Welcome to the labs!

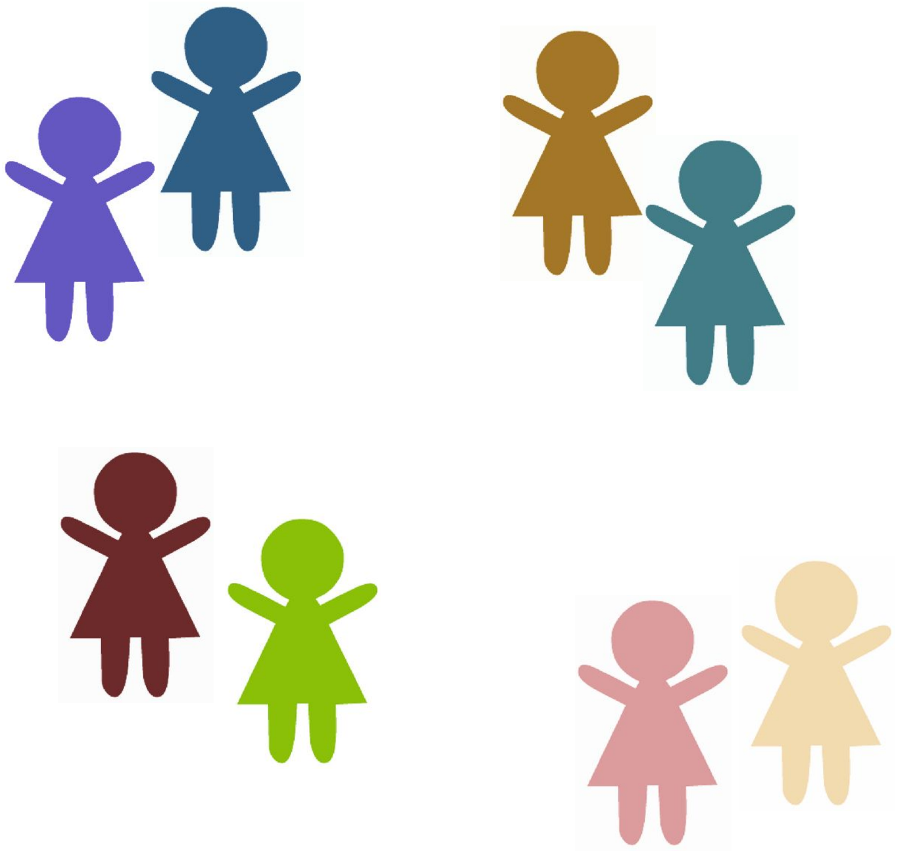
Secret Diary

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

bit.ly/gpn-perth-2020-1

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!

Secret Diary

Using the workbook!

The workbooks will help you put your project together!

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

Tasks - The parts of your project

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

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**

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 6.1: 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!

Check off before you move on from a **Part**! Do some bonuses while you wait!

Checklist - Am I done yet?

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

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

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

☑ **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!**

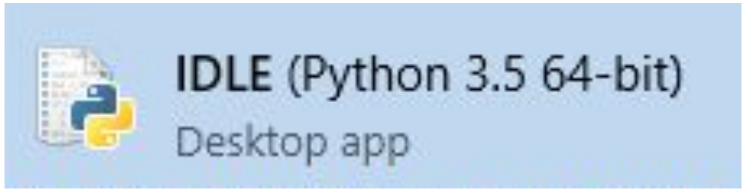
Something to try if you have spare time before the next lecture!

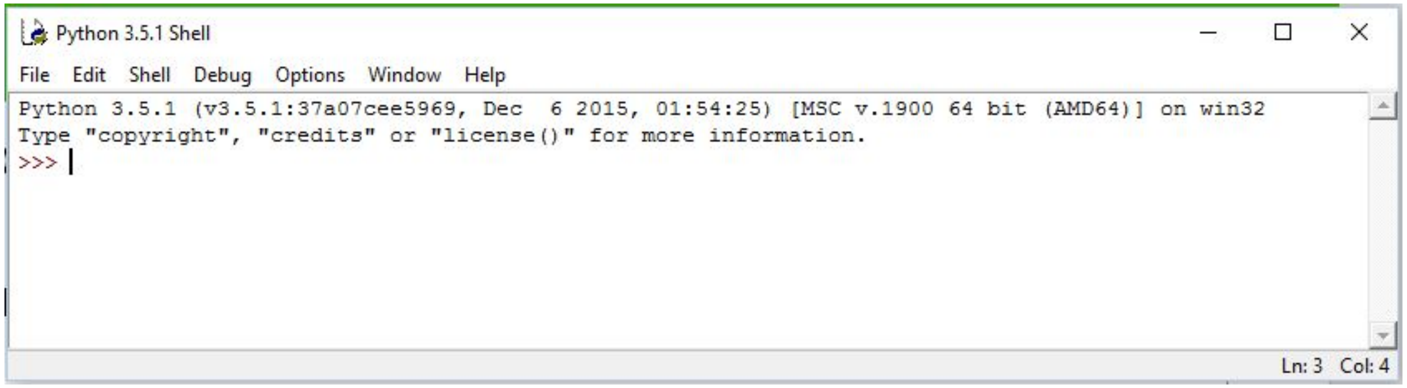
Intro to Python

Let’s get coding!

Where do we program? In IDLE

Click the start button and type IDLE!





```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:54:25) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

Make a mistake!

Type by **button mashing** the keyboard!

Then press enter!

```
asdf asdjlkj;pa j;k4uroei
```

Did you get a big red error message?

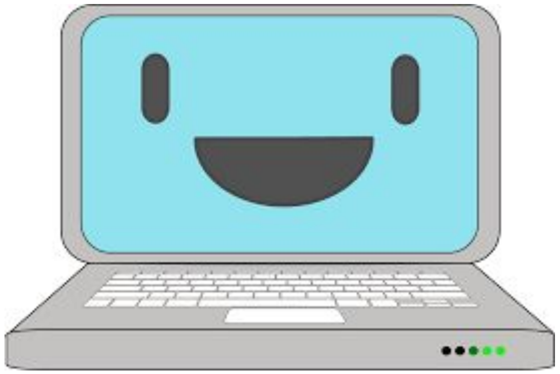
Mistakes are great!

SyntaxError:
Invalid Syntax

Good work you made an error!

ImportError:
No module
named humour

- Programmers make A LOT of errors!
- Errors give us hints to find mistakes
- Run your code often to get the hints!!
- Mistakes won't break computers!



KeyError:
'Hairy Potter'

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

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

We can learn from our mistakes!

Error messages help us fix our mistakes!

We read error messages from bottom to top

3. Where that code is

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

1. What went wrong

2. What code didn't work

Write some code!!

Type this into the window

Then press enter!

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

Did it print:

```
hello world
```

???

Tell me more!

We can `print` on many lines at once!

```
>>> print("""Hello world.
This is me!
Life should be fun for everyone""")
```

Python the calculator!

Try writing some maths into python!

```
>>> 1 + 5


>>> 2 - 7


>>> 2 * 8


>>> 12/3
```

A calculator for words!

What do you think these bits of code do?

Try them and see!

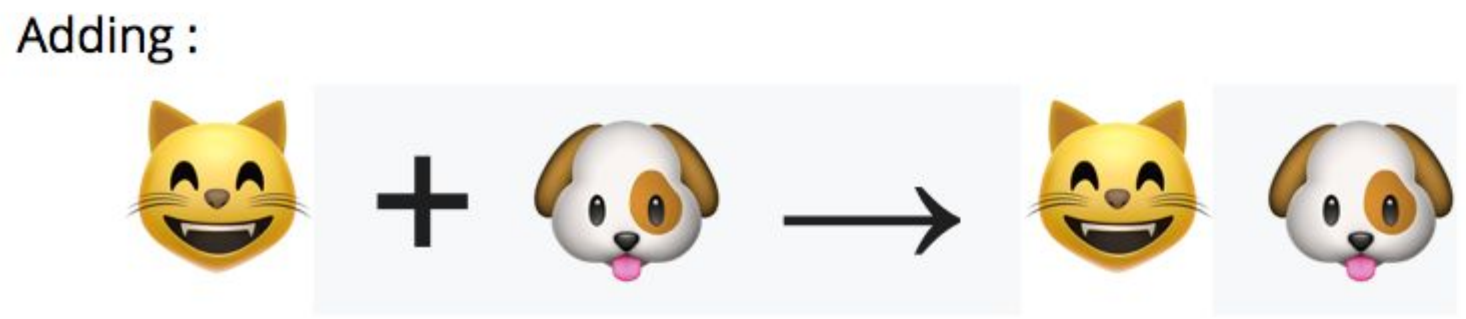
```
>>> "cat" + "dog"
```

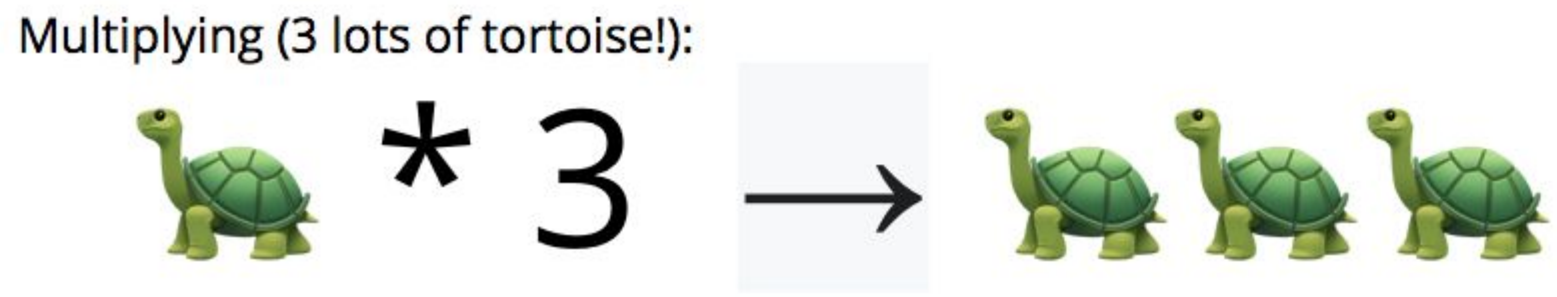
```
>>> "tortoise" * 3
```

Strings!

Strings are things with `"quotes"`

To python they are essentially just a bunch of pictures!





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()

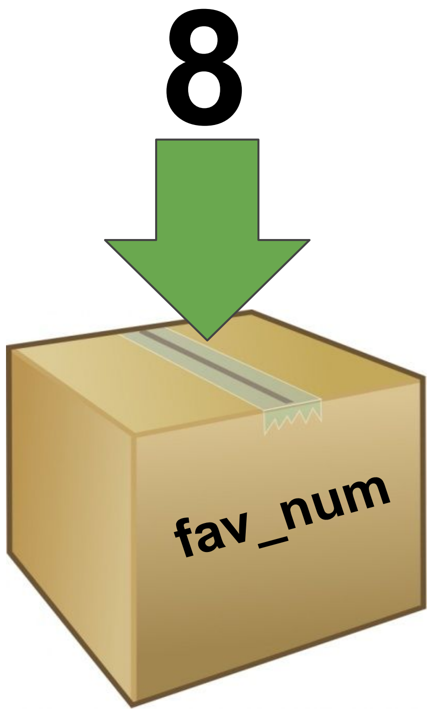
```
>>> str(5) + "5"
```

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
```

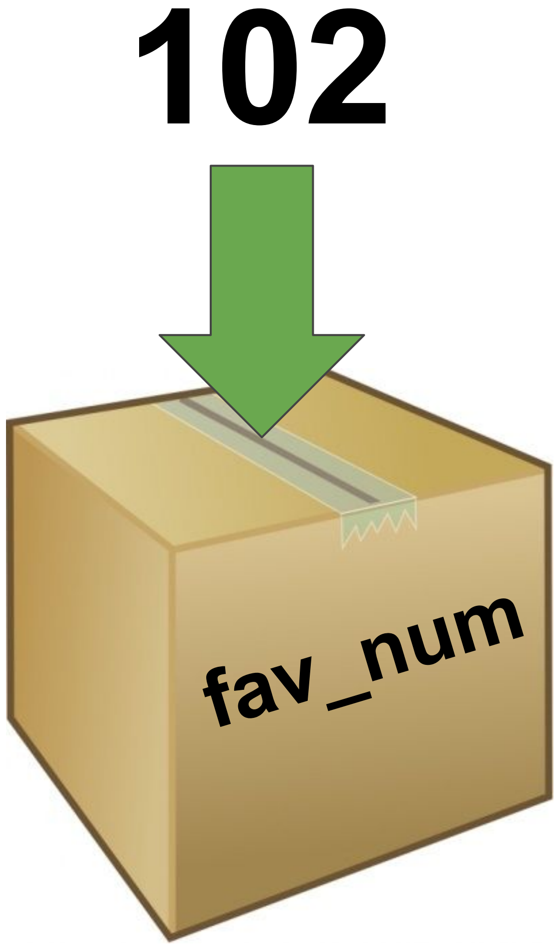
But writing 8 is much shorter than writing `fav_num`???

Variables

Variables are useful for storing things that change

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

Try changing `fav_num` to **102**.



Variables

We're able to use our code for a new purpose, without rewriting everything:

```
fav_num - 6
    => 96
```

```
fav_num + 21
    => 123
```

```
fav_num * 2?
    => 204
```

```
fav_num / 2?
    => 51
```

No variables VS using variables

4
Changes

1
Change

```
8 - 6        102 - 6
8 * 2        102 * 2
8 + 21       102 + 21
8 / 2        102 / 2
```

```
fav_num = 8          fav_num = 102
fav_num - 6          fav_num - 6
fav_num * 2          fav_num * 2
fav_num + 21         fav_num + 21
fav_num / 2          fav_num / 2
```

Reusing variables

We can replace values in variables:

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

What will this output?

What can we store?

We can put any value in a variable:

```
apples = 5 + 5
print(apples)
apples = apples - 1
print(apples)
apples = "Delicious"
print(apples)
```

What will this output?

Variables

Your turn!

Can you guess what each `print` will do?

```
>>> x = 3
>>> print(x)

>>> print(x + x)

>>> y = x
>>> print(y)

>>> y = y + 1
>>> print(y)
```

Switcharoo - Making copies!

Set some variables!

```
>>> x = 3
>>> y = x
>>> x = 5
```

What do x and y contain now?

Let’s find out together!

Switcharoo - Making copies!

Set some variables!

```
>>> x = 3
>>> y = x
>>> x = 5
```

What do x and y contain now?

```
>>> x
5
>>> y
3
```

y hasn't changed because it has a copy of x in it!

Different data!

There are lots of types of data! Our main 4 ones are these:

Strings

Things in quotes used for storing text

```
“This is a string”
```

Ints

Whole numbers we can do maths with

```
a = 1
b = 2
print(a + b)
```

Floats

Decimal numbers for maths

```
a = 1.5
b = 2.0
print(a / b)
```

Booleans

For `True` and `False`

```
a = 5 > 3
boring = False
```

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!

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!

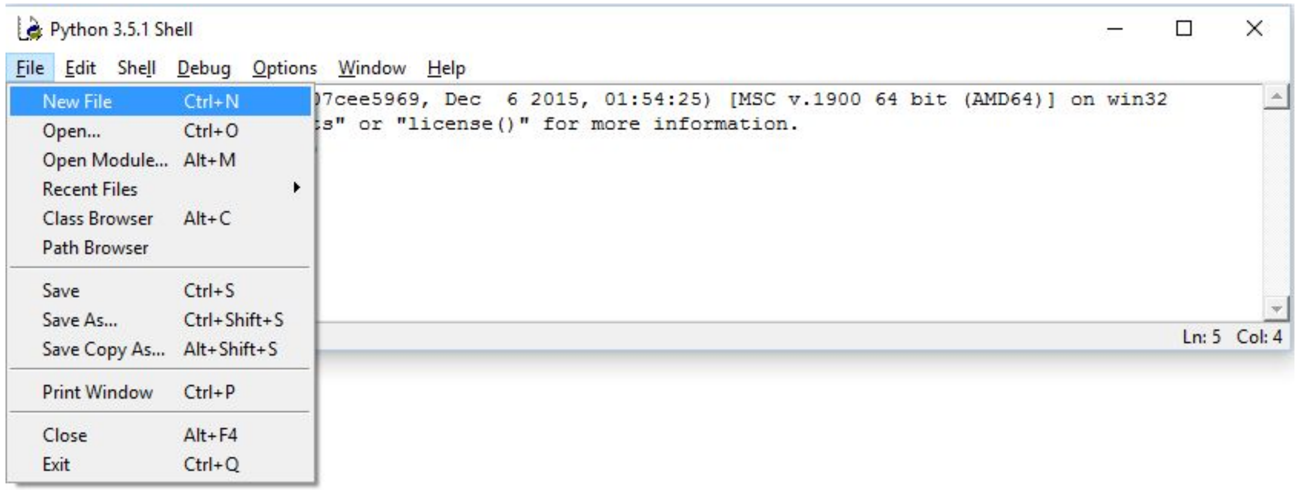
How would we ask somebody for their favourite type of cake?

How would we print their answer?

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

Coding in a file!

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



```
)7cee5969, Dec  6 2015, 01:54:25) [MSC v.1900 64 bit (AMD64)] on win32
:s" or "license()" for more information.
```

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!

Project time!

You now know all about printing and variables!

Let’s put what we learnt into our project

Try to do Part 0 - Part 1

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

<u>If it's raining</u> take an umbrella

Yep it's raining

...... take an umbrella

Booleans (True and False)

Computers store whether a condition is met in the form of

`True` and `False`

To figure out if something is `True` or `False` we do a comparison

Can you guess what these are?

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

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!**

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

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

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
- <u>So it is `True`!</u>

Conditions

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

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

What do you think happens?

```
>>>
```

Conditions

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

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

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

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

Conditions

How about a different number???

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

Conditions

Find out if it’s True!

```
fave_num = 9000
if False :
    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 9000
- And it’s not True that 9000 is less than 10
- So it is False!

Conditions

How about a different number???

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

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

```
>>>
```

Conditions

How about a different number???

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

What do you think happens?

```
>>>
```

Nothing!

If statements

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

This line …

… controls this line

If statements

Actually .....

```
fave_num = 5
if fave_num < 10:
    print("that’s a small number")
    print("and I like that")
    print("A LOT!!")
```

This line ...

... controls anything below it that is indented like this!

If statements

```
fave_num = 5
if fave_num < 10:
    print("that’s a small number")
    print("and I like that")
    print("A LOT!!")
```

What do you think happens?

```
>>>
```

If statements

```
fave_num = 5
if fave_num < 10:
    print("that’s a small number")
    print("and I like that")
    print("A LOT!!")
```

```
>>> that’s a small number
>>> and I like that
>>> A LOT!!
```

If statements

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

What happens?

If statements

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

What happens?

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

If statements

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

What happens?

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

But what if we want something different to happen if the word isn’t "GPN"

Else statements

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

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

What happens?

Else statements

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

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

What happens?

```
>>> The word isn’t GPN :(
```

Elif statements

elif
Means we can give specific instructions for other words

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

What happens?

Elif statements

elif
Means we can give specific instructions for other words

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

What happens?

```
>>> YUMMM Chocolate!
```

Project Time!

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

See `if` you can do Part 2

The tutors will be around to help!

Files

Filing it away!

What happens if we want to use different data in our program? What if that data is too big to write in with the keyboard?

We'd have to change our code!!

It would be better if we could keep all our data in a file and just be able to pick and choose what file we wanted to play today!

people.txt

```
Aleisha,brown,black,hat
Brittany,blue,red,glasses
Charlie,green,brown,glasses
Dave,blue,red,glasses
Eve,green,brown,glasses
Frankie,hazel,black,hat
George,brown,black,glasses
Hannah,brown,black,glasses
Isla,brown,brown,none
Jackie,hazel,blonde,hat
Kevin,brown,black,hat
Luka,blue,brown,none
```

Opening files!

To get access to the stuff inside a file in python we need to **open** it!

That doesn't mean clicking on the little icon!

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

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

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

A missing file causes an error

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

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

You can read a whole file into a string

```
>>> f = open("haiku.txt", "r")
>>> my_string = f.read()
>>> my_string
'Wanna go outside.\nOh NO!
Help! I got outside!\nLet me
back inside!

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

haiku.txt

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

Write to files!

You can also write to files!

```
f = open("newfile.txt", "a")
f.write("This is my new line!")
```

Notice we used `"a"` instead of `"r"`? We opened it in write mode!

This will create a new file if it doesn't exist, and add the new line to the bottom of the file.

Closing Time

Always remember to close your file when you’re finished with it:

```
    f.close()
```

This will close your file and save it.

Project time!

Don't `file` that knowledge away

Use it in the next section of the project!

Try to do Part 3 - Part 4

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?

```python
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~~

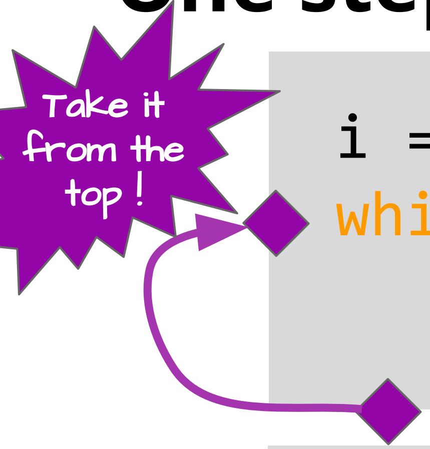
i = 1

UPDATE TIME !

```
i is 0
```

Introducing … while loops!

One step at a time!



```
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!

I 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
```

MY VARIABLES

~~i = 0~~

i = 1

```
i is 0

i is 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
```

MY VARIABLES

~~i = 0~~

~~i = 1~~

i = 2

```
i is 0

i is 1
```

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 !

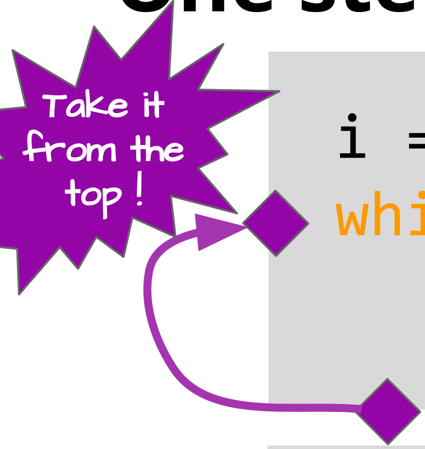
```
i is 0

i is 1

i is 2
```

Introducing … while loops!

One step at a time!



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

MY VARIABLES

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

```
i is 0

i is 1

i is 2
```

Introducing ... `while` loops!

One step at a time!

3 IS NOT less than 3 !

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

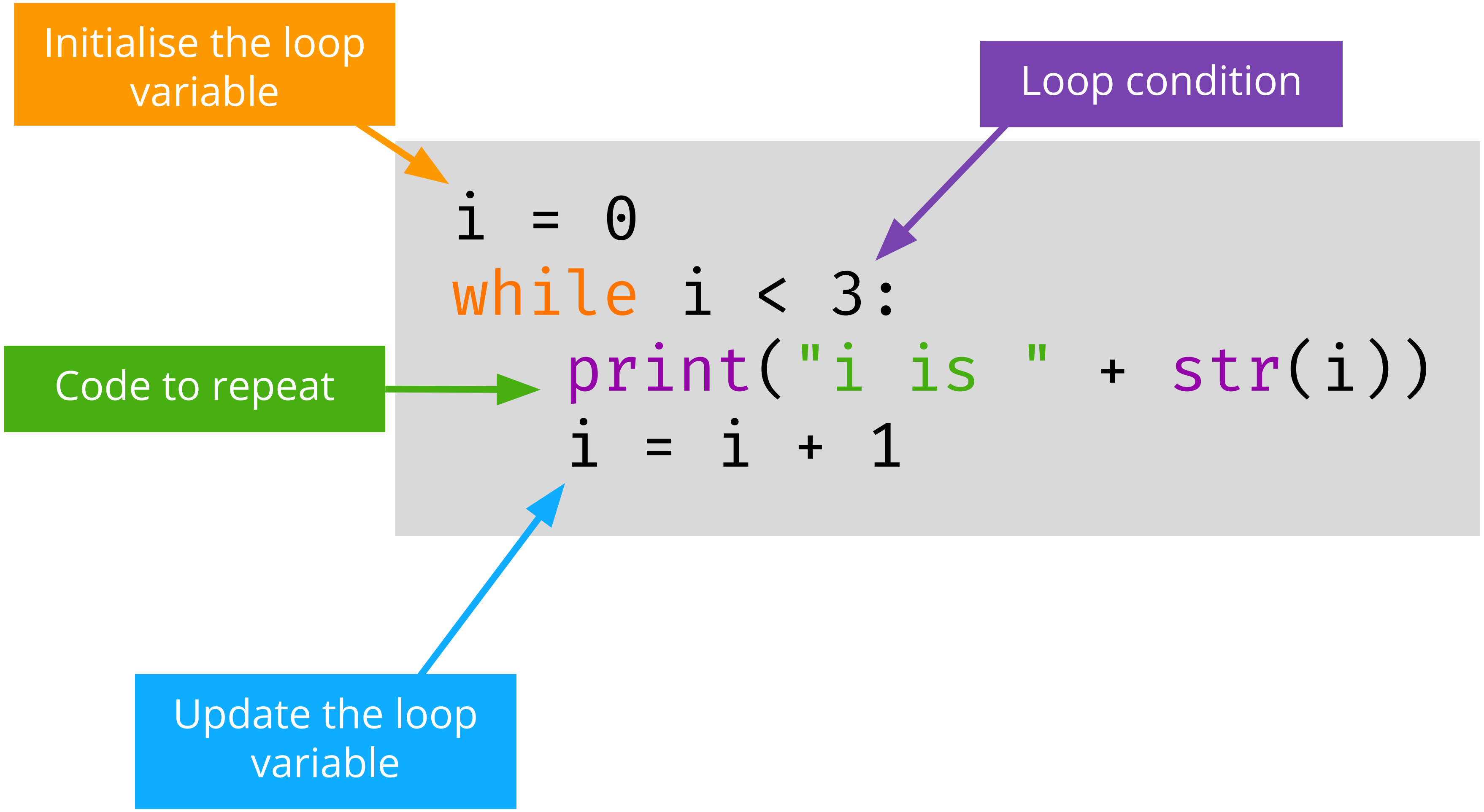
MY VARIABLES

~~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!



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

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?

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

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

Infinite loop!

Sometimes we want our loop to go forever!

So we set a condition that is always True!

We can even just write True!

```
while True:
    print("Are we there yet?")
```

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 Part 5 and Part 6!

And extensions 7 and 8!

The tutors will be around to help!

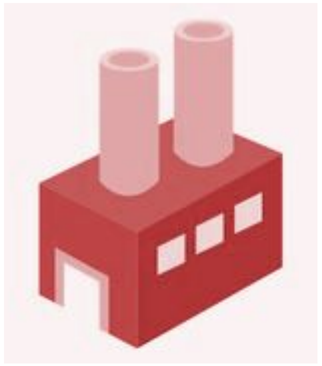
Functions!

Simpler, less repetition, easier to read code!

How functions fit together!

Functions are like factories!

Metal Worker



Your main factory!

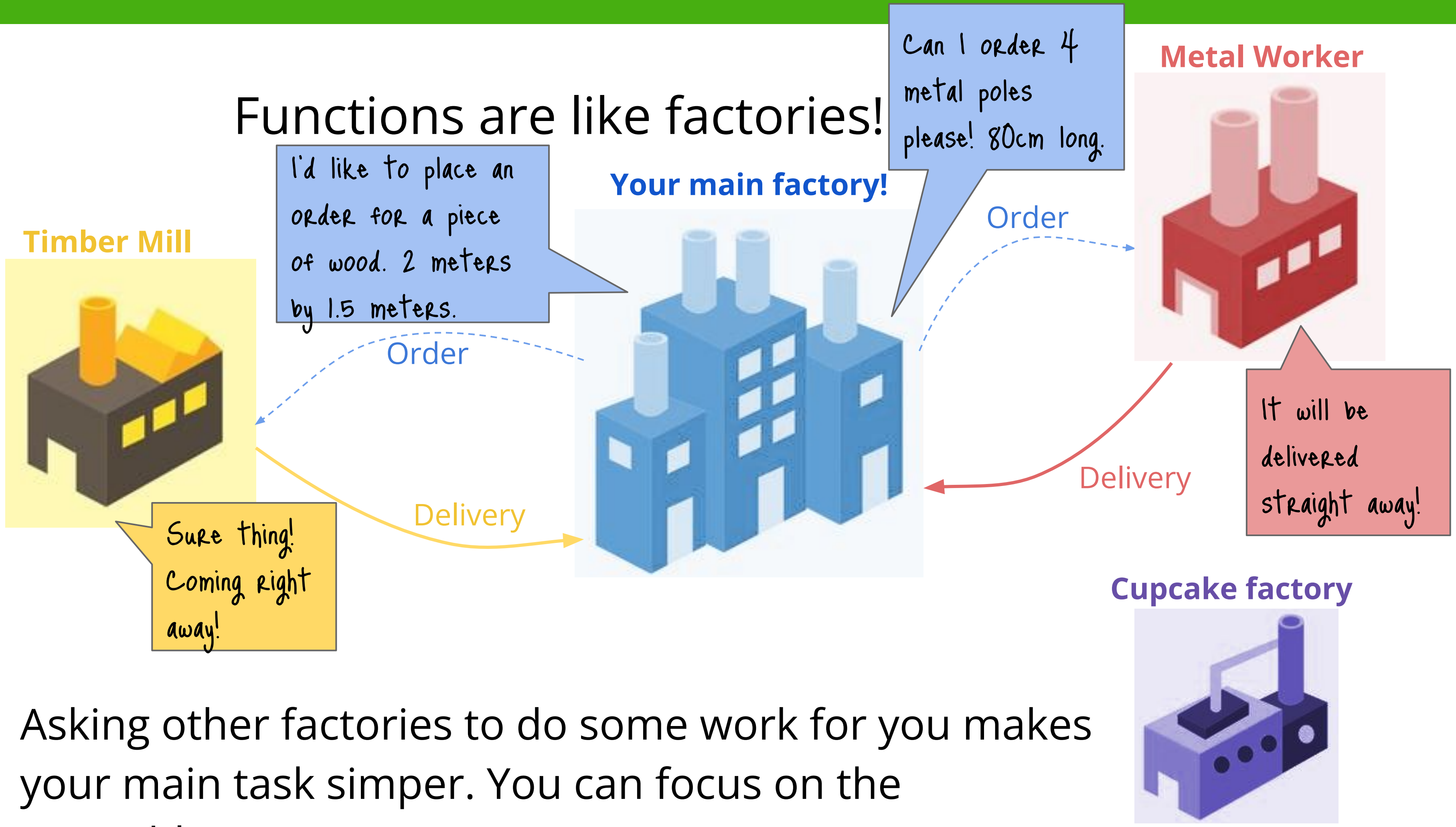
Timber Mill

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!


Metal Worker
Can I order 4 metal poles please! 80cm long.
Your main factory!
Order
Timber Mill
I'd like to place an order for a piece of wood. 2 meters by 1.5 meters.
Order
Delivery
Delivery
It will be delivered straight away!
Sure thing! Coming right away!
Cupcake factory

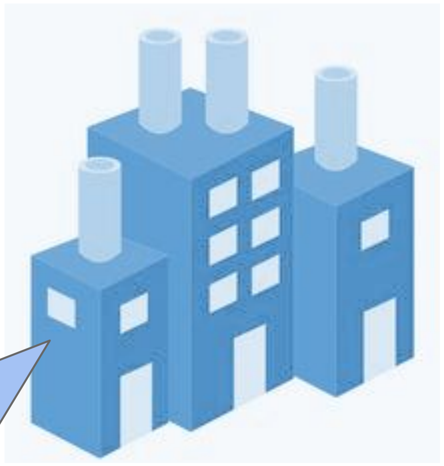
Asking other factories to do some work for you makes your main task simper. You can focus on the assembly!

How functions fit together!

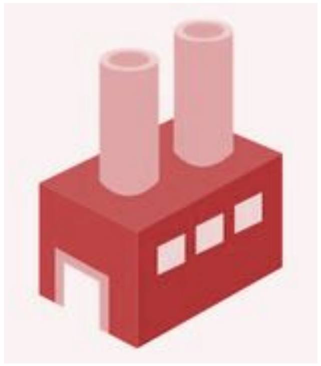
Functions are like factories!

Timber Mill

Your main factory!



Metal Worker



Cupcake factory

Look at this beautiful table I made!

Outsourcing made it simple!

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!

```python
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
```

Pretty Word Printer

Create a new file and make a pretty word printer! It can print any word you like.

1. Define a function called pretty_word_print
2. Set a variable called word
3. Have the function print out some decorative marks as long as the word above and below the word like these examples:

```
~~~
GPN
~~~
```

```
***********
Hello World
***********
```

4. Call your function in your file as many times as you like!

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 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

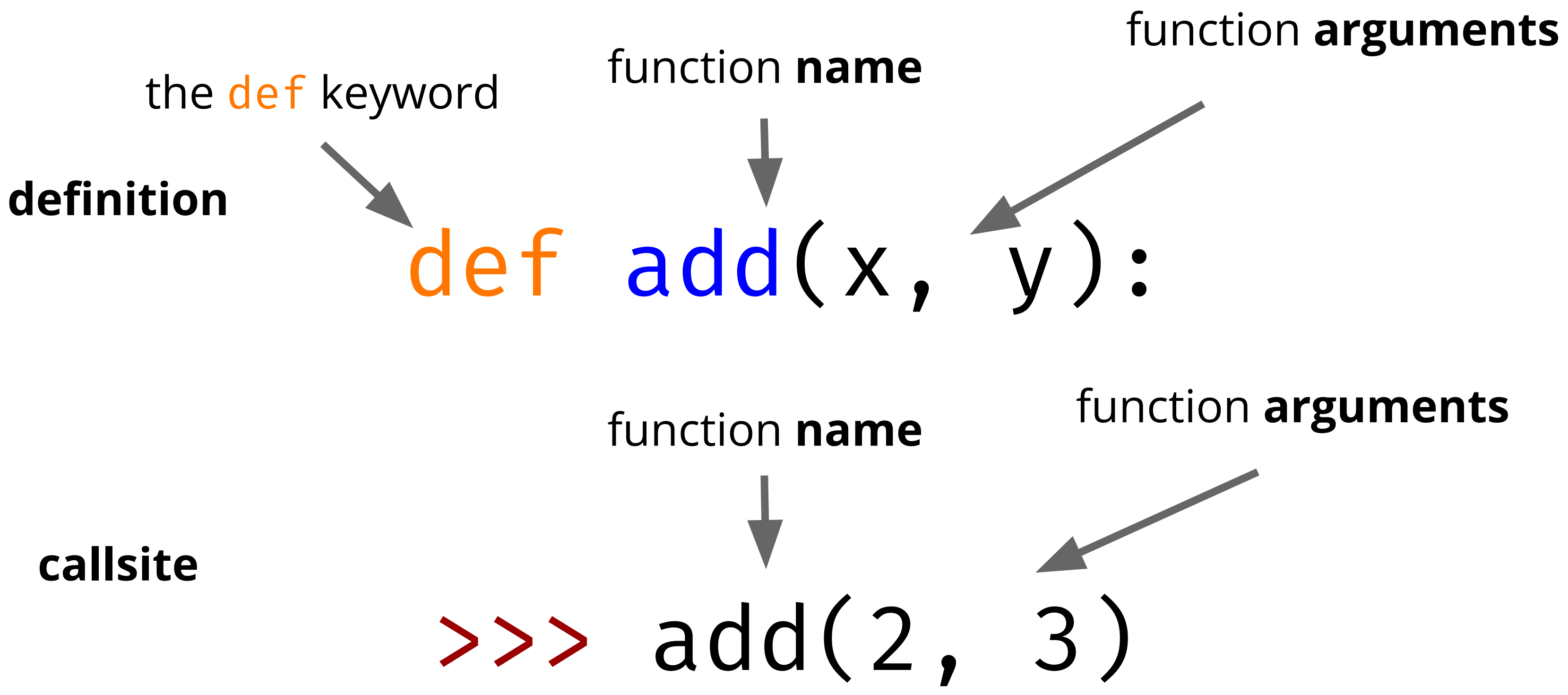
Changing a variable in a function **only changes it *inside* the function.**

```
>>> 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



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

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

Pretty Word Printer

At the moment our pretty word printer always prints the same word. Let's fix that!

Edit your pretty word printer function:

1. Change your function so it takes in an argument called word
2. Remove the line where you set word as a variable, now we are passing in word
3. Change the places where you called your pretty word printer, so now you pass in a word as an argument (make sure you pass in a string).
4. Try calling your function multiple times, but with different words

Calling your function with these arguments might look like this:

```
pretty_word_print("Hi everyone")
pretty_word_print("GPN is cool")
```

```
***********
Hi everyone
***********
************
GPN is cool
************
```

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.

First name finder

Make a new file and write a function that takes in a full name (multiple words, separated by spaces) **and returns the first name.**

1. Define a function called get_first_name
2. Make it take 1 argument called full_name
3. Use `full_name.split(" ")` to split the name into a list. Store it in a variable.
4. Return the item at index 0, that's the first name!
5. Call your function, store the returned value and print it out!

Try some different input arguments out! If you do `“Ada Lovelace”` does it return “Ada”. What about if you do `“Alan Mathison Turing”`

Project time!

Now go be functional.

Do the next part of the project!

Try to do Extensions 9 and 10

The tutors will be around to help!