

# Welcome to the Labs!

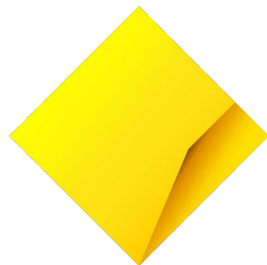
## Tic Tac Toe

# Thank you to our Sponsors!

Platinum Sponsor:



Gold Sponsor:



**Commonwealth  
Bank**

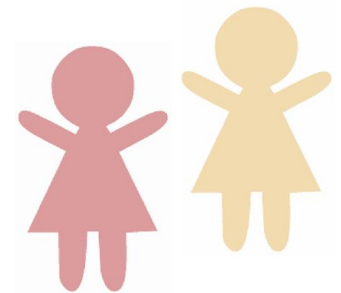
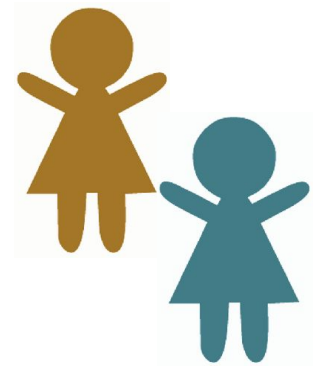


# Who are the tutors?

Who are you?

# Two Truths and a Lie

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



# Log on

## Log on and jump on the GPN website

[girlsprogramming.network/workshop](https://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!



# This is what it'll look like!

Welcome to Tic-Tac-Toe!

Who is playing naughts?

I

▶ Running

# 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*

# 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.

# 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:



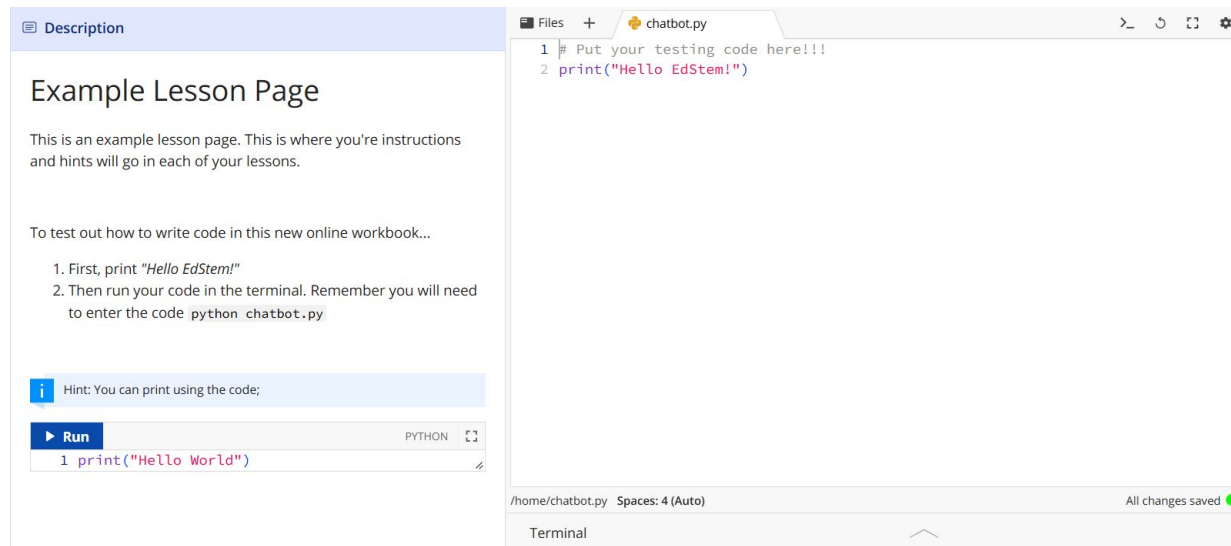
Checkpoint

# 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

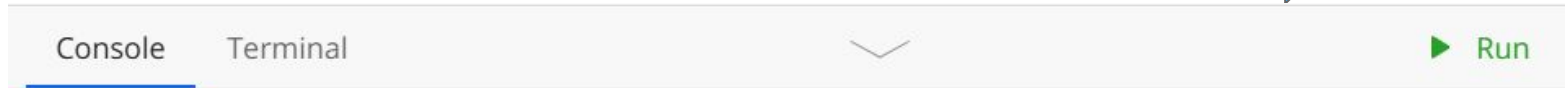
You will need to **copy your code from the last lesson**, then follow the instructions to change your code so that you can work towards finishing the project.



# 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;**





# 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**

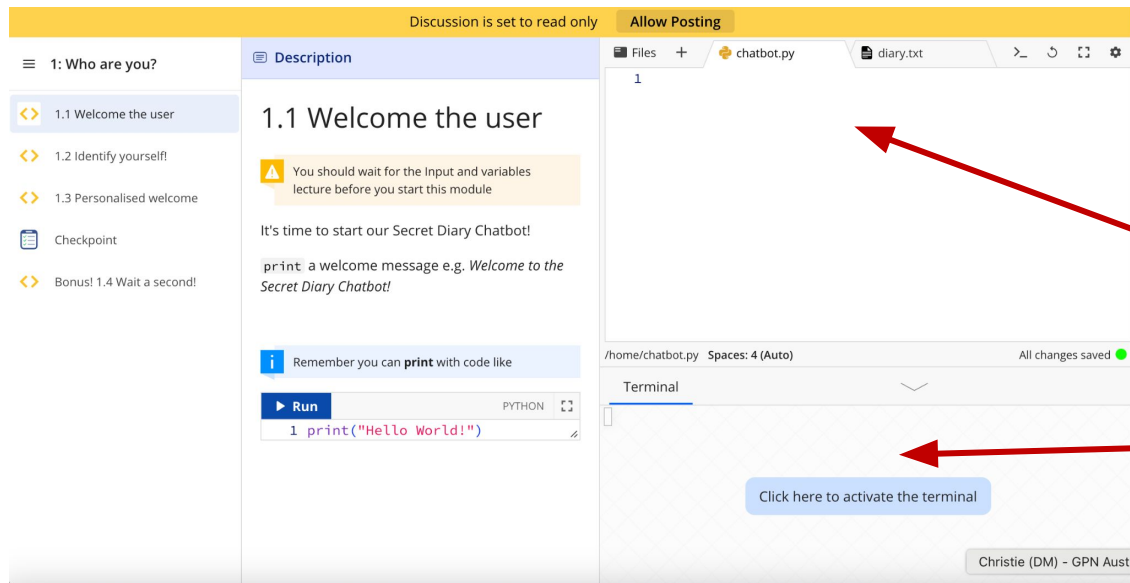
Remember the tutors will be around to help!

# Intro to Python

Let's get coding!



# Let's make a mistake!

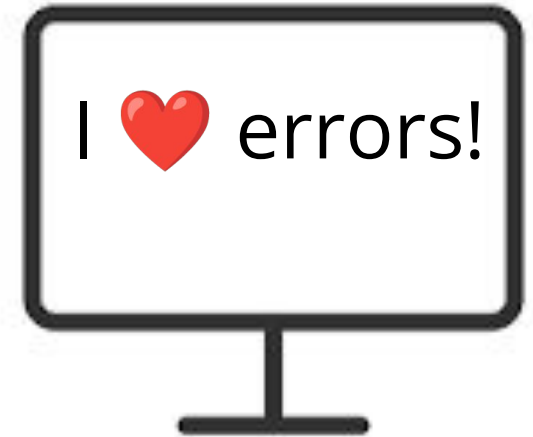


1. Type by **button mashing** the keyboard here e.g.  
`ks@674dbkjSDfk1`
2. **Click** in the Terminal panel here to run your code!

## 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 won't break computers.
- Some of the errors you may see...



```
KeyError: 'Hairy  
Potter'
```

```
SyntaxError:  
Invalid Syntax
```

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

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

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



# 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
```



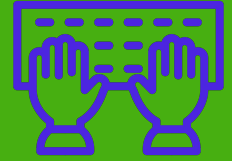
1. What went wrong

2. Which bit of code didn't work

3. Where that code is

We read error messages from bottom to top.

# A calculator for words!

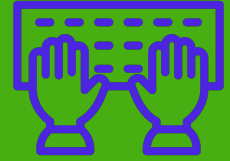


What do you think these bits of code do?

Try them! Run the code after typing in each example.

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

# A calculator for words!



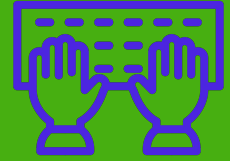
What do you think these bits of code do?

Try them! Run the code after typing in each example.

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



# A calculator for words!

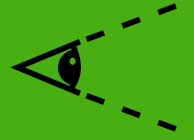


What do you think these bits of code do?

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!"
```

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

# 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"
```

# 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"
```

# 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'
```

# 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!

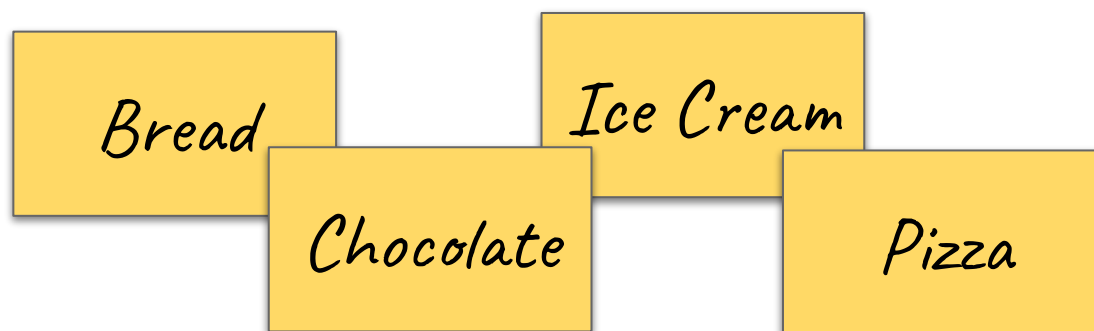


# Lists

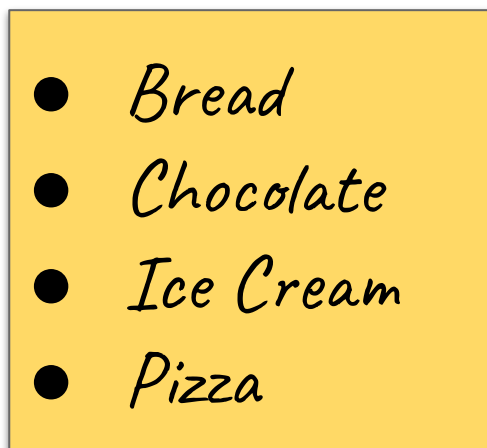
# Lists

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!



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"]
```

# 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?

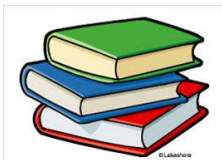
```
>>> faves
```

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

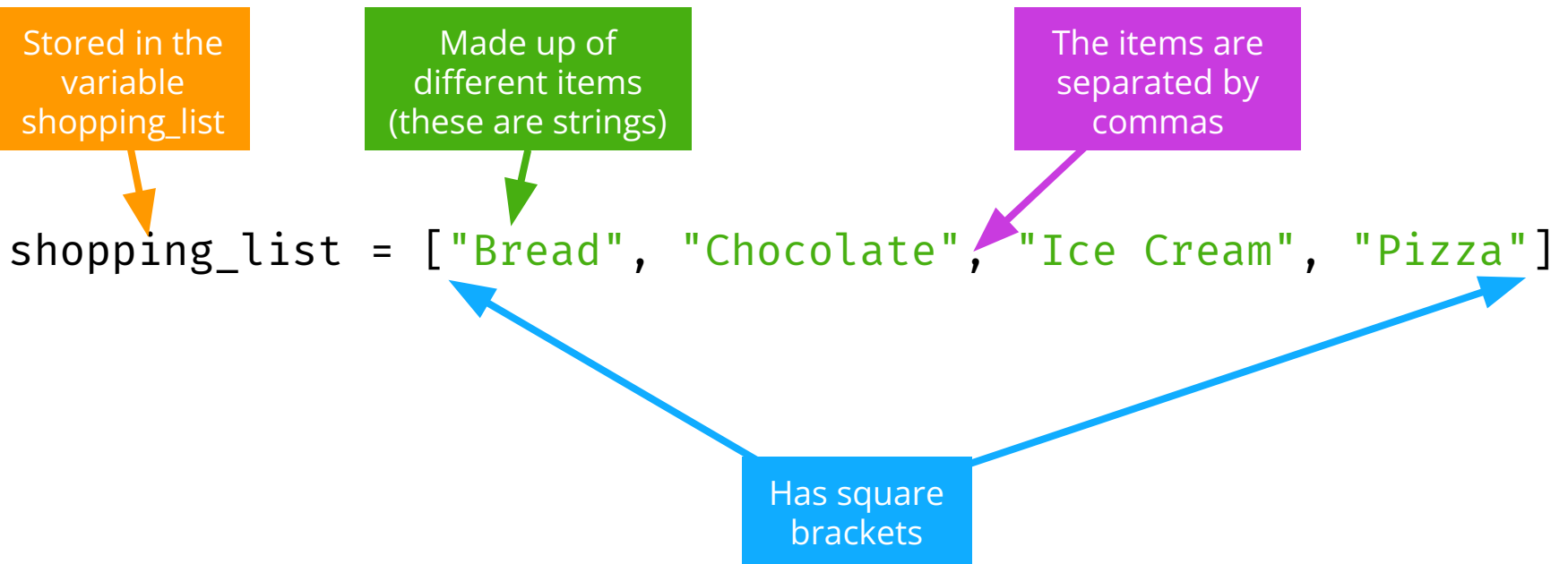
```
>>> faves.remove('butterfly')
```

What does this list look like now?

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



# List anatomy



# 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!**

# 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

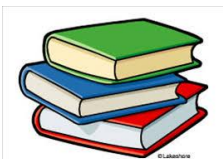
Negative indices count backwards from the end of the **list**:

```
>>> 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 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']
```



# 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

```
>>> first_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!

# Functions!

Simpler, less repetition, easier to read code!



# How functions fit together!

Functions are like factories!

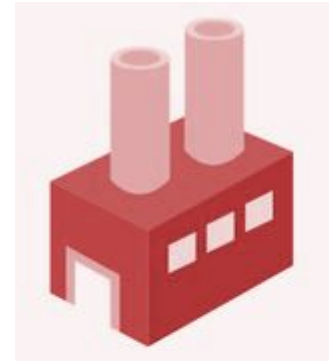
**Your main factory!**



**Timber Mill**



**Metal Worker**



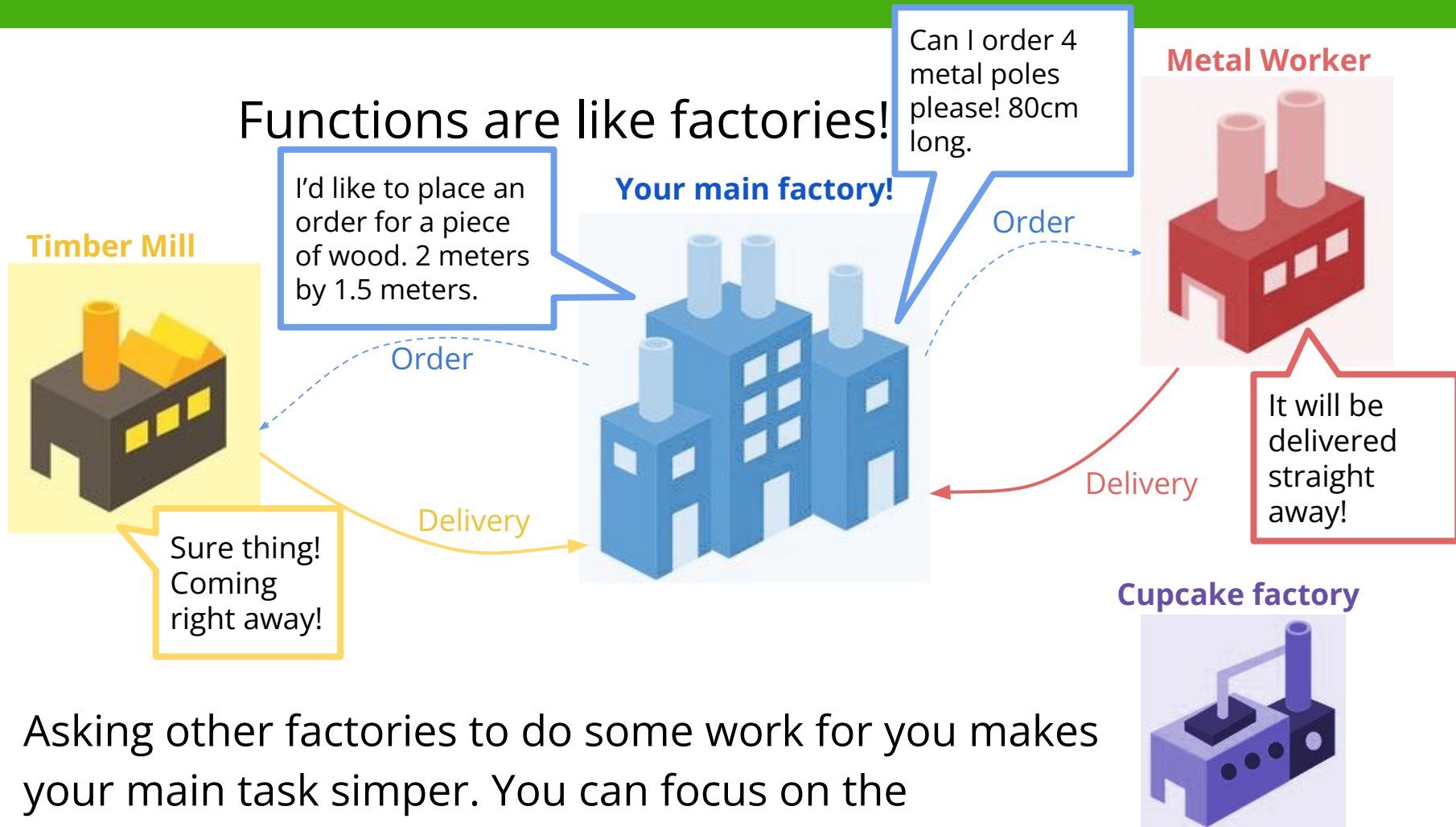
**Cupcake factory**



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



# How functions fit together!



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

# How functions fit together!

Functions are like factories!

Your main factory!

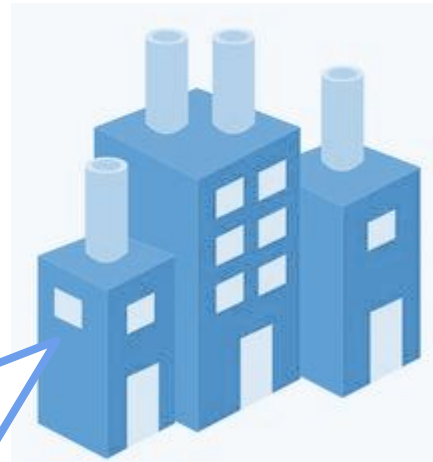
Timber Mill



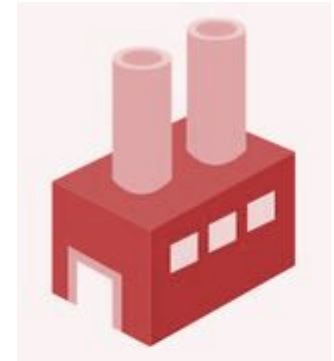
Look at this beautiful table I made!



Outsourcing made it simple!



Metal Worker



Cupcake factory



# 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
```



# Defining your own functions

## Then you can use your function by calling it!

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

```

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

```
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!

$$\begin{array}{ccccccc} & & & & & & \# \\ & & & & & & \\ & & & & & & \# \\ & & & & & & \# \\ \wedge & . & . & \wedge & & \# & \# & \# & \# & \# \\ = & T & T & = & & & & & & ; \\ & & & & & \# & \# & \# & \# & \# & \# & \# & \# \\ & \# & \# & & & \# & \# & & & & & \\ M & M & & & & M & M & & & & & \\ & & & & & & & & & & & \# \\ & & & & & & & & & & & \\ & & & & & & & & & & & \# \\ & & & & & & & & & & & \# \\ \wedge & . & . & \wedge & & \# & \# & \# & \# & \# \\ = & T & T & = & & & & & & ; \\ & & & & & \# & \# & \# & \# & \# & \# & \# & \# \\ & \# & \# & & & \# & \# & & & & & \\ M & M & & & & M & M & & & & & \end{array}$$



# 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?
```

# 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)
```

# 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?

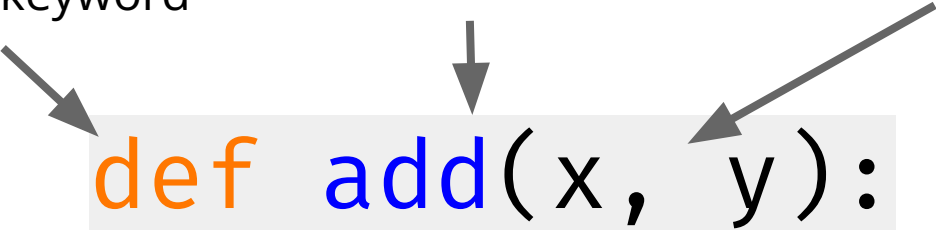
```
>>> print(z)
1
```

# Recap: A function signature

**definition**

the **def** keyword      function **name**      function **arguments**

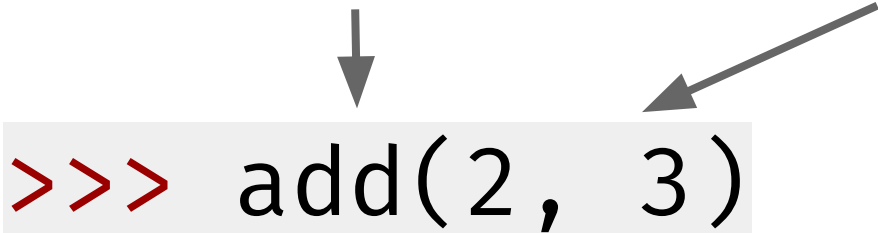
```
def add(x, y):
```

A diagram showing the components of a function definition. The text 'def add(x, y):' is highlighted in a light gray box. Three arrows point to it: one from the left labeled 'definition', one from above pointing to 'def' labeled 'the def keyword', one from above pointing to 'add' labeled 'function name', and one from above pointing to '(x, y):' labeled 'function arguments'.

**callsite**

function **name**      function **arguments**

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

A diagram showing the components of a function call. The text '>>> add(2, 3)' is highlighted in a light gray box. Two arrows point to it: one from above pointing to 'add' labeled 'function name', and one from above pointing to '(2, 3)' labeled 'function arguments'.



# 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
```

# 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`

`3 + 2 == 5`

`5 != 5`

`"Dog" == "dog"`

`"D" in "Dog"`

`"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

<code>5 &lt; 10</code>	<b>True</b>	<code>"Dog" == "dog"</code>
<code>3 + 2 == 5</code>		<code>"D" in "Dog"</code>
<code>5 != 5</code>		<code>"Q" not in "Cat"</code>



# 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**

3 + 2 == 5      **True**

5 != 5

"Dog" == "dog"

"D" in "Dog"

"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

<code>5 &lt; 10</code>	<code>True</code>	<code>"Dog" == "dog"</code>
<code>3 + 2 == 5</code>	<code>True</code>	<code>"D" in "Dog"</code>
<code>5 != 5</code>	<code>False</code>	<code>"Q" not in "Cat"</code>

# 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

<code>5 &lt; 10</code>	<code>True</code>	<code>"Dog" == "dog"</code>	<code>False</code>
<code>3 + 2 == 5</code>	<code>True</code>	<code>"D" in "Dog"</code>	
<code>5 != 5</code>	<code>False</code>	<code>"Q" not in "Cat"</code>	

# 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

<code>5 &lt; 10</code>	<code>True</code>	<code>"Dog" == "dog"</code>	<code>False</code>
<code>3 + 2 == 5</code>	<code>True</code>	<code>"D" in "Dog"</code>	<code>True</code>
<code>5 != 5</code>	<code>False</code>	<code>"Q" not in "Cat"</code>	

# 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**!

```
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")
```

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**

```
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

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

```
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

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

```
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!

# 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
```

# 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!

# Project Time!

You now know all about **if**!

**See **if** you can do the next Part**

The tutors will be around to help!

# While Loops

# Loops



We know how to do things on repeat!

Sometimes we want to do some code on repeat!

# Introducing ... while loops!

## What do you think this does?

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

# Introducing ... while loops!

## 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...

# 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

Set the  
variable

# Introducing ... while loops!

## One step at a time!

0 is less  
than 3!

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

MY VARIABLES

i = 0

# Introducing ... while loops!

## One step at a time!

Print !

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

i is 0

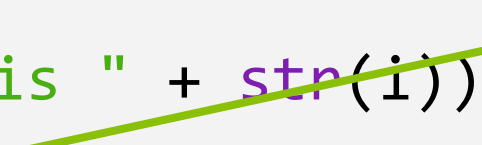
MY VARIABLES

i = 0

# 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

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
```

### MY VARIABLES

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

# 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
```

MY VARIABLES

~~i = 0~~  
i = 1

```
i is 0
```

# Introducing ... while loops!

## One step at a time!

Print!

```
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
```



# 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
```

UPDATE  
TIME!

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

# 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
```

### MY VARIABLES

```
i = 0
i = 1
i = 2
```

# Introducing ... while loops!

## One step at a time!

2 is less  
than 3!

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

### MY VARIABLES

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

i is 0

i is 1

# Introducing ... while loops!

## One step at a time!

Print!

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

```
i is 0
i is 1
i is 2
```


### MY VARIABLES

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

UPDATE  
TIME !

```
i is 0
i is 1
i is 2
```

# 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 is 2
```

### MY VARIABLES

```
i = 0
i = 1
i = 2
i = 3
```

# Introducing ... while loops!

## One step at a time!

3 IS NOT  
less than  
3!

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

MY VARIABLES

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

We are  
are done  
with this  
loop!

```
i is 0
i is 1
i is 2
```

# Introducing ... while loops!

Initialise the loop variable

Loop condition

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

Code to repeat

Update the loop variable



# What happens when.....

What happens if we forget to update the loop variable?

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

# What happens when.....

## What happens if we forget to update the loop variable?

```
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
i	i	o

# Give me a break!

But what if I wanna get out of a loop early?  
That's when we use the **break** keyword!

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

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

    number = int(number)
```

# Continuing on

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

```
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!

# For Loops

# Looping through lists!

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

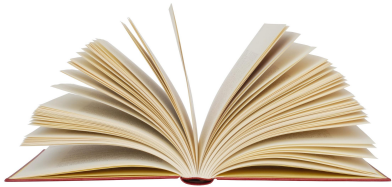
```
words = ['This', 'is', 'a', 'sentence']  
  
print(words[0])  
print(words[1])  
print(words[2])  
print(words[3])
```

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?

# 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!

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



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

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



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

# How does it work??

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





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

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

# How does it work??

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



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

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

dog is now 'apple'!

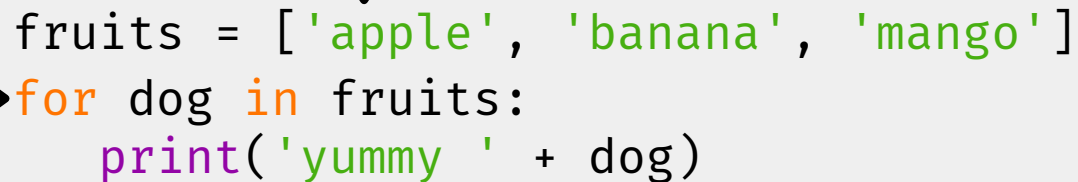
```
print('yummy ' + dog)
```

>>> Yummy apple



# How does it work??

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



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

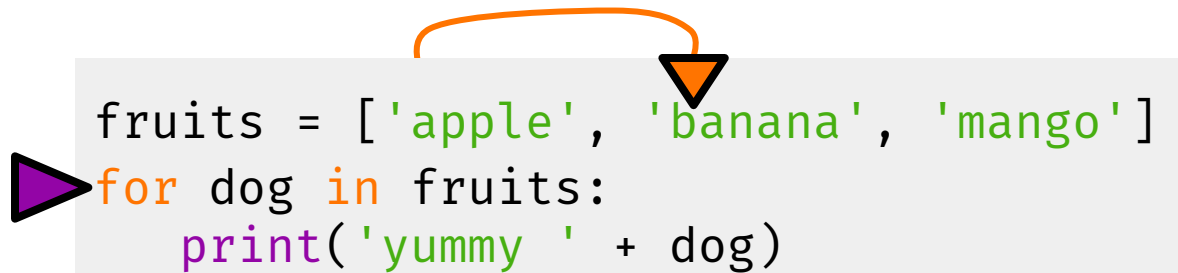
```
>>> Yummy apple
```

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!***



# How does it work??

Everything in the list gets to have a turn at being the dog 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 dog to to the **next** thing in the list!  
dog is now 'banana'!

# How does it work??

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

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

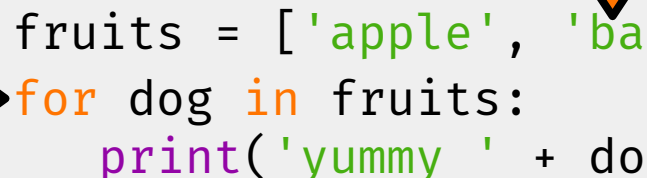
```
>>> Yummy apple  
>>> Yummy banana
```

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 dog to to the **next** thing in the list!  
dog is now 'banana'!  
print('yummy ' + dog)

# How does it work??

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



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

```
>>> Yummy apple
```

```
>>> Yummy banana
```

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 dog to to the **next** thing in the list!

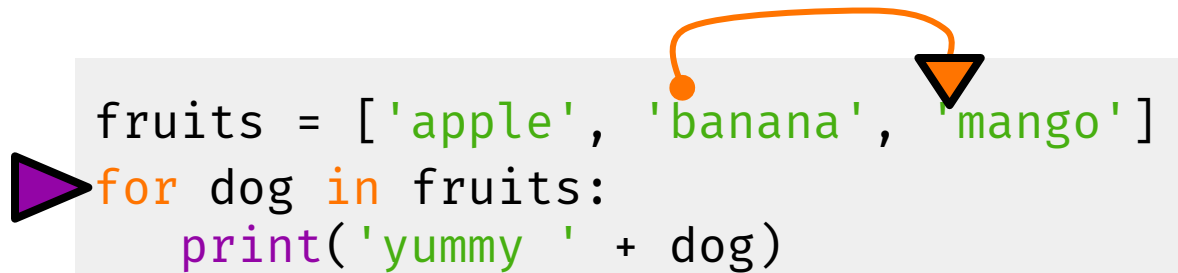
dog is now 'banana'!

```
print('yummy ' + dog)
```

*Out of body, back to the top!*

# How does it work??

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



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

```
>>> Yummy apple
```

```
>>> Yummy banana
```

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 dog 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 dog to to the **next** thing in the list!

dog is now 'mango'!

# How does it work??

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

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

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

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 dog 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 dog to to the **next** thing in the list!  
dog is now 'mango'!  
print('yummy ' + dog)

# How does it work??

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

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

>>> Yummy apple

>>> Yummy banana

>>> Yummy mango



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 dog 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 dog 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!*



# 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!

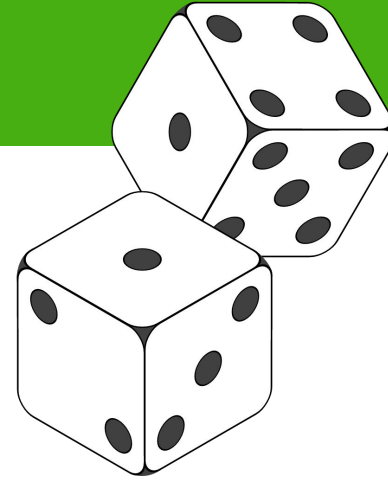


# That's so random!

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



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



We want the computer to be random sometimes!



# 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)
```



# Using the random module

**You can also assign your random choice to a variable**

```
>>> import random
>>> shopping_list = ["Bread", "Chocolate", "Ice Cream",
                    "Pizza"]
>>> 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 the next Part**

The tutors will be around to help!