Welcome to the Labs!

Tic Tac Toe

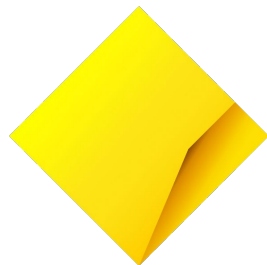# Thank you to our Sponsors!

Platinum Sponsor:

ATLASSIAN

amazon

Gold Sponsor:

Commonwealth Bank

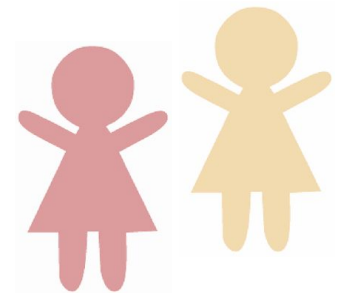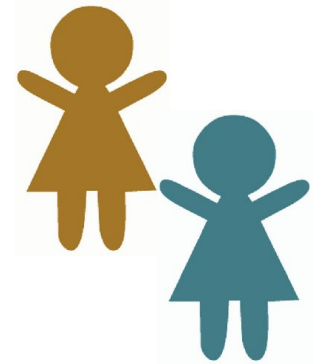# Who are the tutors?

Tech
Inclusion

Who are you?

Tech
Inclusion

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

Choose your location

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
Start of Day Survey
and fill it in now!

Today's project!

Tic Tac Toe!

Tech
Inclusion

This is what it'll look like!

```
Welcome to Tic-Tac-Toe!
Who is playing naughts?
```

Running

Tech
Inclusion

Introduction to Edstem

Signing up to Edstem

We are shifting all our courses to a new website called "Edstem" so here's an overview of how to sign up and how to use it.

First let's go through how to create an account.

1. Follow this link: https://edstem.org/au/join/b6XZzt
2. Type in your name and your personal email address
3. Click Create Account
4. Go to your email to verify your account
5. Create a password
6. It should then take you to the courses home page.
7. Click on the one we will be using for this project: ⟶

Tic Tac Toe P
Tic Tac Toe

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

Tech
Inclusion

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 to Tic Tac Toe!** They have an icon that looks like this:

3. To complete your project, we will work through the chapters one at a time starting with 1 and continuing on.

Tech
Inclusion

Inside a Chapter

Inside a chapter there are two main types of pages:

1. Lesson pages

   The lessons are where you will do your coding. They are called something like **1.1 Welcome!** and have this icon: <>

2. Checkpoints

   Each chapter has a checkpoint. Complete the checkpoint to move to the next chapter. Make sure you scroll down to see all the questions listed. Checkpoints look like this: ☑ Checkpoint

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.
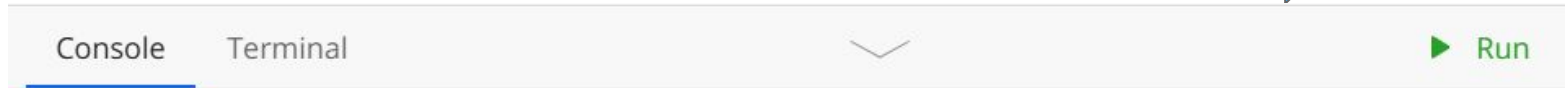
Running your code…

To run your code, you will need to click the button that says Run. It should run automatically and any outputs should be in the "Console" page. You can click the button again to rerun your code.

It should look like this;

Don't worry if you forget. Tutors will help!

Console   Terminal   ⌄   ▶ Run

Tech
Inclusion

Some shortcuts...

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

- **Ctrl + A**   Pressing these keys together will select all the text on a page

- **Ctrl + C**   Pressing these keys together will copy anything that's selected

- **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. You should also have a look at part 0 of your workbook

Remember the tutors will be around to help!

Tech Inclusion

Intro to Python

Let's get coding!

Tech Inclusion

Let's make a mistake!
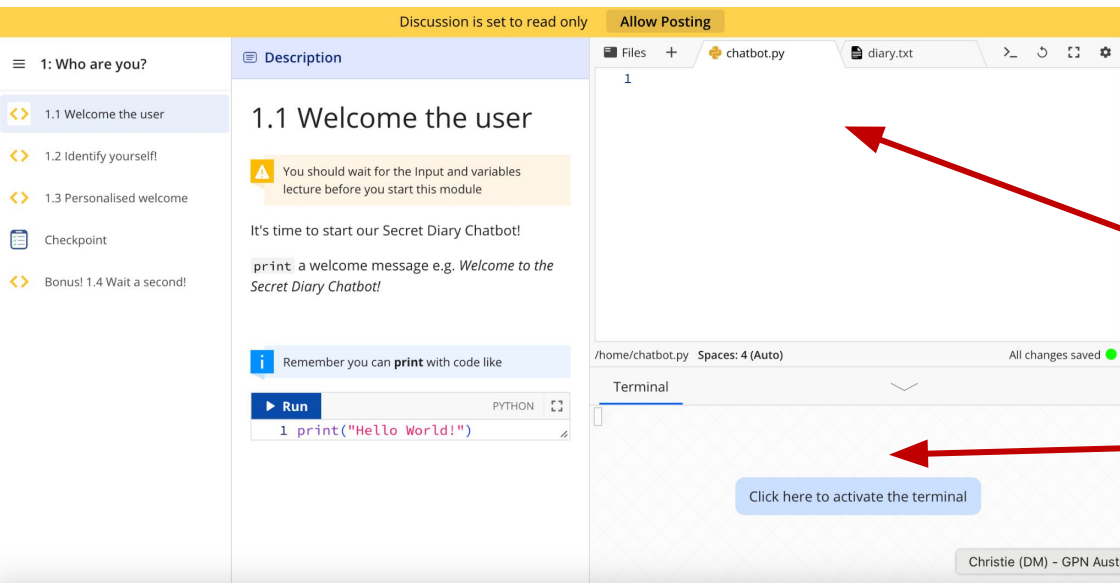
1. Type by **button mashing** the keyboard here e.g.

   `ks@674dbkjSDfkl`

2. **Click** in the Terminal panel here to run your code!

Did you get a big ugly error message?

Good work! You made an error!

- Programmers make a lot of errors!
- Errors give us hints on how to fix.
- Run your code often to get the hints.
- Mistakes <u>won't</u> break computers.
- Some of the errors you may see…

I ❤ errors!
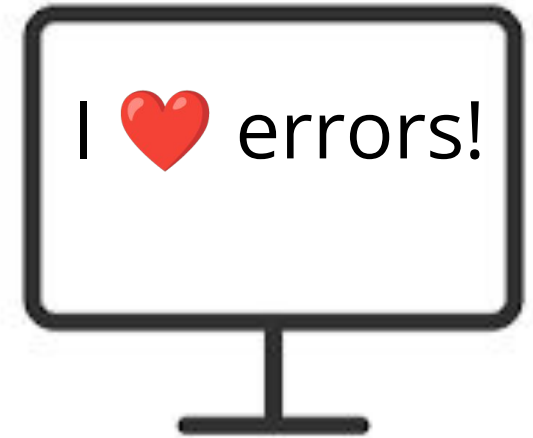
```
KeyError: 'Hairy
Potter'
```

```
ImportError: No
module named
humour
```

```
SyntaxError:
Invalid Syntax
```

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

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

Tech
Inclusion

We can learn from our mistakes!

```
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. Which bit of code didn't work

3. Where that code is

We read error messages from bottom to top.
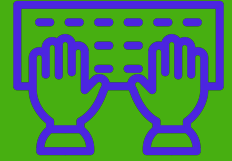
Tech Inclusion

A calculator for words!

What do you think these bits of code do?
Try them! Run the code after typing in each example.

- ```python
  print("cat" + "dog")
  ```

- ```python
  print("tortoise" * 3)
  ```

A calculator for words!
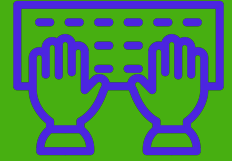
What do you think these bits of code do?
Try them! Run the code after typing in each example.

- ```
  print("cat" + "dog")
  ```
  ```
  catdog
  ```
- ```
  print("tortoise" * 3)
  ```

Tech
Inclusion

A calculator for words!

What do you think these bits of code do?
Try them! Run the code after typing in each example.

- ```
  print("cat" + "dog")
  ```
  ```
  catdog
  ```
- ```
  print("tortoise" * 3)
  ```
  ```
  tortoisetortoisetortoise
  ```

Strings!
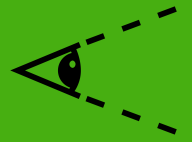
Strings are things with **"quotes"**

Strings can have any letters in them, even spaces!

```
        "Hello, world!"

                        "bla bla bla"

  ":)"      " "

              'I can use single quotes too!'

      "¯\_(ツ)_/¯"

                      "asdfghjklqwertyuiopzxcvbnm"

"DOGS ARE AWESOME!"

          "!@#$%^&*()_+-=[]|\:;'<>,./?"
```

Tech Inclusion

Strings and Ints!

Integers are numbers in python.

We can do maths with integers but not strings

```
>>> 5 + "5"
```

We can turn a string into an integer using int()

```
>>> 5 + int("5")
```

Similarly, we turn an integer into a string using str()

```
>>> str(5) + "5"
```

Tech
Inclusion

Strings and Ints!

Integers are numbers in python.

We can do maths with integers but not strings

```
>>> 5 + "5"
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

We can turn a string into an integer using int()

```
>>> 5 + int("5")
```

Similarly, we turn an integer into a string using str()

```
>>> str(5) + "5"
```

Tech
Inclusion

Strings and Ints!

Integers are numbers in python.

We can do maths with integers but not strings

```
>>> 5 + "5"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

We can turn a string into an integer using int()

```
>>> 5 + int("5")
10
```

Similarly, we turn an integer into a string using str()

```
>>> str(5) + "5"
```

Strings and Ints!

Integers are numbers in python.

We can do maths with integers but not strings

```
>>> 5 + "5"
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

We can turn a string into an integer using int()

```
>>> 5 + int("5")
10
```

Similarly, we turn an integer into a string using str()

```
>>> str(5) + "5"
'55'
```

Tech Inclusion

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 a 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!

You now know all about the building blocks of Python!

Let's put what we learnt into our project

Try to do the next Part!

The tutors will be around to help!

Lists

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

Chocolate
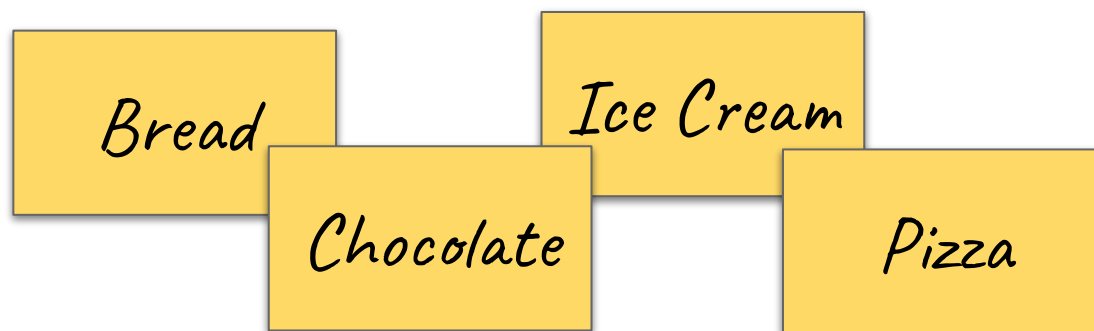
Ice Cream

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"]
```

Tech
Inclusion

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
['books', 'butterfly', 'lollipops', 'skateboard']
>>> faves.remove('butterfly')
```

What does this list look like now?

```
['books', 'lollipops', 'skateboard']
```

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"]
```

Has square brackets

Tech Inclusion
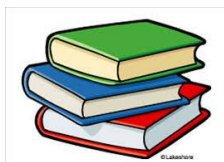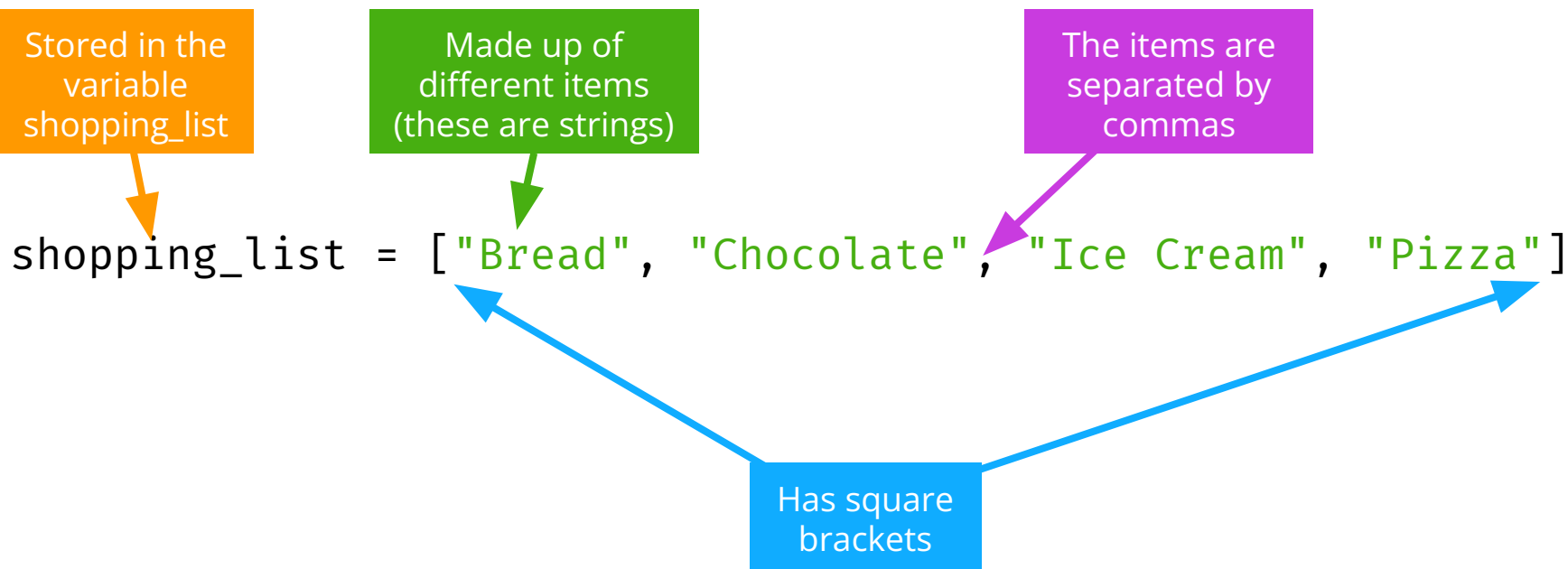
Accessing Lists!

The favourites `list` below 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!

Tech
Inclusion

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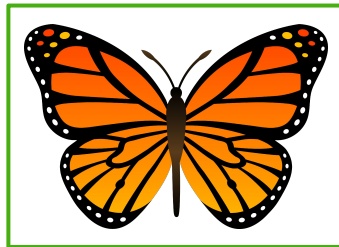
What code do you need to access the second item in the list?

```
>>> faves[1]
'butterfly'
```
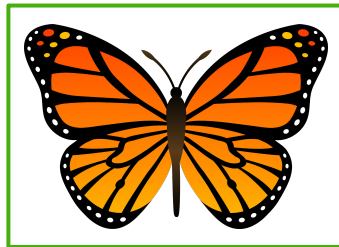
| 0 | **[1]** | 2 | 3 |

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'
```
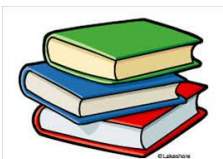
-4   -3   **[-2]**   -1

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
```

Tech
Inclusion

Updating items!

We can also update things in a list:
```
>>> faves = ['books', 'butterfly',
                'chocolate', 'skateboard']
>>> faves[2]
'chocolate'
>>> faves[2] = 'lollipops'
>>> faves
```

Updating items!

We can also update things in a list:

```
>>> faves = ['books', 'butterfly',
              'chocolate', 'skateboard']
>>> faves[2]
'chocolate'
>>> faves[2] = 'lollipops'
>>> faves
['books', 'butterfly', 'lollipops', 'skateboard']
```

Tech
Inclusion

List of lists!

You really can put anything in a list, even more lists!

We could use a list of lists to store different sports teams!

```
tennis_pairs = [
    ["Alex", "Emily"], ["Kass", "Annie"], ["Amara", "Viv"]
]
```

Get the first pair in the list

```
>>> first_pair = tennis_pairs[0]
>>> ["Alex", "Emily"]
```

Now we have the first pair handy, we can get the first the first player of the first pair

```
>>> fist_player = first_pair[0]
>>> "Alex"
```

Project time!

You now know all about lists!

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

The tutors will be around to help!

Tech
Inclusion

Functions!

Simpler, less repetition, easier to read code!

How functions fit together!

Functions are like factories!

Metal Worker

Your main factory!

Timber Mill
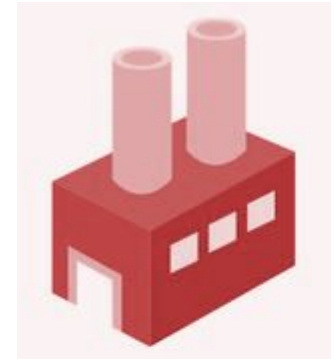
Cupcake factory

Running a factory doesn't mean doing all the work yourself, you can get other factories to help you out!

Tech Inclusion

How functions fit together!

Functions are like factories!

Timber Mill

Your main factory!

Metal Worker

I'd like to place an order for a piece of wood. 2 meters by 1.5 meters.

Can I order 4 metal poles please! 80cm long.

Order

Order

Sure thing! Coming right away!

Delivery

Delivery

It will be delivered straight away!

Cupcake factory
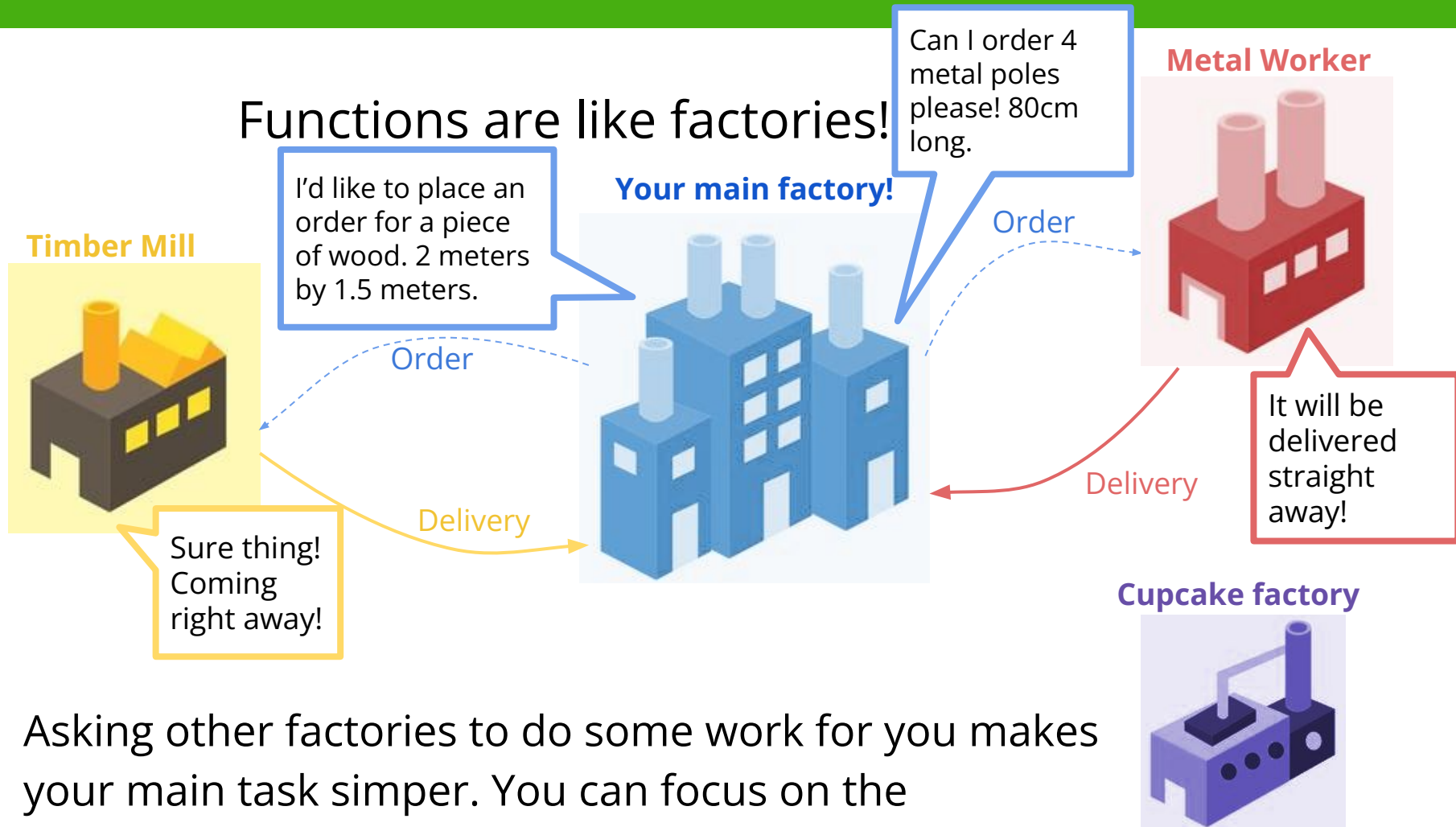
Asking other factories to do some work for you makes your main task simper. You can focus on the assembly!

How functions fit together!

Functions are like factories!

Timber Mill

Your main factory!

Metal Worker

Look at this beautiful table I made!

Outsourcing made it simple!

Cupcake factory

Tech Inclusion
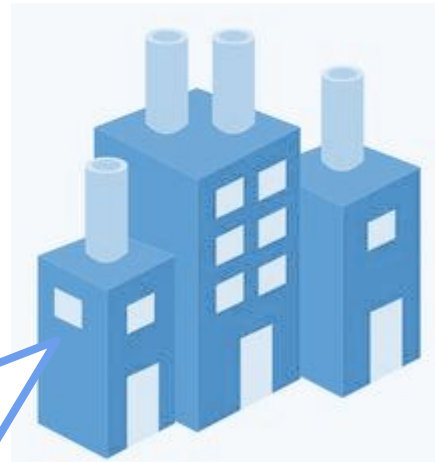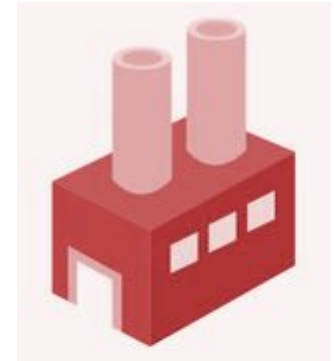
How functions fit together!

Your main code!

You can write a bunch of helpful functions to **simplify** your **main goal**!

You can **write** these **once** and then **use** them **lots** of times! They can be **anything** you like!

Helps with printing nicely

Uses stats to make decisions

Does calculations

Don't reinvent the wheel

We're already familiar with some python in built functions like print and input!

There's lots of functions python gives us to save us reinventing the wheel!

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")
11
```

Try these:

```
>>> name = "Renee"
>>> len(name)
5


>>> int("6")
6


>>> str(6)
"6"
```

Defining your own functions

Built in functions are great! But sometimes we want custom functions!

Defining our own functions means:

- We cut down on repeated code

- Nice function names makes our code clear and easy to read

- We can move bulky code out of the way

Defining your own functions

Then you can use your function by calling it!

```
def cat_print():
    print("""
                    #
                     #
                     #
            ^..^ ####
            =TT=        ;
             #########
             # #    # #
             M M   M M """)


cat_print()
cat_print()
```

Which will do this!

```
                #
                 #
                 #
        ^..^ ####
        =TT=        ;
         #########
         # #    # #
         M M    M M
                 #
                  #
                   #
        ^..^ ####
        =TT=        ;
         #########
         # #    # #
         M M    M M
```

Tech
Inclusion

Defining your own functions

Then you can use your function by calling it!

```
def cat_print():
    print("""


                    #
                     #
                     #
        ^..^ #####
        =TT=         ;
         #########
         # #    # #
         M M   M M """)
```

```
cat_print()
cat_print()
```

When using a function in a **script** make sure you define the function first.

It doesn't matter if you call it from inside another function though!

Which will do this!

```
                  #
                   #
                   #
      ^..^ #####
      =TT=         ;
       #########
       # #    # #
       M M    M M
                 #
                  #
                   #
      ^..^ #####
      =TT=         ;
       #########
       # #    # #
       M M    M M
```

Tech Inclusion

Functions often need extra information

Functions are more useful if we can change what they do

We can do this by giving them arguments (aka parameters)

```
>>> def hello(person):
...    print('Hello, ' + person + ', how are you?')
>>> hello('Alex')
Hello, Alex, how are you?
```

Here, we give the hello() function a name

Any string will work

```
>>> hello('abcd')
Hello, abcd, how are you?
```

Tech
Inclusion

Functions can take multiple arguments

Often we want to work with multiple pieces of information.

You can actually have as many parameters as you like!

This function takes two numbers, adds them together and prints the result.

```
>>> def add(x, y):
...     print(x + y)
>>> add(3, 4)
7
```

Arguments stay inside the function

The arguments are not able to be accessed outside of the function declaration.

```
>>> def hello(person):
...     print('Hello, ' + person + '!')
>>> print(person)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'person' is not defined
```

Variables stay inside the function

Neither are variables made inside the function. They are **local variables**.

```
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
>>> z
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'z' is not defined
```

Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

What's the value of z now?

```
>>> print(z)
```

Tech
Inclusion

Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

What's the value of z now?

```
>>> print(z)
1
```

Tech
Inclusion

Recap: A function signature

the `def` keyword

function **name**

function **arguments**

definition

```
def add(x, y):
```

function **name**

function **arguments**

callsite

```
>>> add(2, 3)
```

Tech
Inclusion
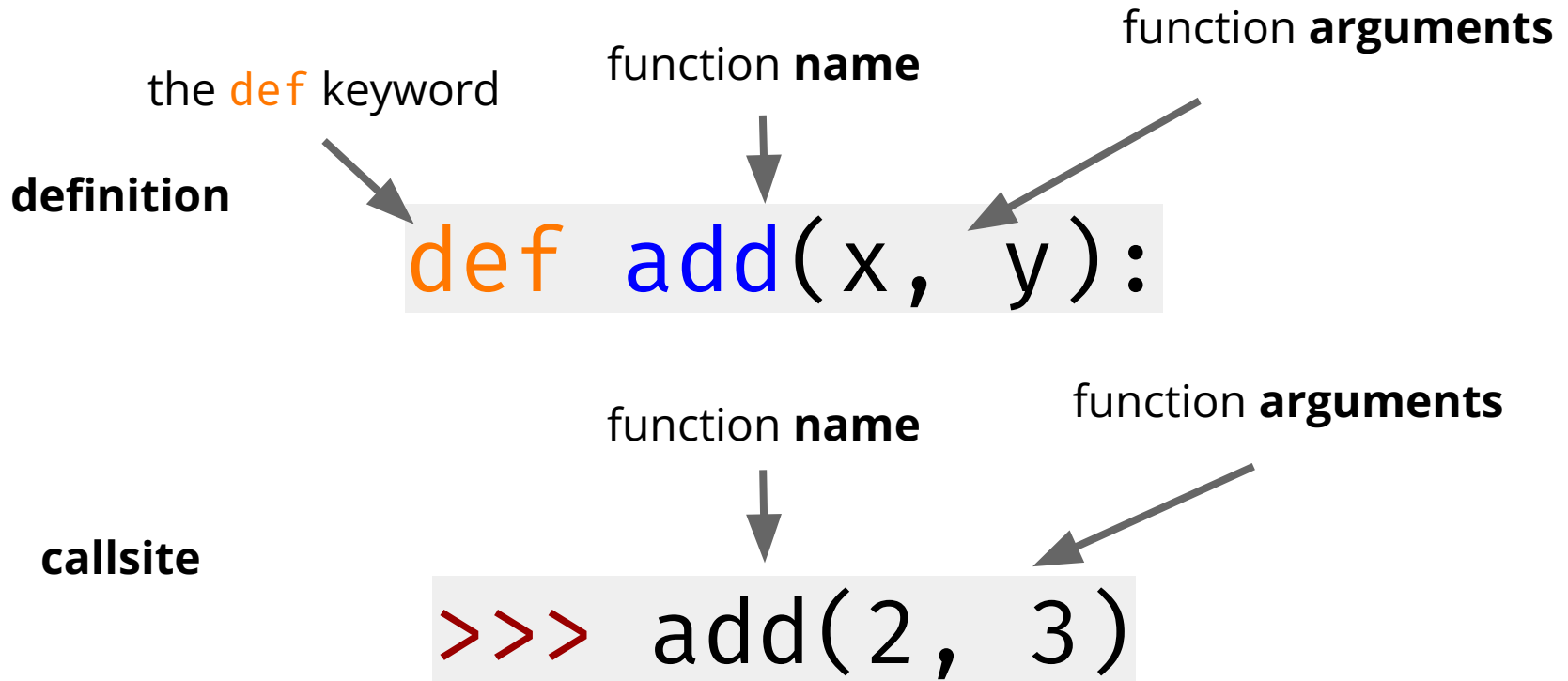
Giving something back

At the moment our function just does a thing, but it's not able to give anything back to the main program.

Currently, we can't use the result of add()

```
>>> def add(x, y):
...     print(x + y)
>>> sum = add(1, 3)
4
>>> sum
```

sum has no value!

Giving something back

Using `return` in a function immediately returns a result.

```
>>> def add(x, y):
...     z = x + y
...     return z
...
>>> sum = add(1, 3)
>>> sum
4
```

Tech
Inclusion

Giving something back

When a function returns something, the *control* is passed back to the main program, so no code after the `return` statement is run.

```
>>> def add(x, y):
...     print('before the return')
...     z = x + y
...     return z
...     print('after the return')
>>> sum = add(1, 3)
before the return
>>> sum
4
```

Here, the `print` statement after the `return` never gets run.

Project time!

Now go be functional.

Do the next part of the project!

Try to do Part 3

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

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

Tech Inclusion

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"

Tech
Inclusion

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"

Tech Inclusion

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")
```

Tech
Inclusion

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!

Tech Inclusion

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**!

Tech Inclusion

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")
```

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

```
>>>
```

Tech
Inclusion

Conditions

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

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

What do you think happens?

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

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!

Else statements

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

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

What happens?

Tech Inclusion

Else statements

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

```python
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

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

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

What happens?

Tech Inclusion

Elif statements

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

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

What happens?
>>> YUMMM Chocolate!

Tech Inclusion

Booleans (True and False)

Python has some special comparisons for checking if something is **in** something else. **Try these!**

```
>>> "A" in "AEIOU"
>>> "Z" in "AEIOU"
>>> "a" in "AEIOU"
```

```
>>> animals = ["cat", "dog", "goat"]
>>> "banana" in animals
>>> "cat" in animals
```

```
>>> phone_book = {"Maddie": 111, "Lucy": 222, "Julia": 333}
>>> "Maddie" in phone_book
>>> "Gabe" in phone_book
>>> 333 in phone_book
```

Tech
Inclusion

Booleans (True and False)

Python has some special comparisons for checking if something is **in** something else. **Try these!**

True ⊢ "A" in "AEIOU"

False ⊢ "Z" in "AEIOU"

False ⊢ "a" in "AEIOU"

```
>>> animals = ["cat", "dog", "goat"]
```
False "banana" in animals

True "cat" in animals

```
>>> phone_book = {"Maddie": 111, "Lucy": 222, "Julia": 333}
```
True "Maddie" in phone_book

False "Gabe" in phone_book

False 333 in phone_book

It only checks in the keys!

Tech Inclusion

Project Time!

You now know all about **if**!

See if you can do the next Part

The tutors will be around to help!

Tech Inclusion

While Loops

Tech
Inclusion

Loops

We know how to do things on repeat!

Sometimes we want to do some code on repeat!

Tech Inclusion

What do you think this does?

```python
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

What do you think this does?

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

```
i is 0

i is 1

i is 2

>>>
```

Introducing … `while` loops!

Stepping through a while loop…

Tech
Inclusion

One step at a time!

MY VARIABLES

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

i = 0

Set the variable

One step at a time!

MY VARIABLES

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

0 is less than 3 !

```
i = 0
```

Introducing … `while` loops!

One step at a time!

Print !

```
i = 0
while i < 3:
    ◆ print("i is " + str(i))
    i = i + 1
```

MY VARIABLES

```
i = 0
```

```
i is 0
```

One step at a time!

MY VARIABLES

```
i = 0
while i < 3:
    print("i is " + str(i))
 ◆ i = i + 1
```

~~i = 0~~
i = 1

UPDATE TIME !

```
i is 0
```

Introducing … `while` loops!

One step at a time!

Take it from the top !

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

```
i is 0
```

~~i = 0~~
i = 1

Tech
Inclusion

Introducing … `while` loops!

One step at a time!

1 is less than 3 !

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

~~i = 0~~
i = 1

```
i is 0
```

Introducing … `while` loops!

One step at a time!

MY VARIABLES

Print !

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

```
i = 0
i = 1
```

```
i is 0

i is 1
```

Tech Inclusion

Introducing … `while` loops!

One step at a time!

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

```
i is 0
i is 1
```

MY VARIABLES

~~i = 0~~
~~i = 1~~
i = 2

UPDATE TIME !

Tech Inclusion

Introducing … `while` loops!

One step at a time!

Take it from the top !

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

```
i is 0
i is 1
```

~~i = 0~~
~~i = 1~~
i = 2

Tech
Inclusion

Introducing … `while` loops!

One step at a time!

MY VARIABLES

2 is less than 3 !

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

i = 0
i = 1
i = 2

i is 0

i is 1

Tech Inclusion

One step at a time!

Print !

MY VARIABLES

~~i = 0~~
~~i = 1~~
i = 2

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

```
i is 0

i is 1

i is 2
```

Introducing … `while` loops!

One step at a time!

MY VARIABLES

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

~~i = 0~~
~~i = 1~~
~~i = 2~~
i = 3

UPDATE TIME !

```
i is 0

i is 1

i is 2
```

Introducing … `while` loops!

One step at a time!

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

MY VARIABLES

~~i = 0~~
~~i = 1~~
~~i = 2~~
i = 3

```
i is 0

i is 1

i is 2
```

Introducing … `while loops!`

One step at a time!

MY VARIABLES

3 IS NOT less than 3!

We are are done with this loop!

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

~~i = 0~~
~~i = 1~~
~~i = 2~~
i = 3

```
i is 0

i is 1

i is 2
```

Tech Inclusion

Introducing … `while` loops!

Initialise the loop variable

Loop condition

Code to repeat

```
i = 0
while i < 3:
    print("i is " + str(i))
    i = i + 1
```

Update the loop variable

Tech Inclusion

What happens when…..

What happens if we forget to update the loop variable?

```python
i = 0
while i < 3:
    print("i is " + str(i))
```

What happens when…..

What happens if we forget to update the loop variable?

```python
i = 0
while i < 3:
    print("i is " + str(i))
```

```
i is 0
i is 0
i is 0
i is 0
i is 0
i is 0
i is 0
i is 0
i is 0
i is 0
i is 0
i is 0
i is 0
```

Give me a break!

But what if I wanna get out of a loop early?

That's when we use the break keyword!

```python
number = 0
while number != 42 :
    number = input("Guess a number: ")

    if number = "I give up":
        print("The number was 42")
        break

    number = int(number)
```

Tech
Inclusion

Continuing on

How about if I wanna skip the rest of the loop body and loop again? We use continue for that!

```python
number = 0
while number != 42 :
    number = input("Guess a number: ")

    if not number.isnumeric():
        print("That's not a number!")
        print("Try again")
        continue

    number = int(number)
```

Project Time!

while we're here:

Try to do the next Parts!

The tutors will be around to help!

Tech
Inclusion

For Loops

Tech
Inclusion

Looping through lists!

What would we do if we wanted to print out this list, one word at a time?

```python
words = ['This', 'is', 'a', 'sentence']

print(words[0])
print(words[1])
print(words[2])
print(words[3])
```
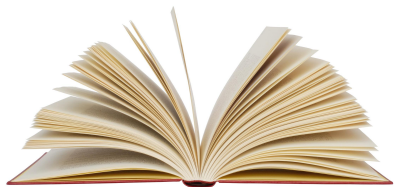
What if it had a 100 items??? That would be **BORING!**

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 page

For each chip in this bag of chips:
   Eat chip

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?

Tech
Inclusion

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!

Looping over a list of ints

Strings are lists of letters!

```
word = "cat"
for i in word:
    print(i)
```

What's going to happen?

Looping over a list of ints

Strings are lists of letters!

```
word = "cat"
for i in word:
    print(i)
```

What's going to happen?
```
>>> c
>>> a
>>> t
```

How does it work??

Somehow it knows how to get one fruit out at a time!!

It's like it knows english!

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

But fruit is just a variable! We could call it anything! Like dog!

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

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

Tech Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

Let's set <u>dog</u> to to the first thing in the list!
dog is now 'apple'!

Tech
Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

>>> Yummy apple

Let's set <u>dog</u> to to the first thing in the list!
dog is now 'apple'!
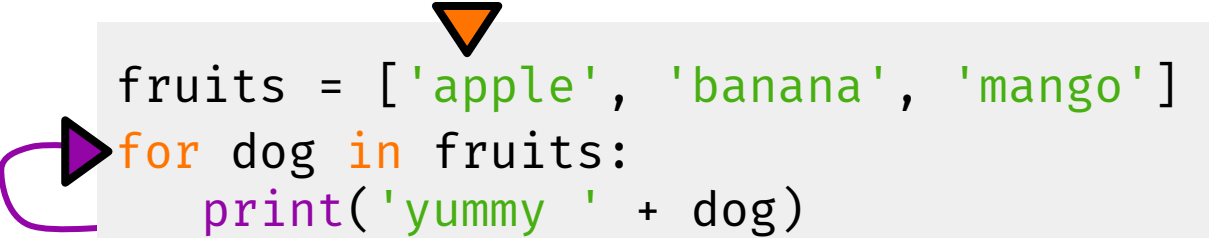print('yummy ' + dog)

Tech
Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

>>> Yummy apple

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

Girls' Programming Network

Tech
Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

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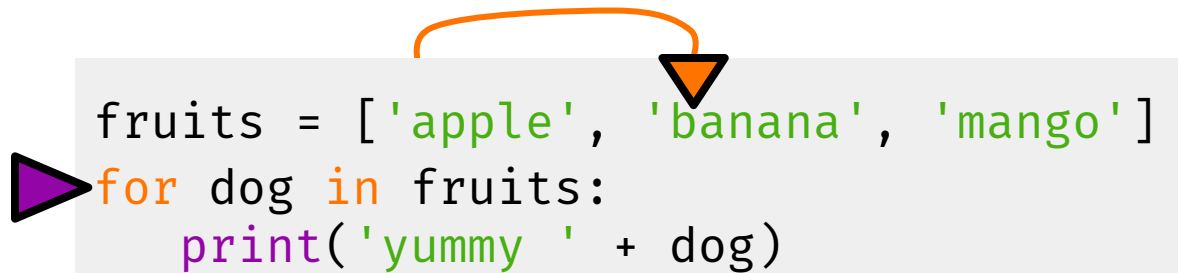
Let's set dog to to the first thing in the list!
dog is now `'apple'`!
print('yummy ' + dog)
We're at the end of the loop body, back to the top!

Let's set <u>dog</u> to to the next thing in the list!
dog is now `'banana'`!

Tech Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

>>> Yummy apple

>>> Yummy banana

Let's set <u>dog</u> to to the first thing in the list!
dog is now 'apple'!
```python
print('yummy ' + dog)
```
We're at the end of the loop body, back to the top!

Let's set <u>dog</u> to to the next thing in the list!
dog is now 'banana'!
```python
print('yummy ' + dog)
```

Tech Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

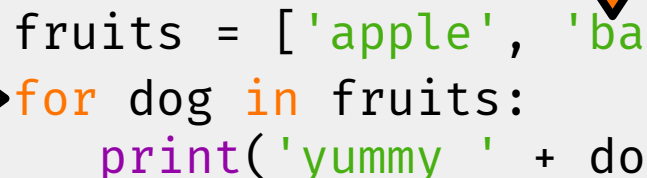>>> Yummy apple

>>> Yummy banana

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

Let's set <u>dog</u> to to the next thing in the list!
dog is now 'banana'!
print('yummy ' + dog)
Out of body, back to the top!

Tech Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

>>> Yummy apple

>>> Yummy banana

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

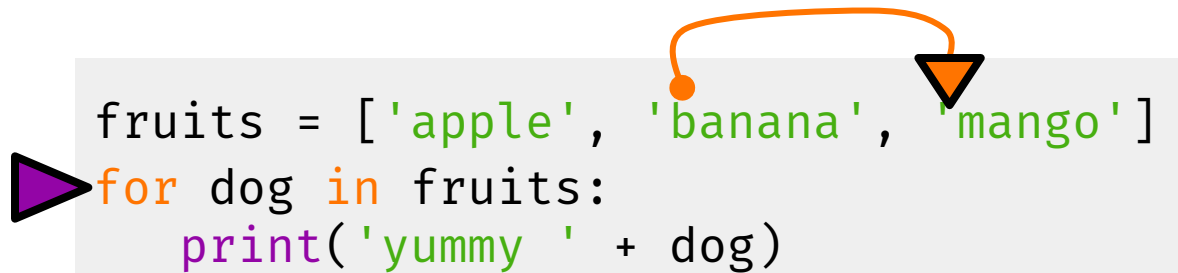**Let's set <u>dog</u> to to the next thing in the list!**
dog is now 'banana'!
print('yummy ' + dog)
Out of body, back to the top!

Let's set <u>dog</u> to to the next thing in the list!
dog is now 'mango'!

Tech
Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

>>> Yummy apple

>>> Yummy banana

>>> Yummy mango

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

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

Let's set <u>dog</u> to to the **next** thing in the list!
dog is now 'mango'!
print('yummy ' + dog)

Tech
Inclusion

How does it work??

Everything in the list gets to have a turn at being the <u>dog</u> variable

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

>>> Yummy apple

>>> Yummy banana

>>> Yummy mango

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

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

Let's set <u>dog</u> to to the **next** thing in the list!
dog is now 'mango'!
print('yummy ' + dog)
Out of body, and out of list!! We're done here!

Tech Inclusion

Project Time!

Now you know how to use a for loop!

Try to do Part 5
...if you are up for it!
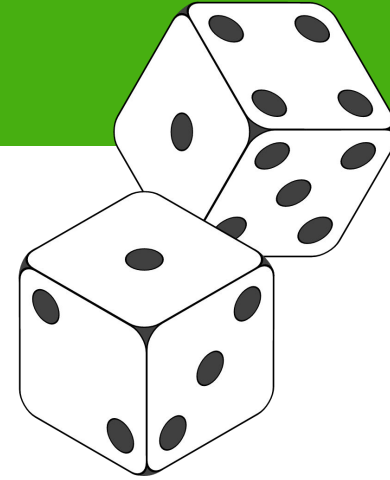
The tutors will be around to help!

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!

Tech
Inclusion

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

3. Choose randomly! Try it a few times!

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

Tech
Inclusion

Using the random module

You can also assign your random choice to a variable

```python
>>> import random
>>> shopping_list = ["Bread", "Chocolate", "Ice Cream",
    "Pizza"]
>>> random_food = random.choice(shopping_list)
>>> print(random_food)
```

Raaaaaaaaaandom! Can you handle that?

Let's try use it in our project!

Try to do the next Part

The tutors will be around to help!