Welcome to the Labs

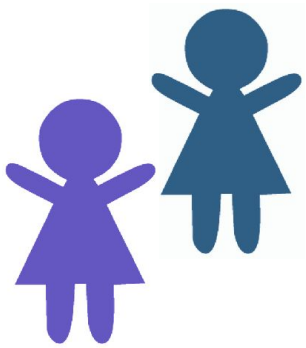
Cryptography!

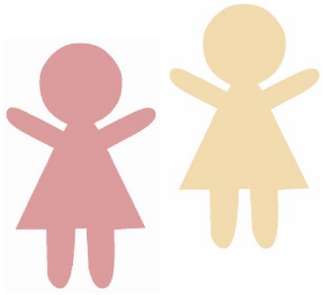
Who are the tutors?

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

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!

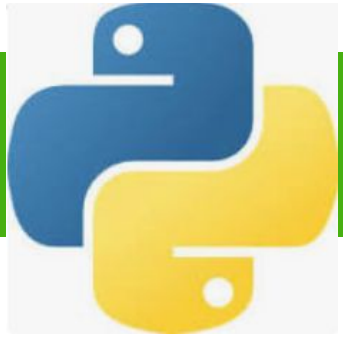
Today's project!

Cryptography!

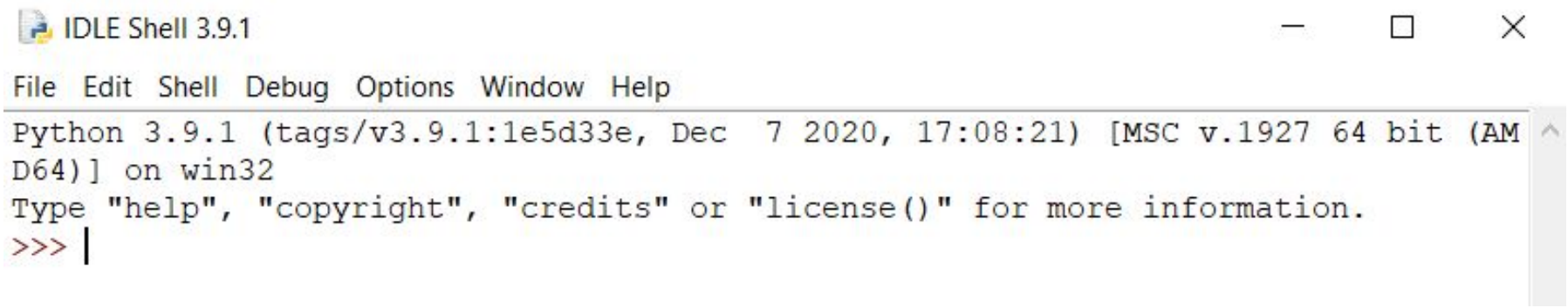
What is Cryptography?

- Dictionary definition = the art of writing or solving codes
- Secure communication that prevents others from reading private messages
- Applications
  - Online Shopping
  - Smart cards
  - Digital currencies
  - Computer passwords

Getting Python IDLE Ready



- IDLE is where we write and run our Python program
- To open the IDLE app on your computer
  - Click the Windows button bottom left 
  - Type IDLE in the Search bar displayed
  - Open the app



Using the workbook!

The workbooks will help you put your project together!

Each **Part** of the workbook is made of tasks & hints!

Tasks - The parts of your project

Follow the tasks **in order** to make the project!

Hints - Helpers for your tasks!

Stuck on a task, we might have given you a hint to help you **figure it out**!

The hints have **unrelated** examples, or tips. **Don't copy and paste** in the code, you'll end up with something **CRAZY**!

Task 6.2: Add a blah to your code!

This has instructions on how to do a part of the project

1. **Start by doing this part**
2. **Then you can do this part**

Task 6.1: Make the thing do blah!

Make your project do blah ….

Hint

A clue, an example or some extra information to help you **figure out** the answer.

```
print('This example is not part of the project' )
```

Using the workbook!

The workbooks will help you put your project together!

Checklist before you move on. Bonuses. Lectures.

Checklist - Am I done yet?

Make sure you can tick off every box in this section before you go to the next Part.

Bonus Activities
Stuck waiting at a lecture marker?
Try a purple bonus. They add extra functionality to your project along the way.

Lecture Markers
This tells you the Lecture where you'll find out how to do things for this section.

☑ **CHECKPOINT** ☑

If you can tick all of these off you're ready to move the next part!
☐ Your program does blah
☐ Your program does blob

★ **BONUS 4.3: Do some extra!**

Something to try if you have spare time before the next lecture!

Intro to Caesar Ciphers

Let’s get encrypting!

What is a cipher?

A cipher is a way to write a message so that no one else can read it!

Unless they know the secret key!

Examples of ciphers

If you've ever made up your own secret language or made notes to your friends so that other people can't read them, you've made a cipher!

For example:

gnidoc evol i

Can you figure out what this says?

Examples of ciphers

If you've ever made up your own secret language or made notes to your friends so that other people can't read them, you've made a cipher!

For example:

gnidoc evol i

Can you figure out what this says?

It says **I love coding** backwards!

Caesar Cipher

So what's a Caesar Cipher?

It's a cypher that Julius Caesar used in ancient Rome to send secret messages to his armies!

Let's learn how it works!

Cipher Wheels

You each have a cipher wheel that looks like this:

You can spin the inside set of letters around and make them line up with different letters

Shifting letters

A Caesar Cipher works by shifting letters in the alphabet so that they line up with new letters.

For example if we were to shift everything by 3 it would look like this:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

Try turning your purple wheel 3 letters **anti-clockwise** so that you have your letters lining up like this!

Making the secret message

Now, let's write a secret message!

I love coding

For our Caesar cipher we take each letter and replace it with the letter that has been shifted

So, let's start with the letter i

What new letter should we use to replace it?

Making the secret message

Now, let's write a secret message!

I love coding

For our Caesar cipher we take each letter and replace it with the letter that has been shifted

So, let's start with the letter i

What new letter should we use to replace it?

The letter L

Writing the whole message!

I love coding

l	Is replaced with	o
o	Is replaced with	r
v	Is replaced with	y
e	Is replaced with	h
c	Is replaced with	f
o	Is replaced with	r
d	Is replaced with	g
i	Is replaced with	l
n	Is replaced with	q
g	Is replaced with	i

Secret Message

So our secret encrypted message is

L oryh frglqj

That’s a lot harder to figure out than it just being backwards!

Encrypt your own name!

Using a key of 5 (so A=F) (Jessica = Ojxxnhf)

Decrypting

Writing secret messages isn't any fun if you can't figure out what they say!

Luckily you can also use your cipher wheel to *decrypt* a secret message.

How do you think we can do that?

What information do we need to know in order to decrypt a secret message?

It’s the key!

To decrypt a secret message **we need to know** the amount that we shifted the wheel when we encrypted it. That number is called **the key**!

Once we know the key we can just turn our wheel and read the wheel from the inside out!

*Find the letter on the **inside** wheel and replace it with it’s matching letter on the **outside** wheel*

It’s the key!

To decrypt a secret message **we need to know** the amount that we shifted the wheel when we encrypted it. That number is called **the key**!

Once we know the key we can just turn our wheel and read the wheel from the inside out!

Let’s check that it works with: L oryh frglqj

Remember that the key is 3!

Let’s check it works!

l	Is replaced with	i
o	Is replaced with	
r	Is replaced with	
y	Is replaced with	
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Let’s check it works!

l	Is replaced with	i
o	Is replaced with	l
r	Is replaced with	
y	Is replaced with	
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Let’s check it works!

l	Is replaced with	i
o	Is replaced with	l
r	Is replaced with	o
y	Is replaced with	
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Let’s check it works!

l	Is replaced with	i
o	Is replaced with	l
r	Is replaced with	o
y	Is replaced with	v
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Let’s check it works!

l	Is replaced with	i
o	Is replaced with	l
r	Is replaced with	o
y	Is replaced with	v
h	Is replaced with	e
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Let's check it works!

l	Is replaced with	**i**
o	Is replaced with	**l**
r	Is replaced with	**o**
y	Is replaced with	**v**
h	Is replaced with	**e**
f	Is replaced with	**c**
r	Is replaced with	**o**
g	Is replaced with	**d**
l	Is replaced with	**i**
q	Is replaced with	**n**
j	Is replaced with	**g**

Another way to decrypt

- Another way to decrypt a message is to change the key value to become the negative of the encryption key value
- We will use this method in our code
- This is because to decrypt a message we need to shift the alphabet the opposite way.
- A negative key value means you turn your inner purple wheel to the right (clockwise)

Turn it back!

l	Is replaced with	
o	Is replaced with	
r	Is replaced with	
y	Is replaced with	
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Turn it back!

l	Is replaced with	i
o	Is replaced with	
r	Is replaced with	
y	Is replaced with	
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Turn it back!

l	Is replaced with	i
o	Is replaced with	l
r	Is replaced with	
y	Is replaced with	
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Turn it back!

l	Is replaced with	i
o	Is replaced with	l
r	Is replaced with	o
y	Is replaced with	
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Turn it back!

l	Is replaced with	i
o	Is replaced with	l
r	Is replaced with	o
y	Is replaced with	v
h	Is replaced with	
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Turn it back!

l	Is replaced with	**i**
o	Is replaced with	**l**
r	Is replaced with	**o**
y	Is replaced with	**v**
h	Is replaced with	**e**
f	Is replaced with	
r	Is replaced with	
g	Is replaced with	
l	Is replaced with	
q	Is replaced with	
j	Is replaced with	

Turn it back!

l	Is replaced with	**i**
o	Is replaced with	**l**
r	Is replaced with	**o**
y	Is replaced with	**v**
h	Is replaced with	**e**
f	Is replaced with	**c**
r	Is replaced with	**o**
g	Is replaced with	**d**
l	Is replaced with	**i**
q	Is replaced with	**n**
j	Is replaced with	**g**

Fun fact!

Turning the wheel **backwards**

is the same as

reading your wheel **inside out**!

Your Turn!

Try doing Part 0 of Workbook 1 using your Caesar Cipher wheels!

Your tutors are here to help you if you get stuck

Intro to Python

Let’s get coding!

Write some code!

Type this into the window

Then press enter!

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

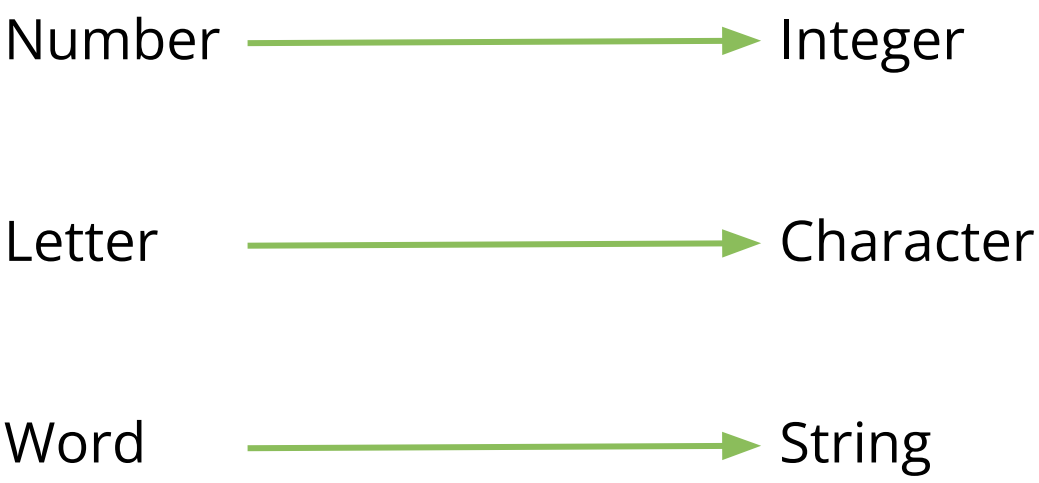
Did it print:

```
hello world
```

???

Data types

In programming, we have special names for the following:



Let’s look at some examples

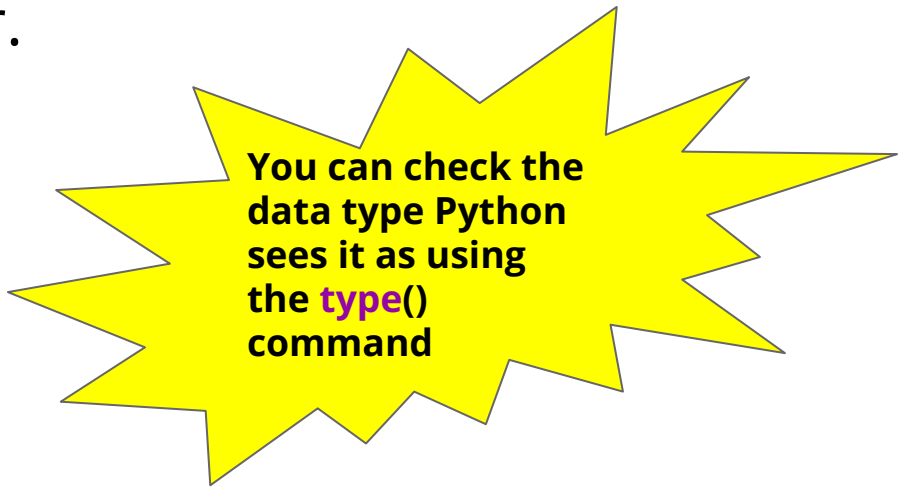
Characters - not always letters

What do all of these have in common?

“A” ‘6’ “f” ‘$’

Anything that only takes up only one space and is surrounded by ‘single’ or “double” quotes, is considered a **character** by the computer.

```
type("5") = char
type(5) = int
```



Strings

Strings are a group of more than one **character** put together and surrounded with `"quotes"`

All of these are strings:

```
"Dog"
        "my name is"
                  "123 hahaha"
                              "$%#^&@(){}[]"
```

A calculator for words!?

What do you think these bits of code do?
Try them and see!

```
>>> "cat" + "dog"
```

```
>>> "tortoise" * 3
```

Calculator for... words!?

What do you think these bits of code do?
Try them and see!

```
>>> "cat" + "dog"
catdog

>>> "tortoise" * 3
```

Calculator for... words!?

What do you think these bits of code do?
Try them and see!

```
>>> "cat" + "dog"
catdog

>>> "tortoise" * 3
tortoisetortoisetortoise
```

Calculator for words and number?

If we can do calculations with numbers, and calculations with words, can we do calculations with words *and* numbers?

Try writing this!

```
>>> 1 + "1"
```

```
>>> "100" * 2
```

How do we deal with this problem? See next slide!

Type casting

We tell the computer exactly what type we want to use!

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"
```

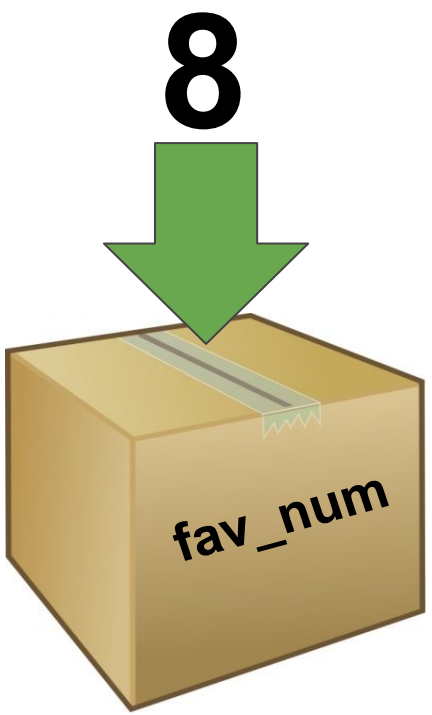
No Storing is Boring!

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

Computers remember things in **"variables"**

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

Let's make our favourite number 8 today!



Variables

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


```
fav_num - 6
```

=> 2

```
fav_num + 21
```

=> 29

```
fav_num * 2
```

=> 16

```
fav_num / 2
```

=> 4

Variables

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


```
fav_num - 6
```

=> 2

```
fav_num + 21
```

=> 29

```
fav_num * 2
```

=> 16

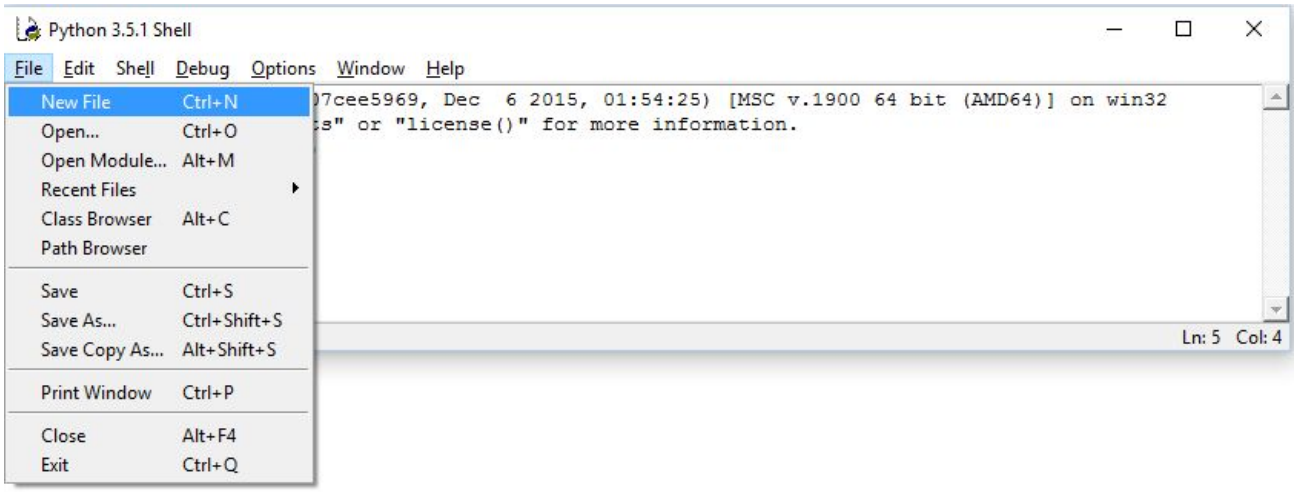
```
f                2
```

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

We'll come back to this later!

Coding in a file!

Code in a file is code we can run multiple times! Make a reusable “hello world”!



1. Make a new file called hello.py, like the picture
2. Put your `print('hello world')` code in it
3. Run your file using the F5 key

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!")
```

Try it!

1. Add a comment to your hello.py file
2. Run your code to make sure it doesn't do anything extra!

Asking a question!

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

Try out this code to get the computer to ask you a question!

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

What do you think happens?

Asking a question!

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

What do you think happens?

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

Asking a question!

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
```

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

Asking a question!

How would we ask somebody for their favourite type of cake?

How would we print their answer?

Give it a try on your own computer first!

```
What cake do you like? chocolate
chocolate cake for you!
```

Asking a question!

How would we ask somebody for their favourite type of cake?

How would we print their answer?

Give it a try on your own computer first!

```
flavour = input("What cake do you like? ")
```

```
What cake do you like? chocolate
chocolate cake for you!
```

Asking a question!

How would we ask somebody for their favourite type of cake?

How would we print their answer?

Give it a try on your own computer first!

```
flavour = input("What cake do you like? ")
print(flavour + "cake for you"!)
```

```
What cake do you like? chocolate
chocolate cake for you!
```

Your Turn!

Try doing Parts 1 & 2 of Workbook 1

To start IDLE (Step 1.1) click on Windows button bottom left and search for IDLE

Your tutors are here to help if you get stuck

Strings, Ints & Modulo

Cutting Strings

We can get individual letters from a string using indexes.

```
>>> yum = "chocolate"
>>> yum[0]

>>> yum[5]

>>> yum[-1]

>>> yum[500]
```

Cutting Strings

We can get individual letters from a string using indexes.

```
>>> yum = "chocolate"
>>> yum[0]
'c'
>>> yum[5]


>>> yum[-1]


>>> yum[500]
```

Computers start counting from 0, not 1!

Cutting Strings

We can get individual letters from a string using indexes.

```
>>> yum = "chocolate"
>>> yum[0]
'c'
>>> yum[5]
'l'
>>> yum[-1]

>>> yum[500]
```

Computers start counting from 0, not 1!

Cutting Strings

We can get individual letters from a string using indexes.

```
>>> yum = "chocolate"
>>> yum[0]
'c'
>>> yum[5]
'l'
>>> yum[-1]
'e'
>>> yum[500]
```

Computers start counting from 0, not 1!

Cutting Strings

We can get individual letters from a string using indexes.

```
>>> yum = "chocolate"
>>> yum[0]
'c'
>>> yum[5]
'l'
>>> yum[-1]
'e'
>>> yum[500]
IndexError: string index out of range
```

Computers start counting from 0, not 1!

Maths on Indexes!

We can use any sort of int as an index, including the result of an expression or maths equation!

```
>>> yum = "chocolate"
>>> len(yum)


>>> yum[9 - 1]


>>> yum[10 % len(yum)]
```

Maths on Indexes!

We can use any sort of int as an index, including the result of an expression or maths equation!

```
>>> yum = "chocolate"
>>> len(yum)
9
>>> yum[9 - 1]

>>> yum[10 % len(yum)]
```

Maths on Indexes!

We can use any sort of int as an index, including the result of an expression or maths equation!

```
>>> yum = "chocolate"
>>> len(yum)
9
>>> yum[9 - 1]
'e'
>>> yum[10 % len(yum)]
```

Maths on Indexes!

We can use any sort of int as an index, including the result of an expression or maths equation!

```
>>> yum = "chocolate"
>>> len(yum)
9
>>> yum[9 - 1]
'e'
>>> yum[10 % len(yum)]
'h'
```

Notice how we used the remainder from dividing by the length to count again from the beginning of the word?

Modulo %

Modulo **%** is a maths operation

% gives the **remainder** of a division

You'll need to use it in your code!

- 10 % 8 =
- 20 % 7 =
- 5 % 6 =

Modulo %

Modulo **%** is a maths operation

% gives the **remainder** of a division

You'll need to use it in your code!

- 10 % 8 = 2 (10 divided by 8 is 1 with remainder 2)
- 20 % 7 =
- 5 % 6 =

Modulo %

Modulo **%** is a maths operation

% gives the **remainder** of a division

You'll need to use it in your code!

- 10 % 8 = 2 (10 divided by 8 is 1 with remainder 2)
- 20 % 7 = 6 (20 divided by 7 is 2 with remainder 6)
- 5 % 6 =

Modulo %

Modulo **%** is a maths operation

% gives the **remainder** of a division

You'll need to use it in your code!

- 10 % 8 = 2 (10 divided by 8 is 1 with remainder 2)
- 20 % 7 = 6 (20 divided by 7 is 2 with remainder 6)
- 5 % 6 = 5 (5 divided by 6 is 0 with remainder 5)

Project time!

You now know all about strings, ints and modulo!

Let's put what we learnt into our project

Try to do Part 3

The tutors will be around to help!

For Loops

Looping through a string

Strings are a group of characters!

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

What's going to happen?

```
>>> c
>>> a
>>> t
```

How does it work??

Every character in the string gets to have a turn at being the i variable

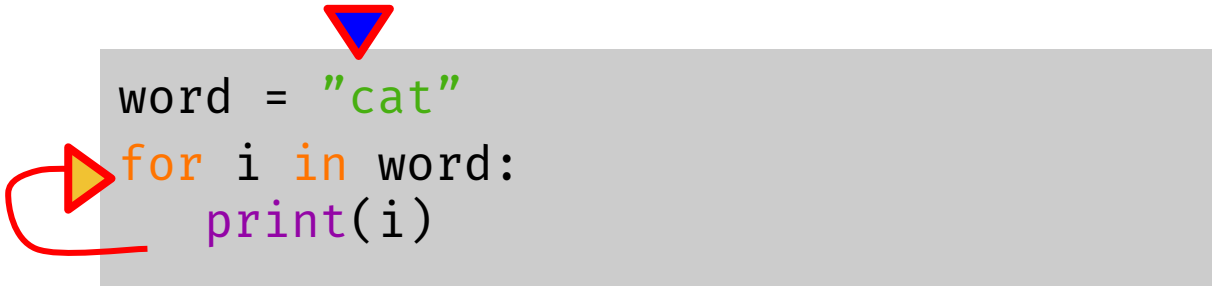
```
word = "cat"
for i in word:
    print(i)
```

```
>>> c
```

Let's set i to to the first character in the string!

```
i is now 'c'
print(i)
```

How does it work??




```
>>> c
```

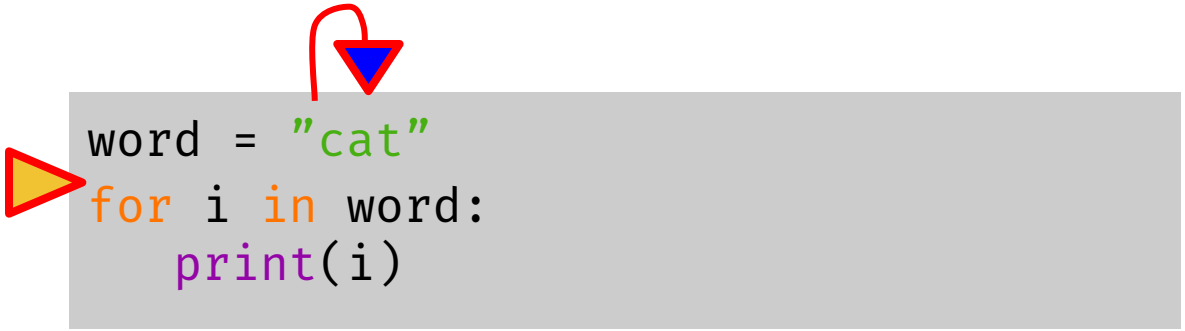
Now we're at the end of the loop body, so go back to the start

Let's set i to to the first thing in the string!

```
i is now 'c'
print(i)
```

How does it work??

Every character in the string gets to have a turn at being the i variable, so we now set i to the next character



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

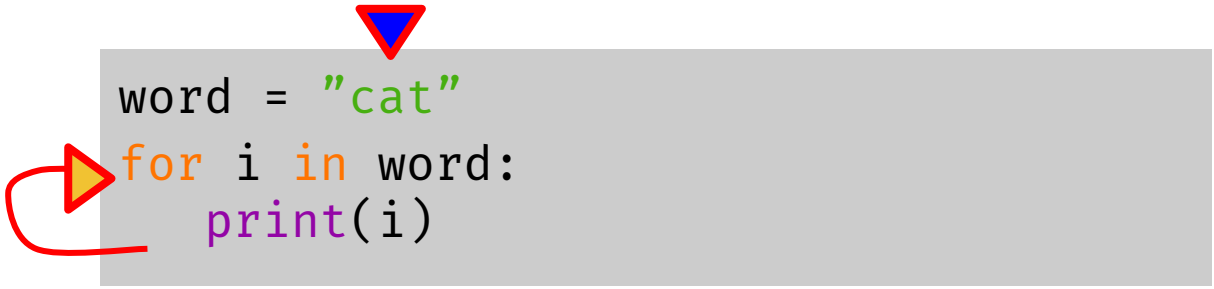
```
>>> c
>>> a
```

Let's set i to to the next character in the string!

`i` is now `'a'`!

`print(i)`

How does it work??



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

```
>>> c
>>> a
```

Now we're at the end of the loop body AGAIN, so go back to the start

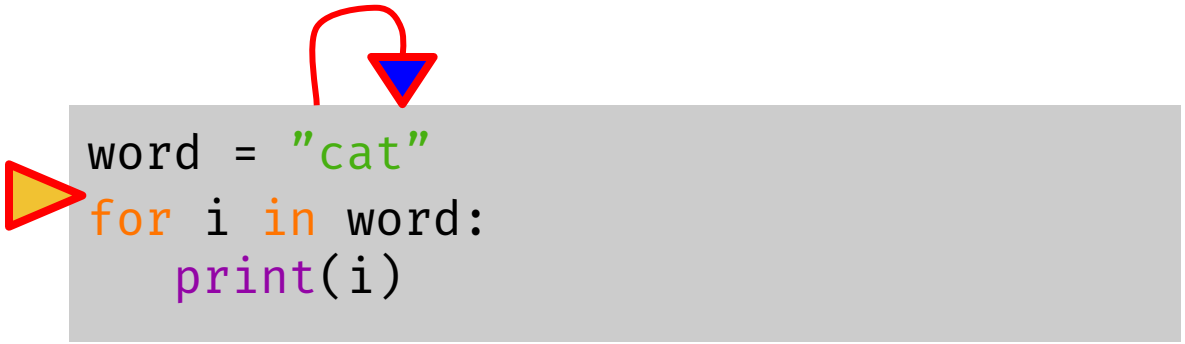
Let's set i to to the next thing in the string!

`i` is now `'a'`!

```
print(i)
```

How does it work??

Every character in the string gets to have a turn at being the i variable, so we now set i to the next character



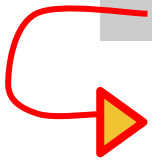
```
word = "cat"
for i in word:
    print(i)
```

```
>>> c
>>> a
>>> t
```

Let's set i to to the next thing in the string!
i is now 't'!
print(i)

How does it work??

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



```
>>> c
>>> a
>>> t
```

Now we're at the end of the loop body AGAIN but we have been through all the characters in the string so we exit the for loop

Let's set i to to the next thing in the string!
i is now 't'!
print(i)

Project Time!

Now you know how to use a for loop!

Try to do Part 4

...if you are up for it!

The tutors will be around to help!

If Statements

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

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

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
```

Booleans (True and False)

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

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

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

Booleans (True and False)

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

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

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

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"]
>>> "banana" in animals
>>> "cat" in animals
```

Booleans (True and False)

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

```
"A" in "AEIOU"
```
`True`

```
"Z" in "AEIOU"
```
`False`

```
"a" in "AEIOU"
```
`False`

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

```
>>> "cat" in animals
```

Booleans (True and False)

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

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

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

Conditions

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

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

What do you think happens?

```
>>>
```

Conditions

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

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

What do you think happens?

```
>>> 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

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

What do you think happens?

```
>>>
```

Conditions

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

What do you think 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?

Else statements

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

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

What happens?

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

Elif statements

elif
Means we can give specific instructions for other words

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

What happens?

Elif statements

elif
Means we can give specific instructions for other words

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

What happens?

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

Project Time!

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

See `if` you can do Part 5

The tutors will be around to help!

Intro to Vigenere Ciphers

Caesar Cipher

So now you know what a Caesar Cipher is, let's look at a more complicated cipher!

A Caesar Cipher uses just 1 key to encrypt and decrypt the message, a Vigenere cypher uses a whole word as the key!

The keyword

Let’s see how it uses a whole word by doing an example together!

Let’s use the keyword

pizza

Splitting it into keys

Now we take the keyword and we split it into a bunch of keys!

Each letter of the alphabet equals a different number (a=0, b=1, c=2 etc.)

Now we change our keyword into a bunch of different keys by replacing each letter with its number in the alphabet

p	**i**	**z**	**z**	**a**

Splitting it into keys

Now we take the keyword and we split it into a bunch of keys!

Each letter of the alphabet equals a different number (a=0, b=1, c=2 etc.)

Now we change our keyword into a bunch of different keys by replacing each letter with its number in the alphabet

p	**i**	**z**	**z**	**a**
15				

Splitting it into keys

Now we take the keyword and we split it into a bunch of keys!

Each letter of the alphabet equals a different number (a=0, b=1, c=2 etc.)

Now we change our keyword into a bunch of different keys by replacing each letter with its number in the alphabet

p	**i**	**z**	**z**	**a**
15	8			

Splitting it into keys

Now we take the keyword and we split it into a bunch of keys!

Each letter of the alphabet equals a different number (a=0, b=1, c=2 etc.)

Now we change our keyword into a bunch of different keys by replacing each letter with its number in the alphabet

p	**i**	**z**	**z**	**a**
15	8	25		

Splitting it into keys

Now we take the keyword and we split it into a bunch of keys!

Each letter of the alphabet equals a different number (a=0, b=1, c=2 etc.)

Now we change our keyword into a bunch of different keys by replacing each letter with its number in the alphabet

p	**i**	**z**	**z**	**a**
15	8	25	25	

Splitting it into keys

Now we take the keyword and we split it into a bunch of keys!

Each letter of the alphabet equals a different number (a=0, b=1, c=2 etc.)

Now we change our keyword into a bunch of different keys by replacing each letter with its number in the alphabet

p	**i**	**z**	**z**	**a**
15	8	25	25	0

Loop the word

Let’s try encrypting a message with our keyword using a Vigenere cipher now!

I love coding

Each letter in our message will line up with a letter in our keyword and we will keep looping the keyword like this:

i	**l**	**o**	**v**	**e**	**c**	**o**	**d**	**i**	**n**	**g**
p	**i**	**z**	**z**	**a**	**p**	**i**	**z**	**z**	**a**	**p**

Using the numbers

Now we replace each letter of our keyword with the numbers that we worked out before:

i	l	o	v	e	c	o	d	i	n	g
15	8	25	25	0	15	8	25	25	0	15

Next we just shift each letter in our message like we do with a Caesar Cipher but with the key that it lines up with.

What key does the letter C use?

Using the numbers

Now we replace each letter of our keyword with the numbers that we worked out before:

i	l	o	v	e	c	o	d	i	n	g
15	8	25	25	0	15	8	25	25	0	15

Next we just shift each letter in our message like we do with a Caesar Cipher but with the key that it lines up with.

What key does the letter C use? 15

Making the secret message

i	Using key: **15**	Is replaced with	**x**
l	Using key: **8**	Is replaced with	
o	Using key: **25**	Is replaced with	
v	Using key: **25**	Is replaced with	
e	Using key: **0**	Is replaced with	
c	Using key: **15**	Is replaced with	
o	Using key: **8**	Is replaced with	
d	Using key: **25**	Is replaced with	
i	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
g	Using key: **15**	Is replaced with	

Making the secret message

i	Using key: **15**	Is replaced with	**x**
l	Using key: **8**	Is replaced with	**t**
o	Using key: **25**	Is replaced with	
v	Using key: **25**	Is replaced with	
e	Using key: **0**	Is replaced with	
c	Using key: **15**	Is replaced with	
o	Using key: **8**	Is replaced with	
d	Using key: **25**	Is replaced with	
i	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
g	Using key: **15**	Is replaced with	

Making the secret message

i	Using key: **15**	Is replaced with	**x**
l	Using key: **8**	Is replaced with	**t**
o	Using key: **25**	Is replaced with	**n**
v	Using key: **25**	Is replaced with	
e	Using key: **0**	Is replaced with	
c	Using key: **15**	Is replaced with	
o	Using key: **8**	Is replaced with	
d	Using key: **25**	Is replaced with	
i	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
g	Using key: **15**	Is replaced with	

Making the secret message

i	Using key: **15**	Is replaced with	**x**
l	Using key: **8**	Is replaced with	**t**
o	Using key: **25**	Is replaced with	**n**
v	Using key: **25**	Is replaced with	**u**
e	Using key: **0**	Is replaced with	
c	Using key: **15**	Is replaced with	
o	Using key: **8**	Is replaced with	
d	Using key: **25**	Is replaced with	
i	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
g	Using key: **15**	Is replaced with	

Making the secret message

i	Using key: **15**	Is replaced with	**x**
l	Using key: **8**	Is replaced with	**t**
o	Using key: **25**	Is replaced with	**n**
v	Using key: **25**	Is replaced with	**u**
e	Using key: **0**	Is replaced with	**e**
c	Using key: **15**	Is replaced with	
o	Using key: **8**	Is replaced with	
d	Using key: **25**	Is replaced with	
i	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
g	Using key: **15**	Is replaced with	

Making the secret message

i	Using key: **15**	Is replaced with	**x**
l	Using key: **8**	Is replaced with	**t**
o	Using key: **25**	Is replaced with	**n**
v	Using key: **25**	Is replaced with	**u**
e	Using key: **0**	Is replaced with	**e**
c	Using key: **15**	Is replaced with	**r**
o	Using key: **8**	Is replaced with	**w**
d	Using key: **25**	Is replaced with	**c**
i	Using key: **25**	Is replaced with	**h**
n	Using key: **0**	Is replaced with	**n**
g	Using key: **15**	Is replaced with	**v**

Secret Message

So our secret encrypted message is **x tnue rwchnv**

To decrypt it you do the same thing with each letter and key that you did to decrypt in the Caesar cipher

- change the key value to become the negative of the encryption key value
- turn the wheel backwards (clockwise) to undo the encryption and get the secret message
- this shifts the alphabet the opposite way to what we did to encrypt the message

Turn it back!

x	Using key: **15**	Is replaced with	i
t	Using key: **8**	Is replaced with	
n	Using key: **25**	Is replaced with	
u	Using key: **25**	Is replaced with	
e	Using key: **0**	Is replaced with	
r	Using key: **15**	Is replaced with	
w	Using key: **8**	Is replaced with	
c	Using key: **25**	Is replaced with	
h	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
v	Using key: **15**	Is replaced with	

Turn it back!

x	Using key: **15**	Is replaced with	i
t	Using key: **8**	Is replaced with	l
n	Using key: **25**	Is replaced with	
u	Using key: **25**	Is replaced with	
e	Using key: **0**	Is replaced with	
r	Using key: **15**	Is replaced with	
w	Using key: **8**	Is replaced with	
c	Using key: **25**	Is replaced with	
h	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
v	Using key: **15**	Is replaced with	

Turn it back!

x	Using key: **15**	Is replaced with	i
t	Using key: **8**	Is replaced with	l
n	Using key: **25**	Is replaced with	o
u	Using key: **25**	Is replaced with	
e	Using key: **0**	Is replaced with	
r	Using key: **15**	Is replaced with	
w	Using key: **8**	Is replaced with	
c	Using key: **25**	Is replaced with	
h	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
v	Using key: **15**	Is replaced with	

Turn it back!

x	Using key: **15**	Is replaced with	i
t	Using key: **8**	Is replaced with	l
n	Using key: **25**	Is replaced with	o
u	Using key: **25**	Is replaced with	v
e	Using key: **0**	Is replaced with	e
r	Using key: **15**	Is replaced with	
w	Using key: **8**	Is replaced with	
c	Using key: **25**	Is replaced with	
h	Using key: **25**	Is replaced with	
n	Using key: **0**	Is replaced with	
v	Using key: **15**	Is replaced with	

Turn it back!

x	Using key: **15**	Is replaced with	i
t	Using key: **8**	Is replaced with	l
n	Using key: **25**	Is replaced with	o
u	Using key: **25**	Is replaced with	v
e	Using key: **0**	Is replaced with	e
r	Using key: **15**	Is replaced with	c
w	Using key: **8**	Is replaced with	o
c	Using key: **25**	Is replaced with	d
h	Using key: **25**	Is replaced with	i
n	Using key: **0**	Is replaced with	n
v	Using key: **15**	Is replaced with	g

Your Turn!

Now you try on your own!

Try doing Part 0 - Part 1 of the second workbook!

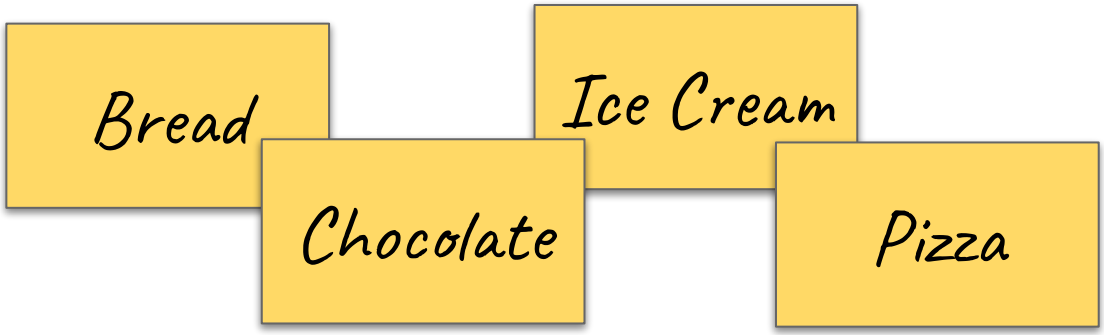
Your tutors are here to help you if you get stuck

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!

- 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 list!

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

You can put (almost) anything into a list

- You can have a list of integers

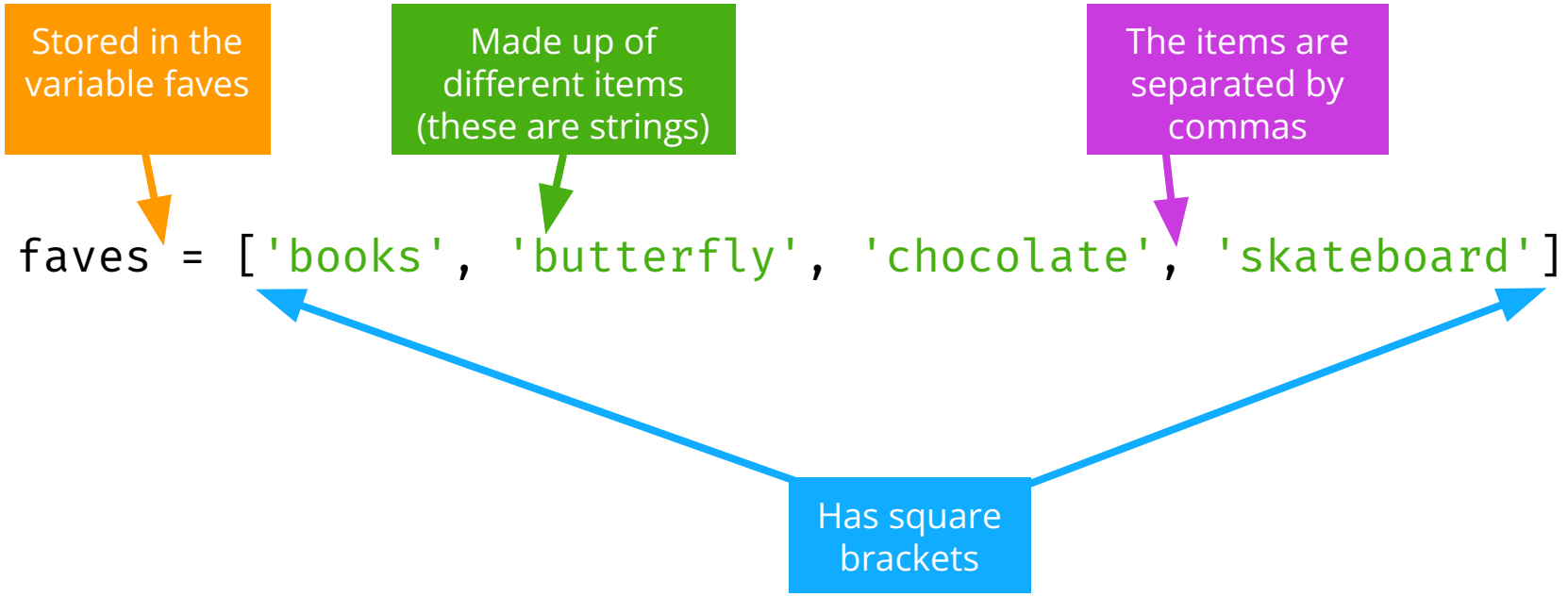
```
>>> primes = [1, 2, 3, 5, 11]
```

- You can have a list of strings

```
>>> numbers = ["one", "two", "three"]
```

- Every element of a list should be the same (eg integer, string). You should be able to treat every element of the list the same way.

List anatomy



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'
```

Going Negative

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

```
>>> faves = ['books', 'butterfly', 'chocolate',
'skateboard']
>>> faves[-1]
'skateboard'
```

What would `faves[-3]` return?

Going Negative

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

```
>>> faves = ['books', 'butterfly', 'chocolate',
'skateboard']
>>> faves[-1]
'skateboard'
```

What would `faves[-3]` return?

```
'butterfly'
```

Updating items!

We can also update things in a list:

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

Updating items!

We can also update things in a list:

```
>>> faves = ['books', 'butterfly',
'chocolate', 'skateboard']
>>> faves[1]
'butterfly'
```

Updating items!

We can also update things in a list:

```
>>> faves = ['books', 'butterfly',
'chocolate', 'skateboard']
>>> faves[1]
'butterfly'
>>> faves[1] = 'kittens'
>>> faves[1]
'kittens'
```

Updating items!

We can also update things in a list:

```
>>> faves = ['books', 'butterfly',
'chocolate', 'skateboard']
>>> faves[1]
'butterfly'
>>> faves[1] = 'kittens'
```

Removing items!

We can remove items from the list if they're no longer needed!

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

What if we decided that we didn't like butterflies anymore?

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

What does this list look like now?

Removing items!

We can remove items from the list if they're no longer needed!

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

What if we decided that we didn't like butterflies anymore?

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

What does this list look like now?

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

Adding items!

We can also add new items to the list!

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

What if we decided that we also liked programming?

```
>>> faves.append('programming')
```

What does this list look like now?

Adding items!

We can also add new items to the list!

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

What if we decided that we also liked programming?

```
>>> faves.append('programming')
```

What does this list look like now?

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

What can you do with a list?

- Define an empty list to add to in your code

```
>>> songs = []
```

- Loop through a list

```
>>> odd_numbers = [1, 3, 5, 7]
>>> for i in odd_numbers:
        print(i)
```

Project time!

You now know all about lists!

Let's put what we learnt into our project.

Try to do Part 2 of the second workbook!

The tutors will be around to help!

Functions!

Simpler, less repetition, easier to read code!

How functions fit together!

Functions are like factories!

Metal Worker

Your main factory!

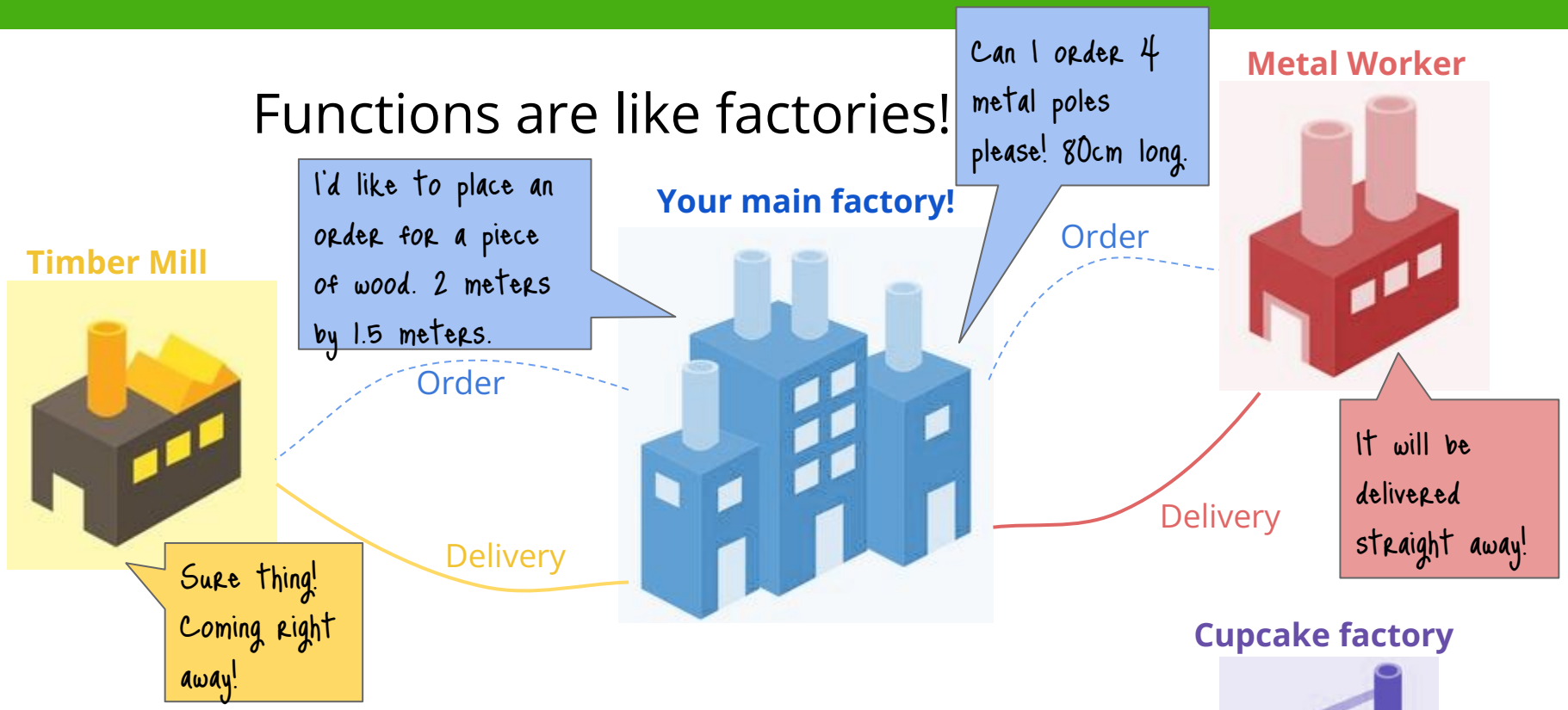
Timber Mill

Cupcake factory

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

How functions fit together!

Functions are like factories!



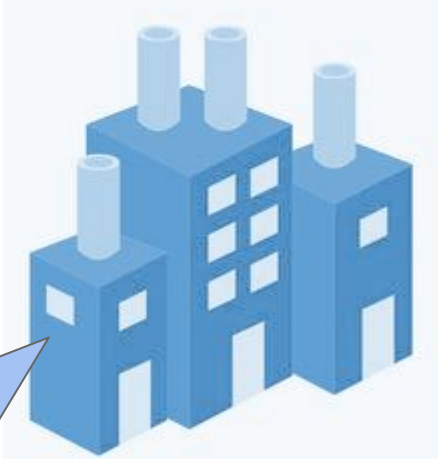
Asking other factories to do some work for you makes your main task simper. You can focus on the assembly!

How functions fit together!

Functions are like factories!

Metal Worker

Your main factory!



Timber Mill

Look at this beautiful table I made!

Outsourcing made it simple!

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=      ;
            #########
            # #   # #
            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
```

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.

```
>>> 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 declared 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

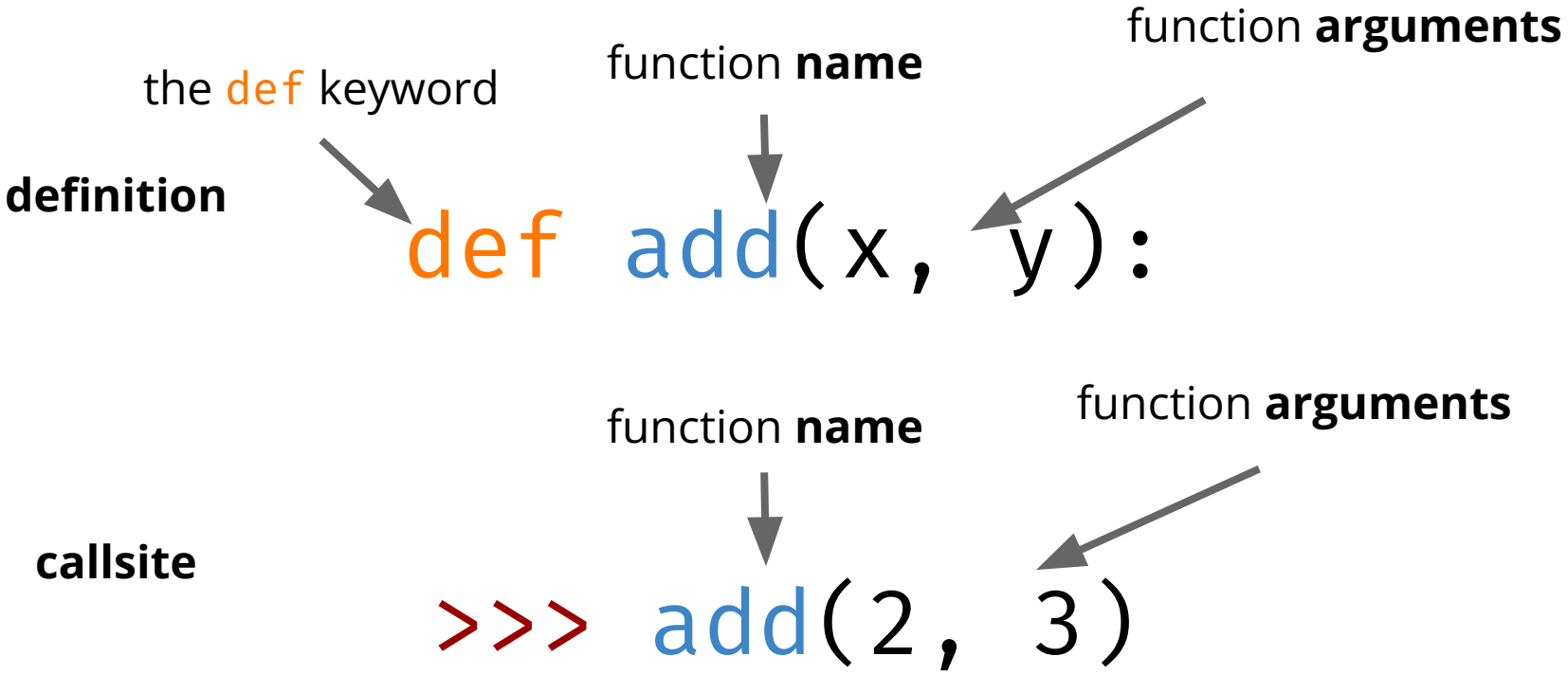
Changing a variable in a function **only changes it *inside* the function.**

```
>>> 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



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
>>> print(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)
>>> print(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
>>> print(sum)
4
```

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

Project time!

Now you know how to build function!

Now try to do Part 3 - Part 6 of the second workbook!

The tutors will be around to help!