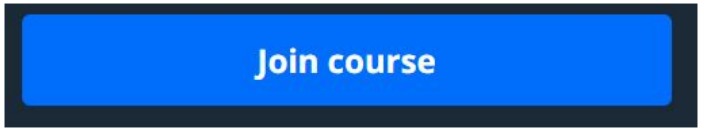


If you haven’t got an account, let’s make one:

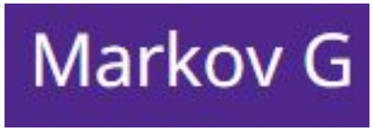
1. Type in your Full Name
2. Type in your personal email
3. Click Create Account
4. Go to your email and verify your new account
5. Create a password



Click Join Course



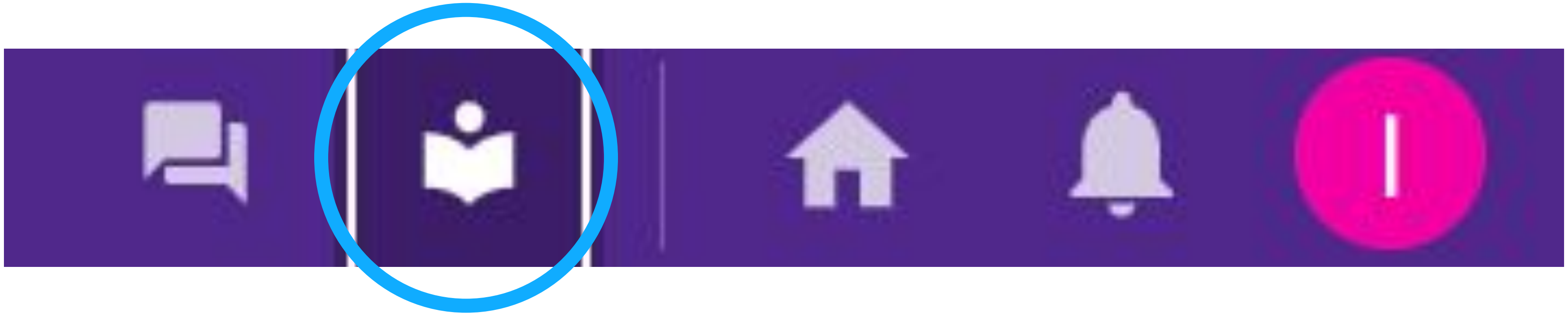
The name of your course will be at the top :



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)



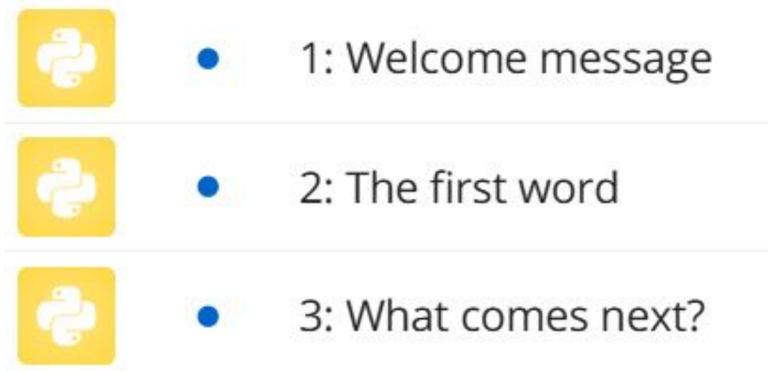
The set up of the workbook

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

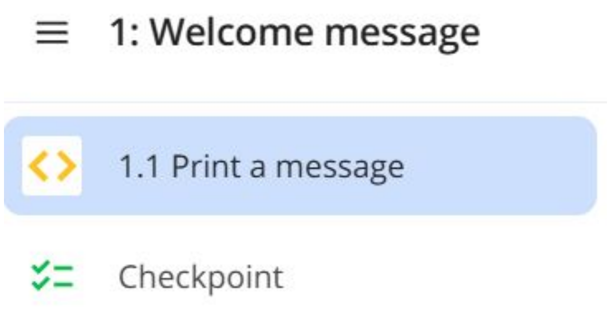
Inside a Chapter there are two main types of pages:

- **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



1: Welcome message

1.1 Print a message

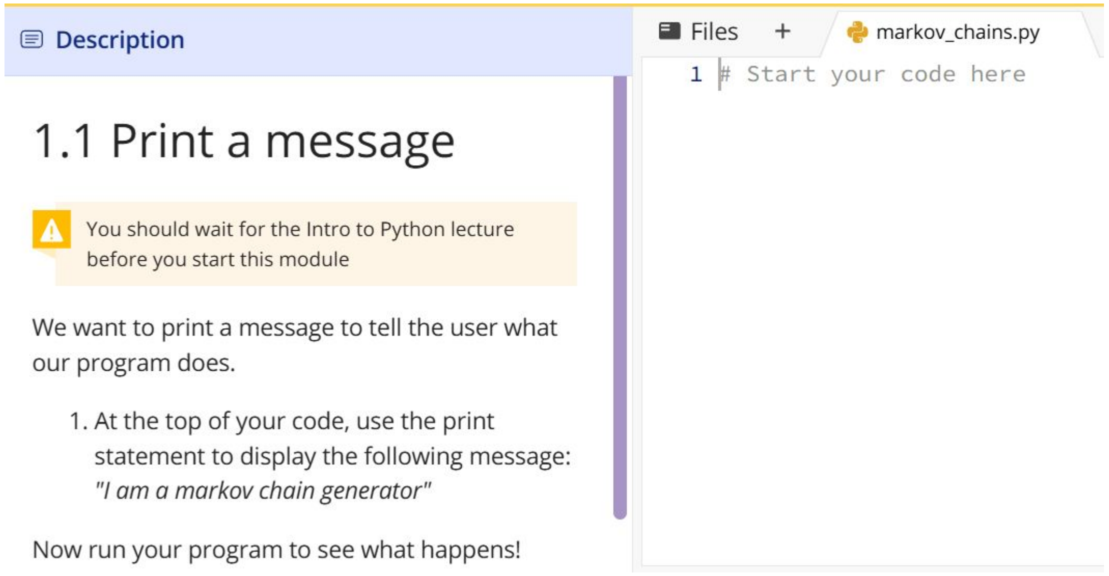
Checkpoint

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



```
# 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

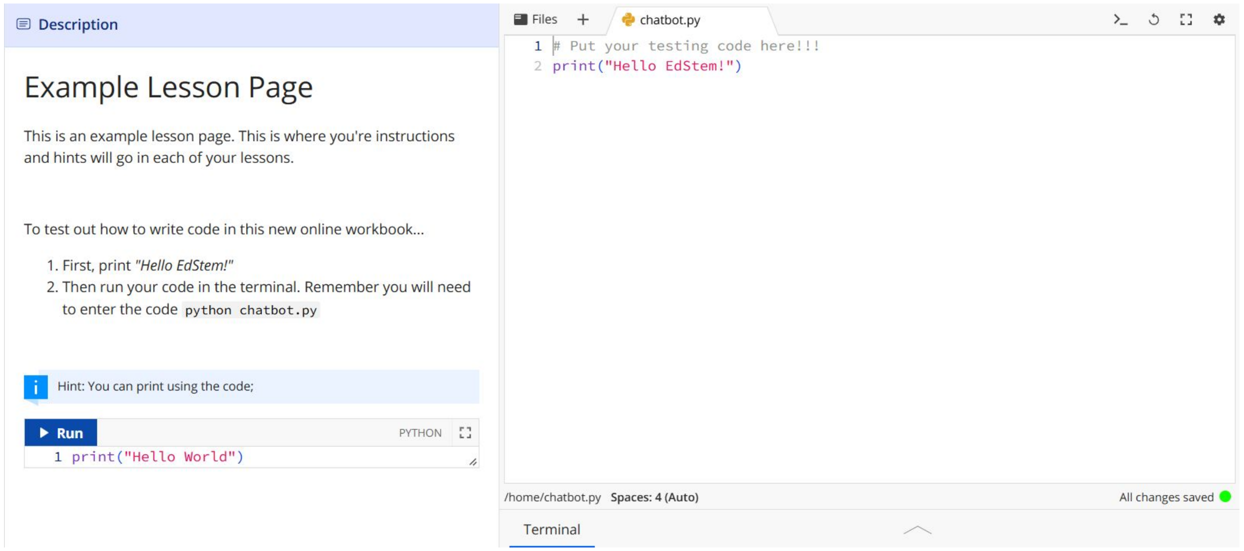
How to do the work

In each lesson there is:

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

You will need to **copy your code from the last lesson**, then follow the instructions to change your code so that you can work towards finishing the project.

There are also Hints and Code Blocks to help you



Example Lesson Page

This is an example lesson page. This is where you're instructions and hints will go in each of your lessons.

To test out how to write code in this new online workbook...

1. First, print *"Hello EdStem!"*
2. Then run your code in the terminal. Remember you will need to enter the code `python chatbot.py`

Hint: You can print using the code;

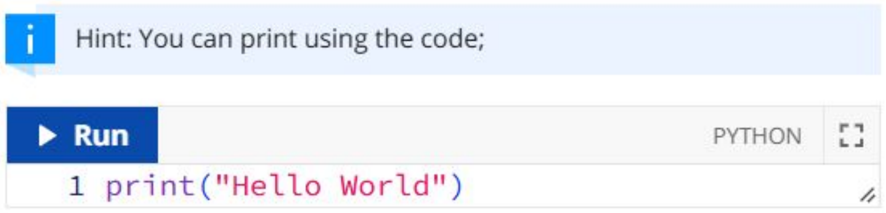
```
print("Hello World")
```

```
# Put your testing code here!!!
print("Hello EdStem!")
```

/home/chatbot.py Spaces: 4 (Auto) All changes saved

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")
```

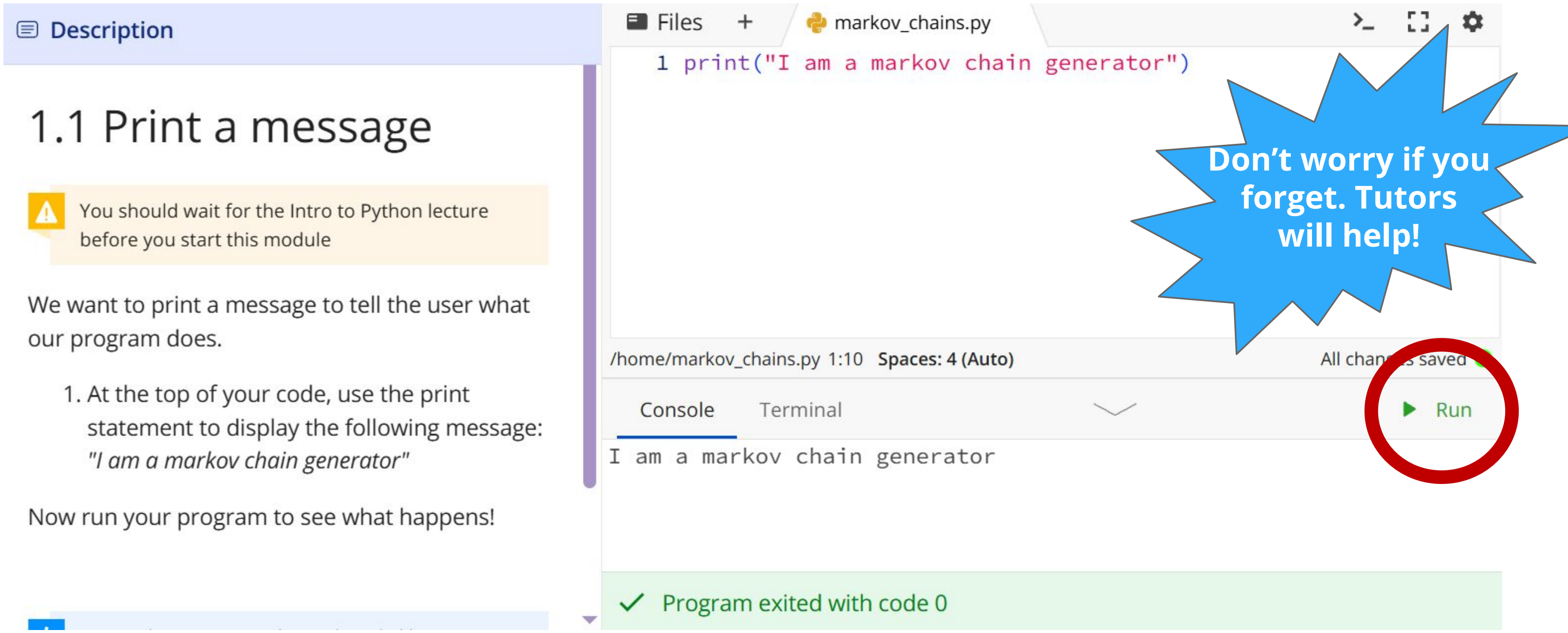
If you press the blue run button it will show you what that code does, you can even change the code to see if/how it changes.

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 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!

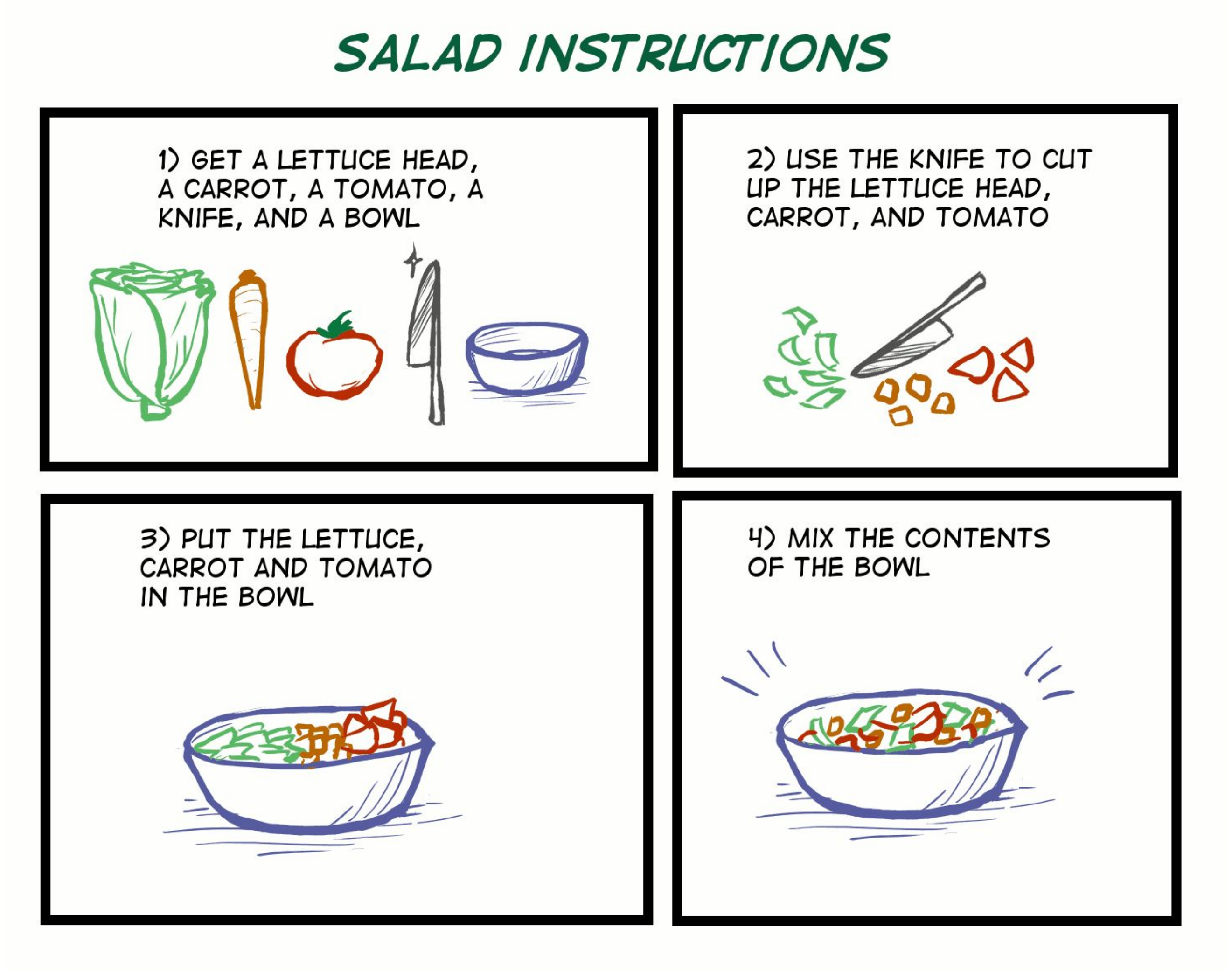
Programming is a language to talk to computers

People are smart! Computers are dumb!

Programming is like a recipe!

Computers do EXACTLY what you say, every time.

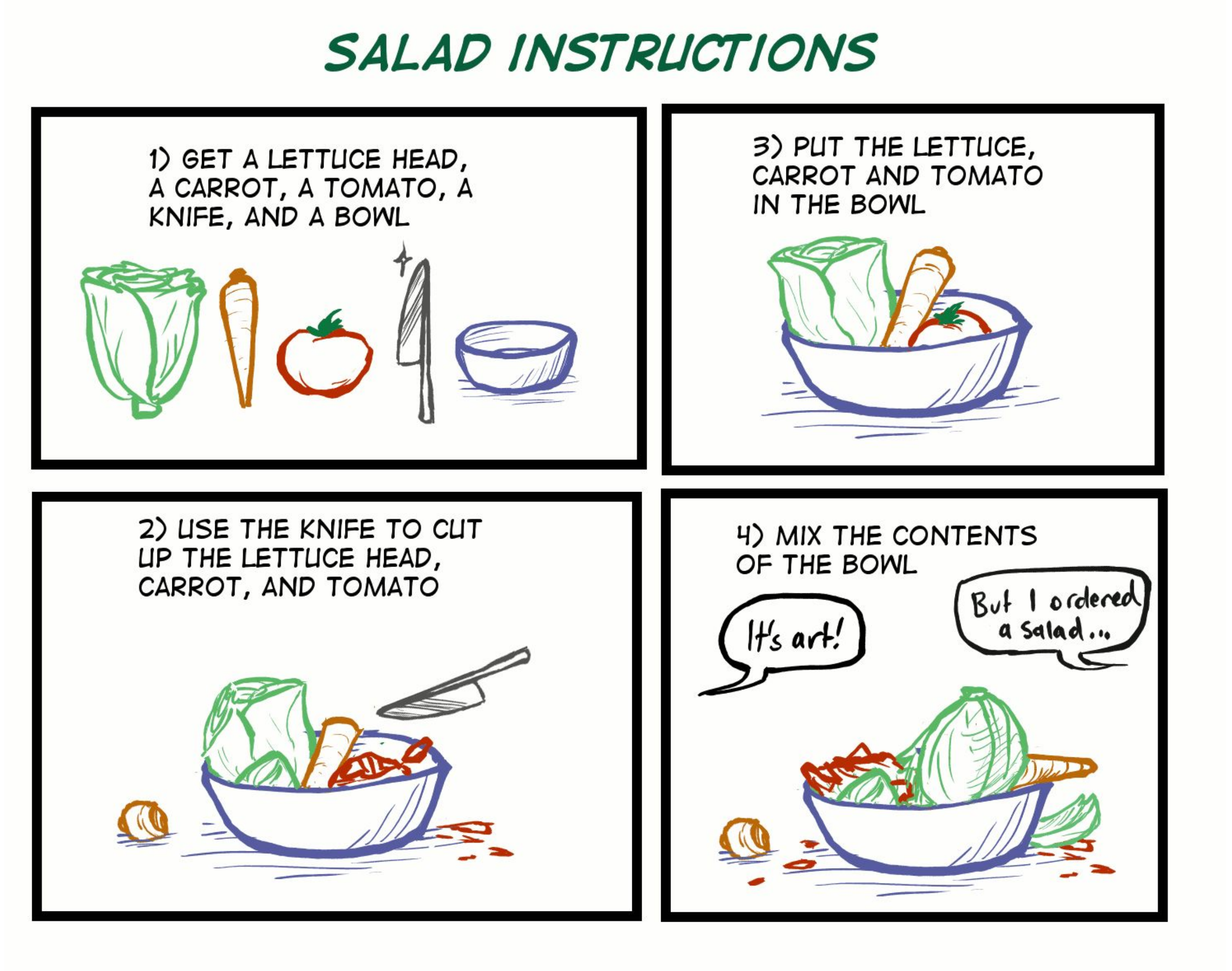
Which is great if you give them a good recipe!


SALAD INSTRUCTIONS
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….

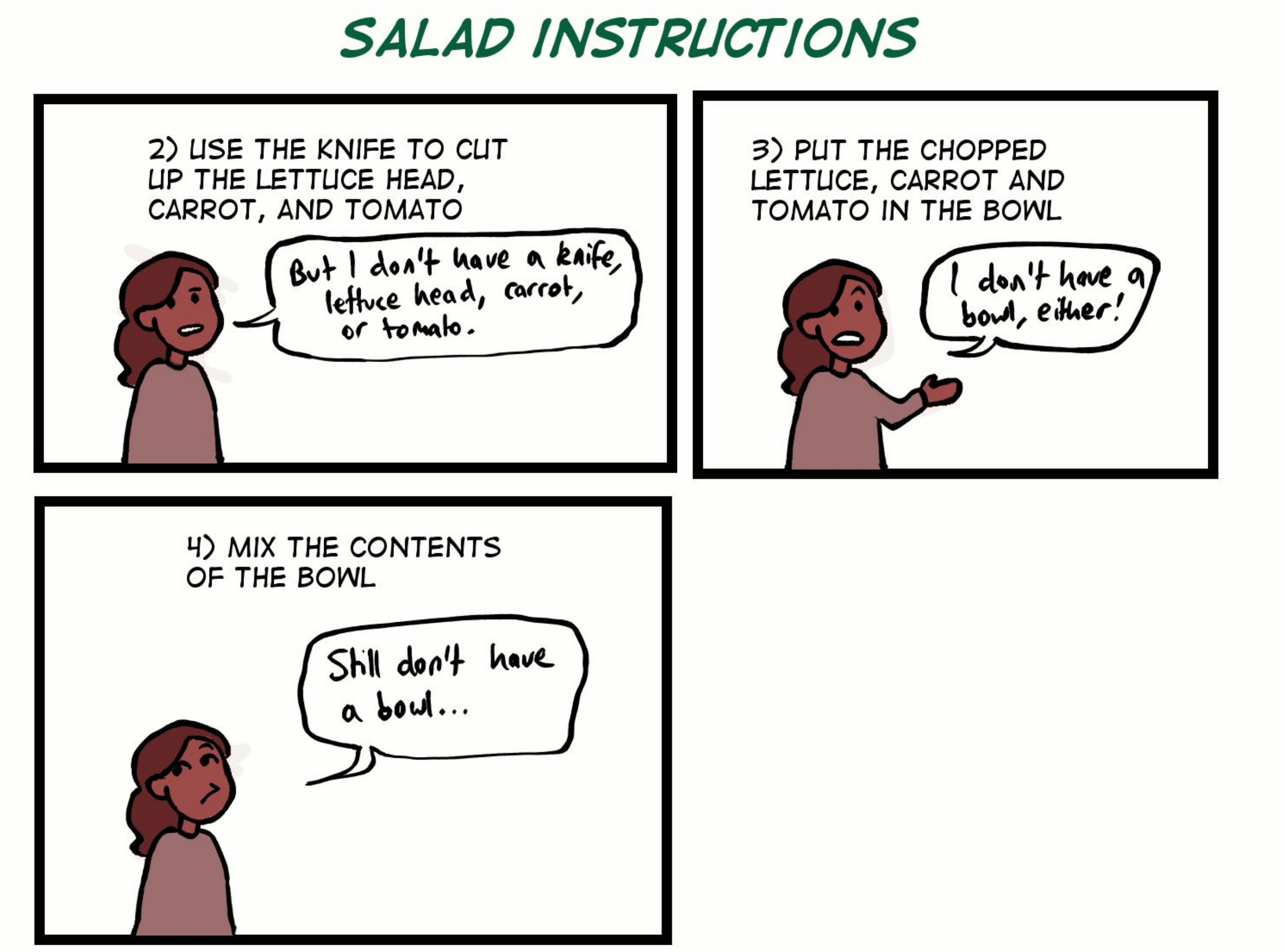
A computer wouldn't know this recipe was wrong!


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
It's art!
But I ordered a salad...

People are smart! Computers are dumb!

Computers are bad at filling in the gaps!

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


SALAD INSTRUCTIONS
2) USE THE KNIFE TO CUT UP THE LETTUCE HEAD, CARROT, AND TOMATO
But I don't have a knife, lettuce head, carrot, or tomato.
3) PUT THE CHOPPED LETTUCE, CARROT AND TOMATO IN THE BOWL
I don't have a bowl, either!
4) MIX THE CONTENTS OF THE BOWL
Still don't have a bowl...

Everyone/thing has strengths!

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

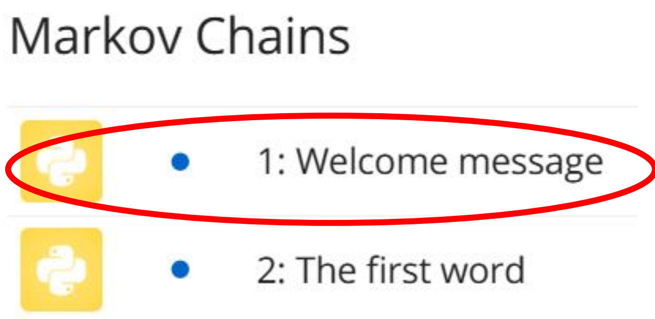
- Does exactly what you tell it
- Does it the same every time
- Doesn’t need to sleep!
- Will work for hours on end!
- Get smarter when you tell them how

Intro to Python

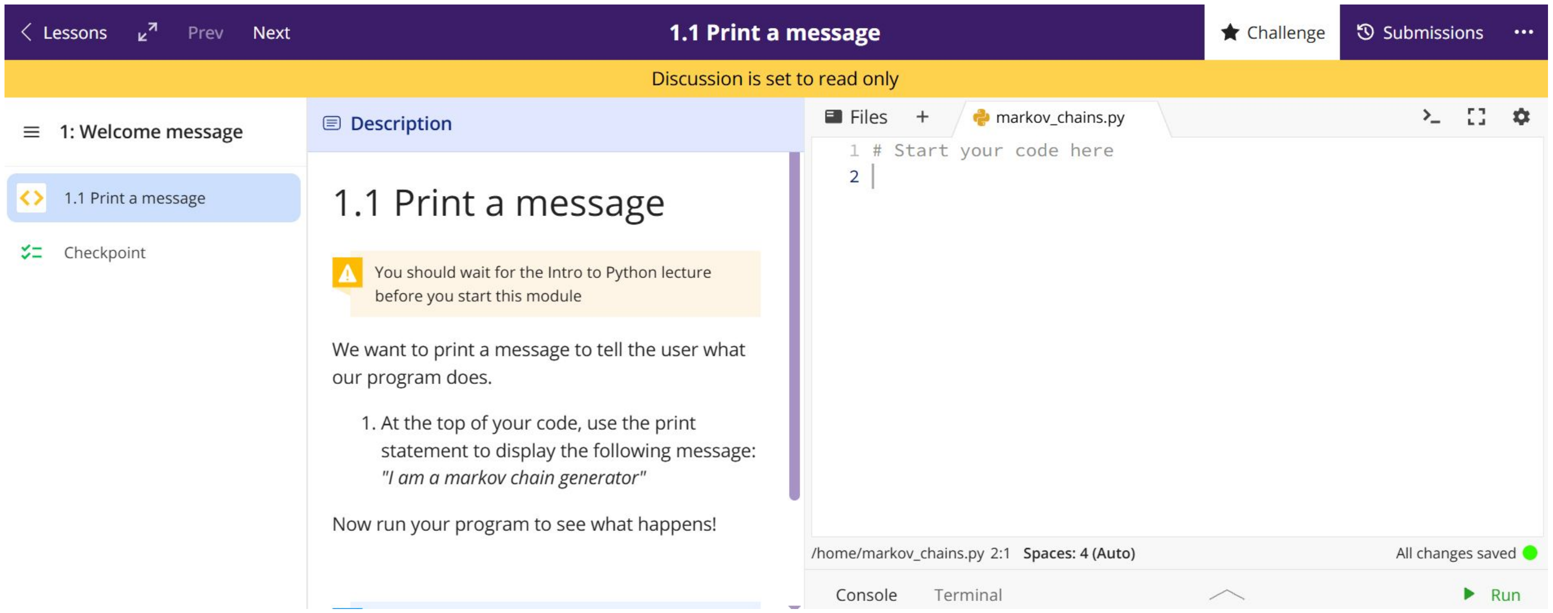
Let's get coding!

Let's make a mistake!

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


Markov Chains

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"*

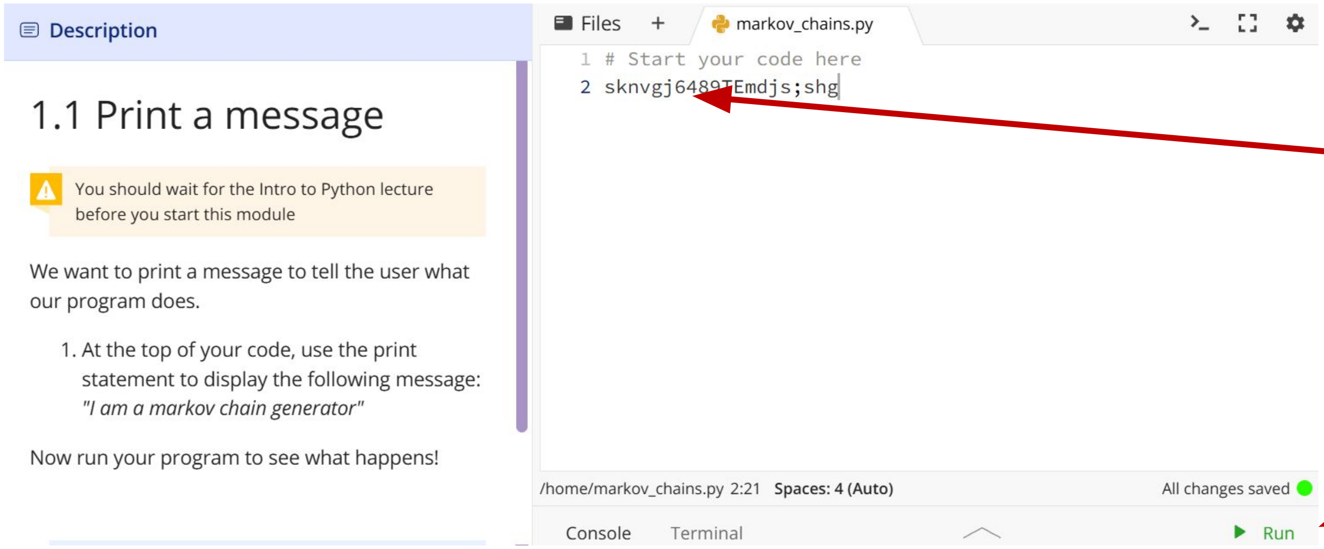
Now run your program to see what happens!

```
# Start your code here

```

/home/markov_chains.py 2:1 Spaces: 4 (Auto) All changes saved

Let's make a mistake!



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!

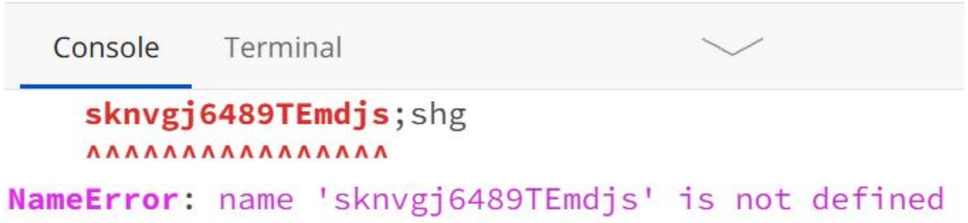
```
# Start your code here
sknvgj6489TEmdjs;shg
```

/home/markov_chains.py 2:21 **Spaces: 4 (Auto)** All changes saved

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
```

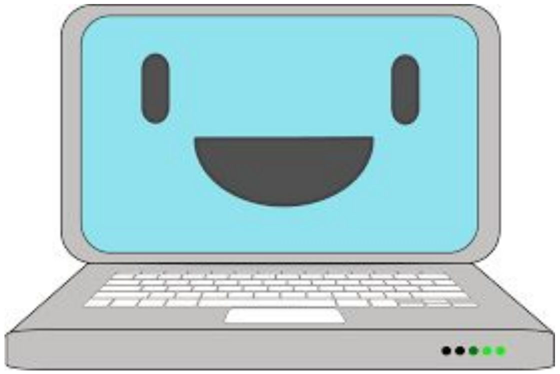
Mistakes are great!

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
```

We can learn from our mistakes!

Error messages help us fix our mistakes!

We read error messages from bottom to top

```
Traceback (most recent call last):
  File "C:/Users/Madeleine/Desktop/tmp.py", line 9, in <module>
    print("I have " + 5 + " apples")
TypeError: can only concatenate str (not "int") to str
```

We can learn from our mistakes!

Error messages help us fix our mistakes!

We read error messages from bottom to top

```
Traceback (most recent call last):
  File "C:/Users/Madeleine/Desktop/tmp.py", line 9, in <module>
    print("I have " + 5 + " apples")
TypeError: can only concatenate str (not "int") to str
```

1. What went wrong

We can learn from our mistakes!

Error messages help us fix our mistakes!

We read error messages from bottom to top

```
Traceback (most recent call last):
  File "C:/Users/Madeleine/Desktop/tmp.py", line 9, in <module>
    print("I have " + 5 + " apples")
TypeError: can only concatenate str (not "int") to str
```

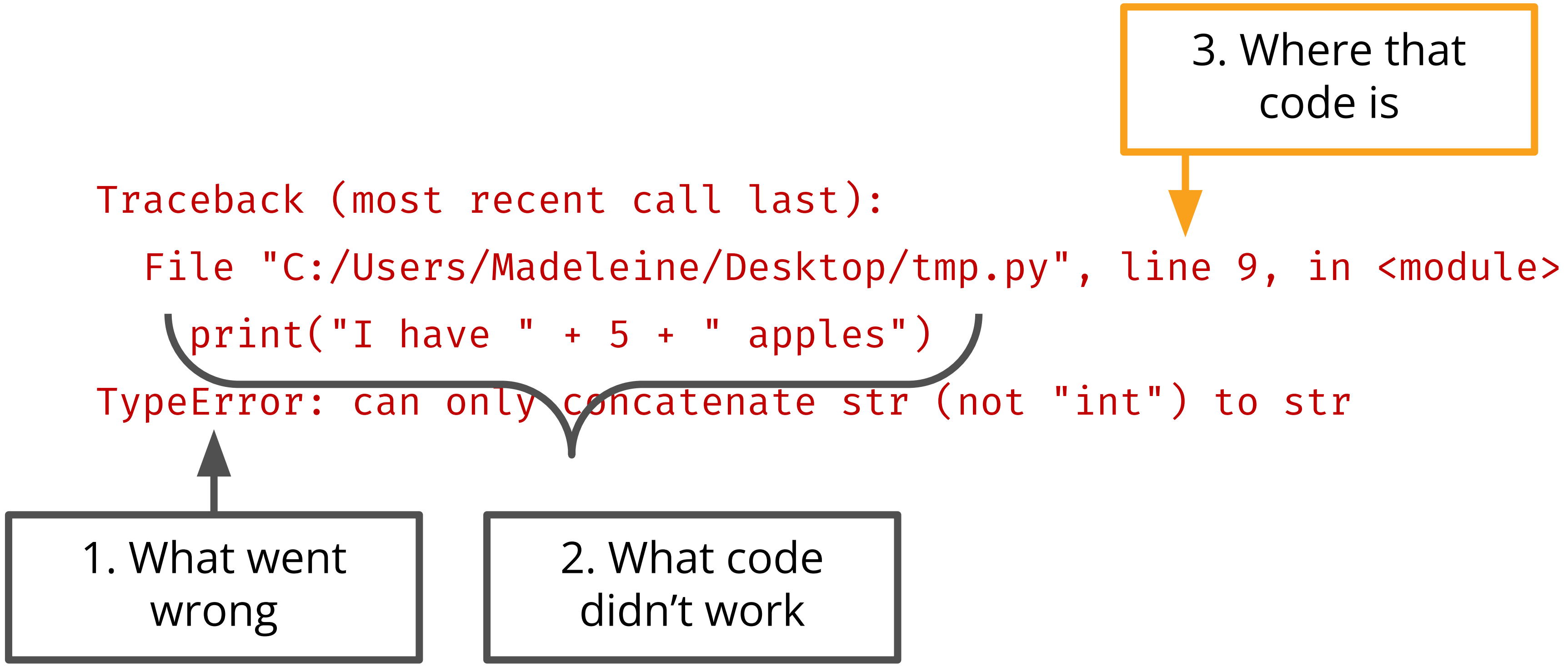
1. What went wrong

2. What code didn't work

We can learn from our mistakes!

Error messages help us fix our mistakes!

We read error messages from bottom to top



```
Traceback (most recent call last):
  File "C:/Users/Madeleine/Desktop/tmp.py", line 9, in <module>
    print("I have " + 5 + " apples")
TypeError: can only concatenate str (not "int") to str
```

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!

Write some code!!

1. Type the following into the code window (make sure you include the quotes (“) at the start and end)
2. Run the code by clicking Run

```
print(“hello world”)
```

Did it print the text:

```
hello world
```

???

Tell me more!

We can `print` things in lots of different ways in python!

```
>>> print("Hello world!")


>>> print("Hello", "world!")


>>> print("Hello", "world", end="!")
```

Tell me more!

We can `print` things in lots of different ways in python!

```
>>> print("Hello world!")
Hello world!
>>> print("Hello", "world!")


>>> print("Hello", "world", end="!")
```

Tell me more!

We can `print` things in lots of different ways in python!

```
>>> print("Hello world!")
Hello world!
>>> print("Hello", "world!")
Hello world!
>>> print("Hello", "world", end="!")
```

Using a comma (,) puts a space between the words

Tell me more!

We can `print` things in lots of different ways in python!

```
>>> print("Hello world!")
Hello world!
>>> print("Hello", "world!")
Hello world!
>>> print("Hello", "world", end="!")
Hello world!
```

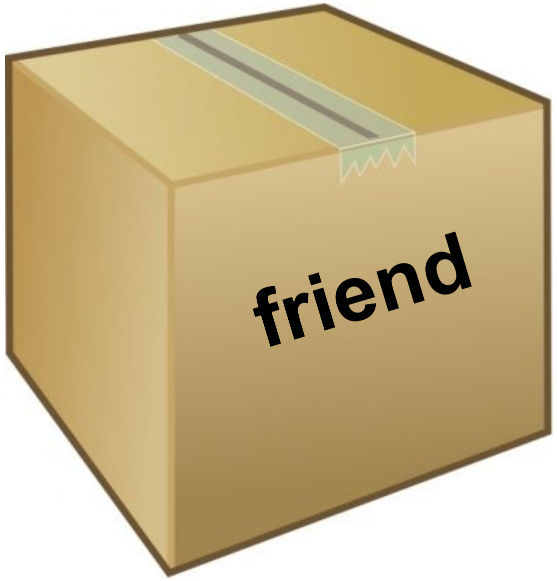
Note that this last one will not have a new line after it!

Variables

Variables are useful for storing things that change

(i.e. things that "vary" - hence the word "variable")

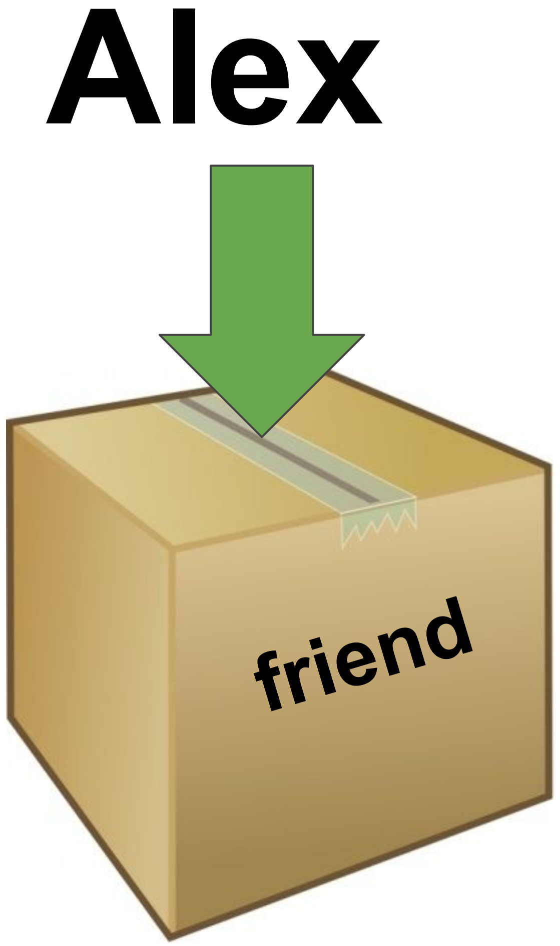
You can think of it like putting information in a box and giving it a label



Variables

When coding, we can make a variable called **friend** and set it to a value like this

friend = “Alex”

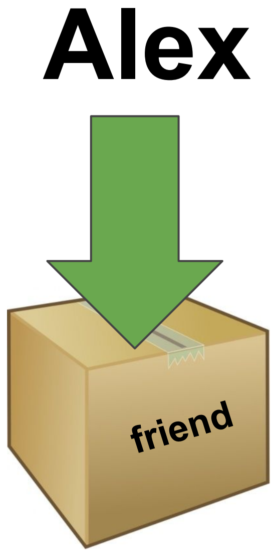


Variables

Instead of writing the word "Alex", we can write **friend** (the variable's name).

The computer will substitute the current value of friend.

It's like we're getting the value out of the box!



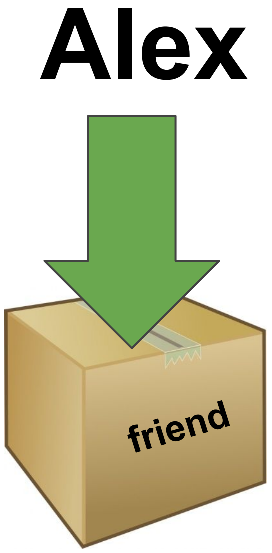
```
print(friend)
```

Variables

Instead of writing the word “Alex”, we can write **friend** (the variable’s name).

The computer will substitute the current value of friend.

It’s like we’re getting the value out of the box!



```
print(friend)
```

```
Alex
```

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

Store the answer in the variable my_name

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

What do you think happens?

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

Breaking it down

Store the answer in the variable my_name

Writing input tells the computer to wait for a response

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

What do you think happens?

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

Breaking it down

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)
```

What do you think happens?

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

Breaking it down

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

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

What do you think happens?

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

We can use the answer the user wrote that we then stored later!

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 decisions.

First we test if the condition is met!

Then maybe we'll do the thing

<u>If it's raining</u> 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

```
5 < 10                 "Dog" == "dog"
3 + 2 == 5             "D" in "Dog"
5 != 5                 "Q" not in "Cat"
```

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

```
5 < 10          True          "Dog" == "dog"
3 + 2 == 5                    "D" in "Dog"
5 != 5                        "Q" not in "Cat"
```

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

```
5 < 10          True          "Dog" == "dog"
3 + 2 == 5      True          "D" in "Dog"
5 != 5                        "Q" not in "Cat"
```

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

5 < 10	True	"Dog" == "dog"
3 + 2 == 5	True	"D" in "Dog"
5 != 5	False	"Q" not in "Cat"

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

5 < 10	True	"Dog" == "dog"	False
3 + 2 == 5	True	"D" in "Dog"	
5 != 5	False	"Q" not in "Cat"	

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

5 < 10	True	"Dog" == "dog"	False
3 + 2 == 5	True	"D" in "Dog"	True
5 != 5	False	"Q" not in "Cat"	

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

5 < 10	True	"Dog" == "dog"	False
3 + 2 == 5	True	"D" in "Dog"	True
5 != 5	False	"Q" not in "Cat"	True

Conditions

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

```python
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 fave_num < 10:
    print("that's a small number")
```

That's the condition!

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")
```

Put in the answer to the question

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")
```

What do you think happens?

```
>>>
```

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")
```

<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")
```

Conditions

Find out if it's `True`!

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

What should this say?

Conditions

Find out if it's `True`!

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

What should this say?

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

- Well, fave_num is 9000
- And it's not `True` that 9000 is less than 10
- So it is `False`!

Conditions

Find out if it's `True`!

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

What should this say?

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

- Well, fave_num is 9000
- And it's not `True` that 9000 is less than 10
- So it is `False`!

Conditions

Find out if it's True!

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

What do you think happens?

```
>>>
```

Conditions

Find out if it's True!

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

What do you think 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!!")
```

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

```
>>>
```

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

```python
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!
```

Remember ...

==

When testing for equals in your condition

:

At end of each if line to say you have finished writing your condition

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!

Bread

Ice Cream

Chocolate

Pizza

We put it in one big shopping list!

- Bread
- Chocolate
- Ice Cream
- Pizza

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"]
```

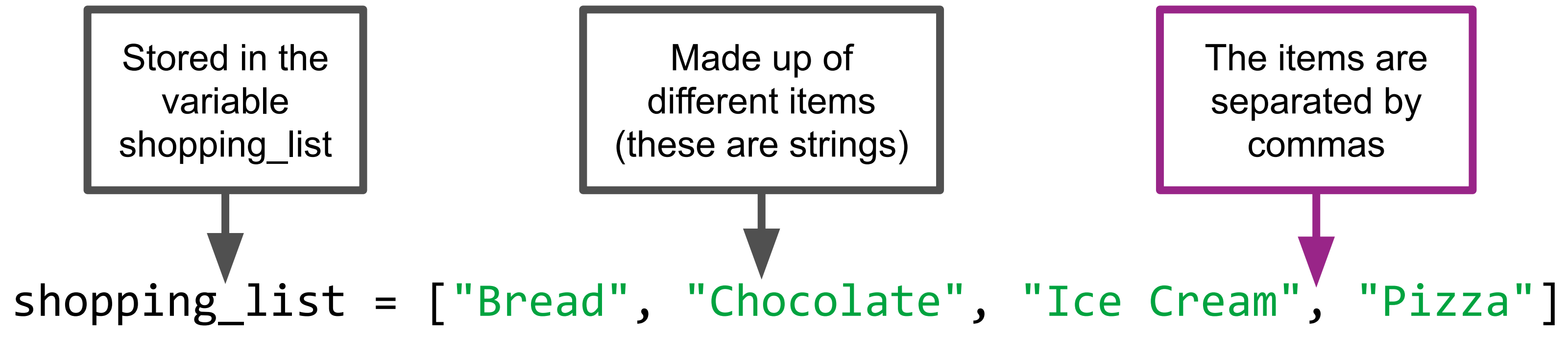
List anatomy

Stored in the variable shopping_list

Made up of different items (these are strings)

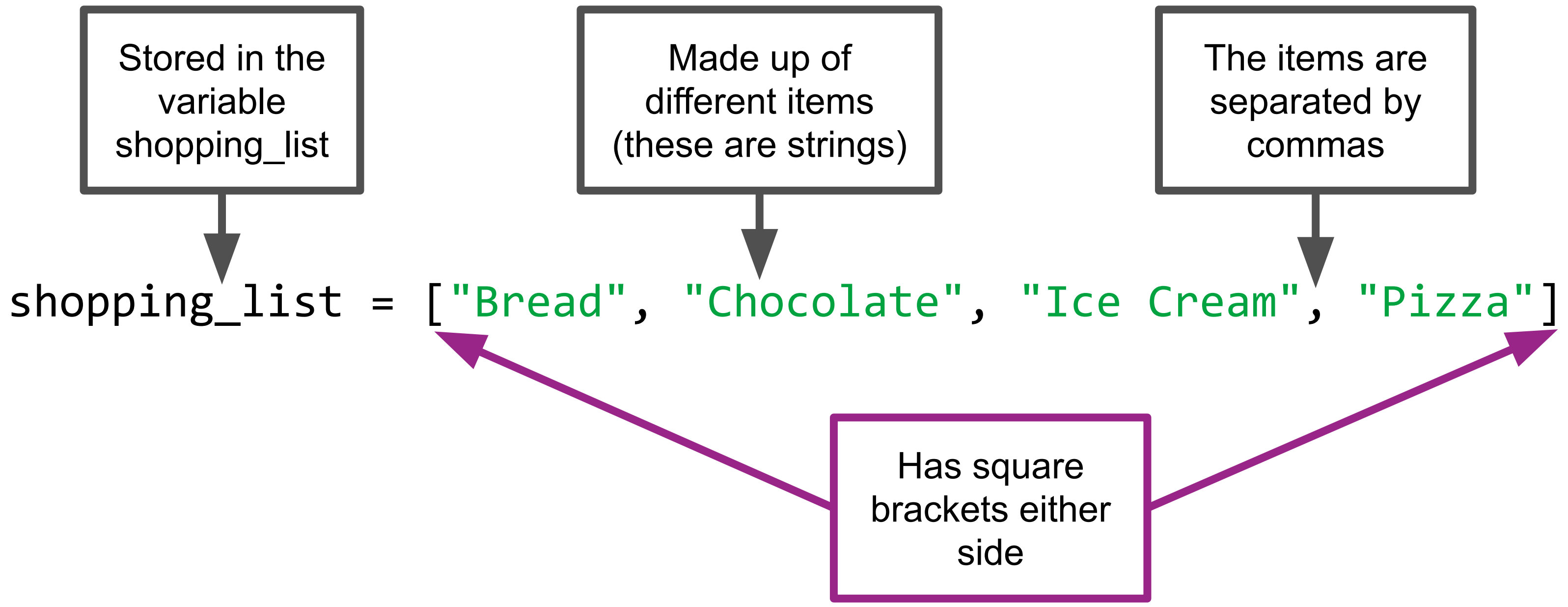
```
shopping_list = ["Bread", "Chocolate", "Ice Cream", "Pizza"]
```

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"]
```

Lists

You can print a list

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

```
>>> ["Bread", "Chocolate", "Ice Cream", "Pizza"]
```

What’s an index?

A list has many items, indexing lets us get one item from the list using its position number;

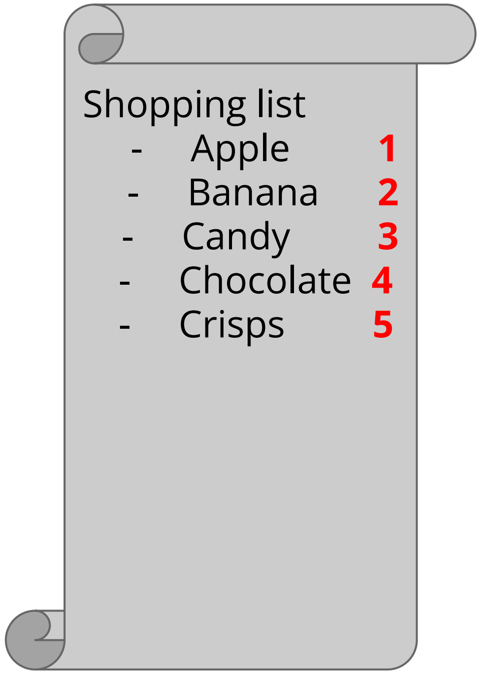
What is the fourth item I need to buy?

Shopping list

- Apple
- Banana
- Candy
- Chocolate
- Crisps

What's an index?

A list has many items, indexing lets us get one item from the list using its position number;



Shopping list

-	Apple	**1**
-	Banana	**2**
-	Candy	**3**
-	Chocolate	**4**
-	Crisps	**5**

What is the fourth item I need to buy?

chocolate!

How do I know how long

We use indexes (or the position number) to pick an item in a list

```
fruits = ["apple", "banana", "cherry"]
fruits[1]
```

The list we want to pick an item from

How do I know how long

We use indexes (or the position number) to pick an item in a list

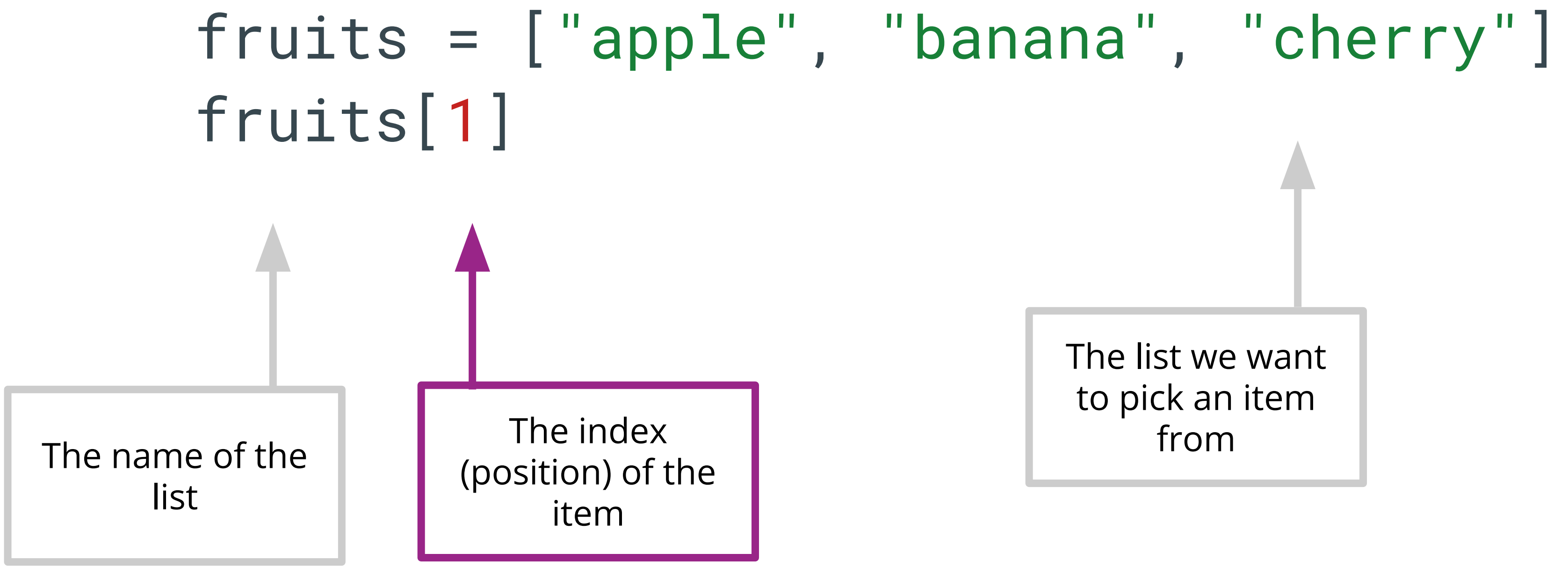
```
fruits = ["apple", "banana", "cherry"]
fruits[1]
```

The name of the list

The list we want to pick an item from

How do I know how long

We use indexes (or the position number) to pick an item in a list



```
fruits = ["apple", "banana", "cherry"]
fruits[1]
```

The name of the list
The index (position) of the item
The list we want to pick an item from

But wait!

When we index, we start counting from 0

```
                0            1            2
fruits = ["apple", "banana", "cherry"]
fruits[1]
```

So we are actually picking the item “banana”

Project Time!

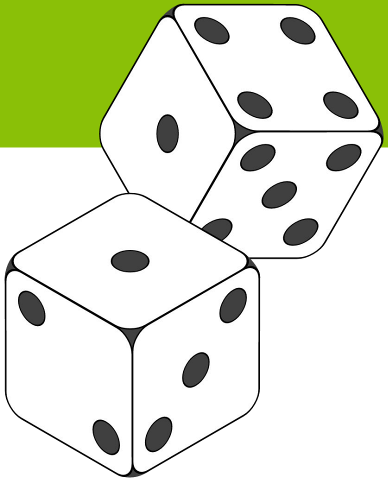
You now know all about `if` and **lists**!

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!

We use **random.choice** to randomly select something from a list

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

Each time we run this we would probably get a different answer

```
eggs OR bread OR apples OR milk
```

Using the random module

You can also assign your random choice to a variable and then use that variable in your code

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

The variable **random_food** contains the random choice that was made from the list

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 help!

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?

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

Looping how many times?

We can loop through a list:

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

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

What’s going to happen?

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

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!