# Intro to Programming

# What is programming?

**Programming is not a bunch of crazy numbers!**

**It's giving computers a set of instructions!**

# A Special Language

A language to talk to dogs!

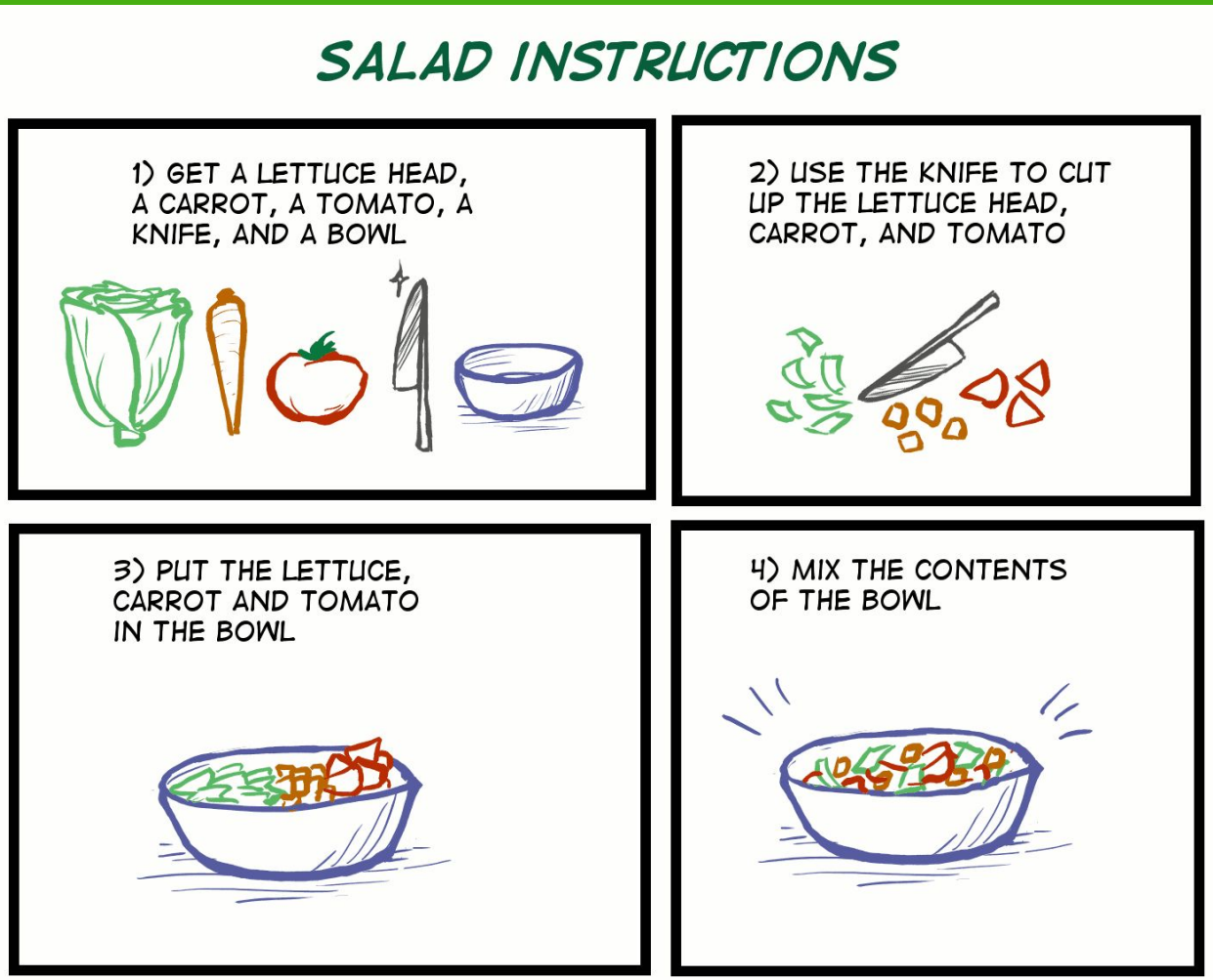

Programming is a language to talk to computers

# People are smart! Computers are dumb!

Programming is like a recipe!

Computers do EXACTLY what you say, every time.

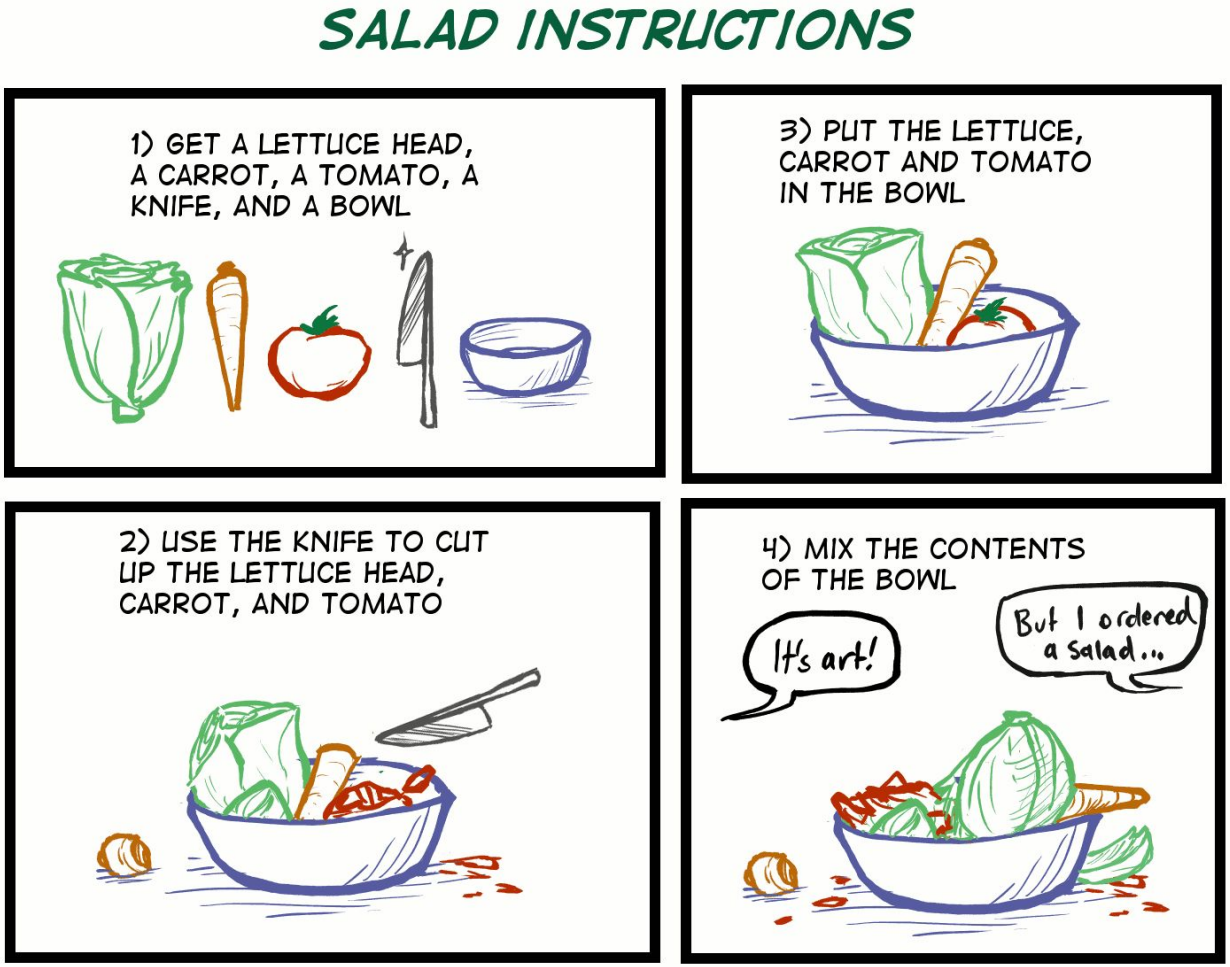
Which is great if you give them a good recipe!



# People are smart! Computers are dumb!

But if you get it out of order….

A computer wouldn't know this recipe was wrong!



# People are smart! Computers are dumb!

Computers are bad at filling in the gaps!

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

# Everyone/thing has strengths!


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

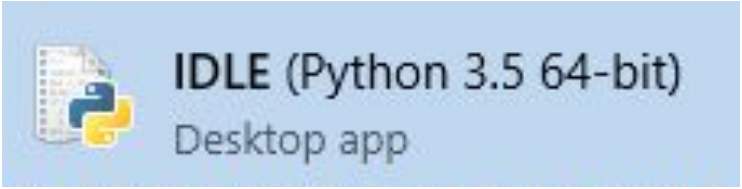
- Does exactly what you tell it
- Does it the same every time
- Doesn't need to sleep
- Will work for hours on end
- Doesn't get bored
- Really really fast
- Get smarter when you tell it how

# Intro to Python

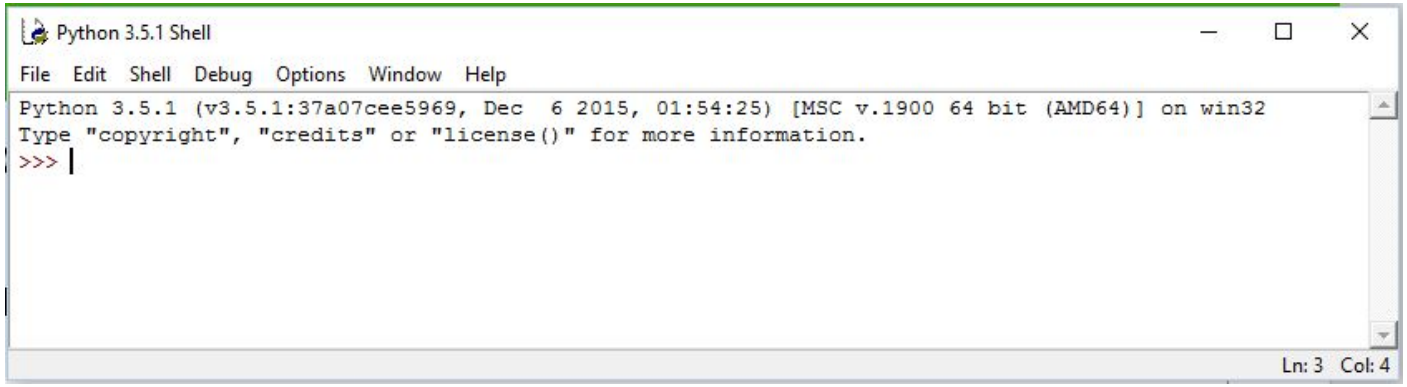
Let’s get coding!

# Where do we program? In IDLE

Click the start button and type IDLE!




Make sure the first number after “Python” is 3!



# 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'

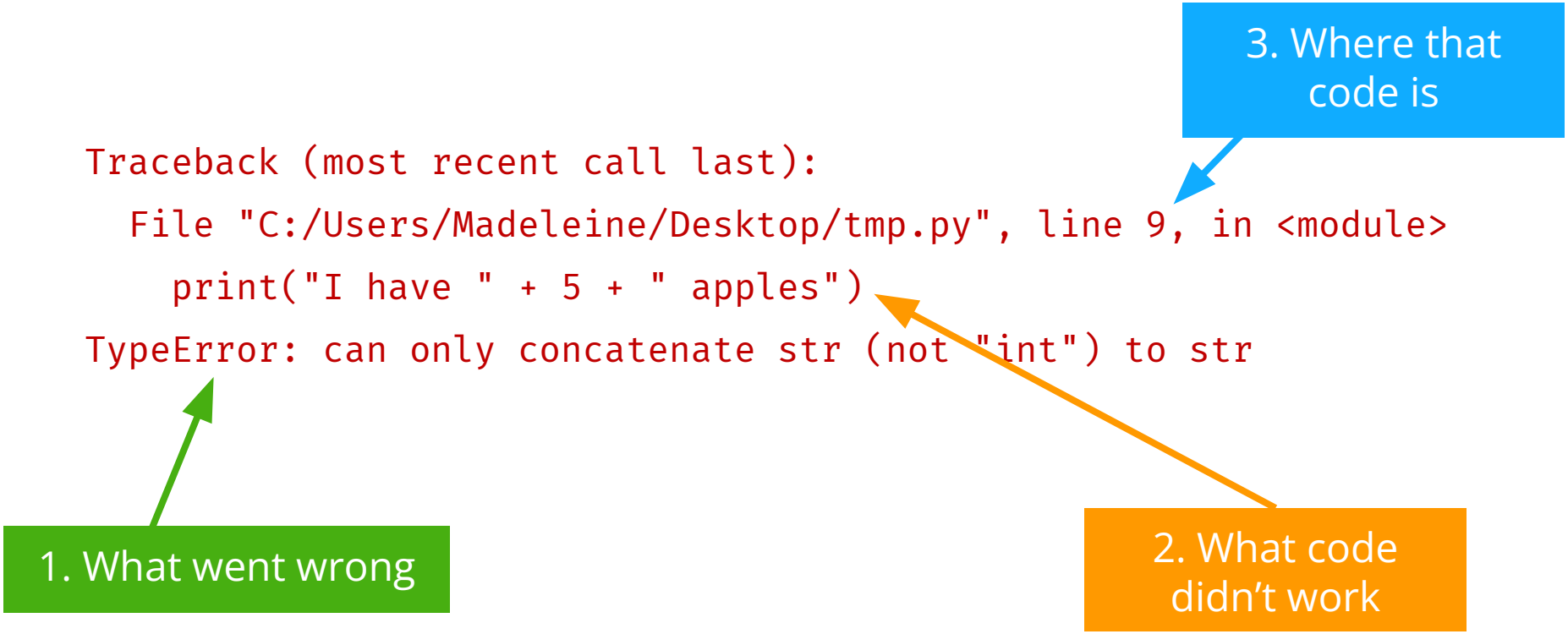
```
AttributeError:
'NoneType' object
has no attribute
'foo'
```

```
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



# 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` things in lots of different ways in python!

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

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

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

# Tell me more!

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

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

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

# Tell me more!

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

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

# Tell me more!

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

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

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

# Tell me more!

We can `print` on many lines at once!

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

# Tell me more!

We can `print` on many lines at once!

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

# Python the calculator!

Try writing some maths into python!

```
>>> 1 + 5
6
>>> 2 - 7


>>> 2 * 8


>>> 12/3
```

# Python the calculator!

Try writing some maths into python!

```
>>> 1 + 5
6
>>> 2 - 7
-5
>>> 2 * 8

>>> 12/3
```

# Python the calculator!

Try writing some maths into python!

```
>>> 1 + 5
6
>>> 2 - 7
-5
>>> 2 * 8
16
>>> 12/3
```

# Python the calculator!

Try writing some maths into python!

```
>>> 1 + 5
6
>>> 2 - 7
-5
>>> 2 * 8
16
>>> 12/3
4
```

# A calculator for words!

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

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

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

# A calculator for words!

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

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

>>> "tortoise" * 3
```

# A calculator for words!

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

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

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

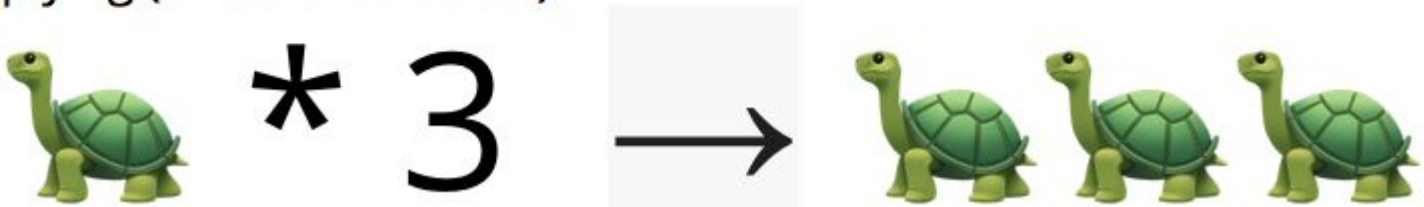
# Strings!

## Strings are things with `"quotes"`

**To python they are essentially just a bunch of pictures!**

Adding :

Multiplying (3 lots of tortoise!):



# Strings!

Strings can have any letters in them, even just 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'
```

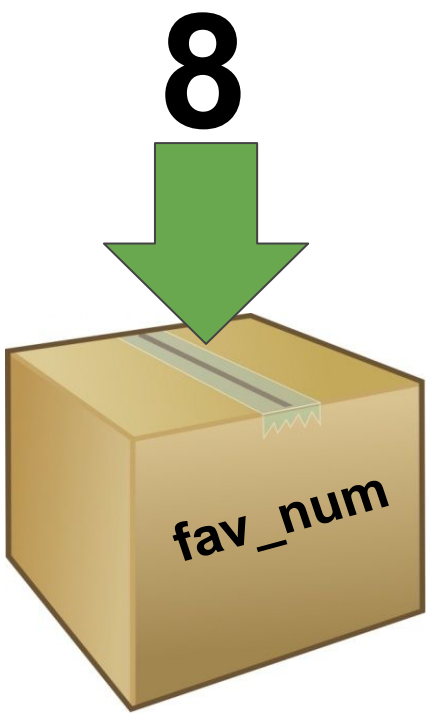
# No Storing is Boring!

**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**

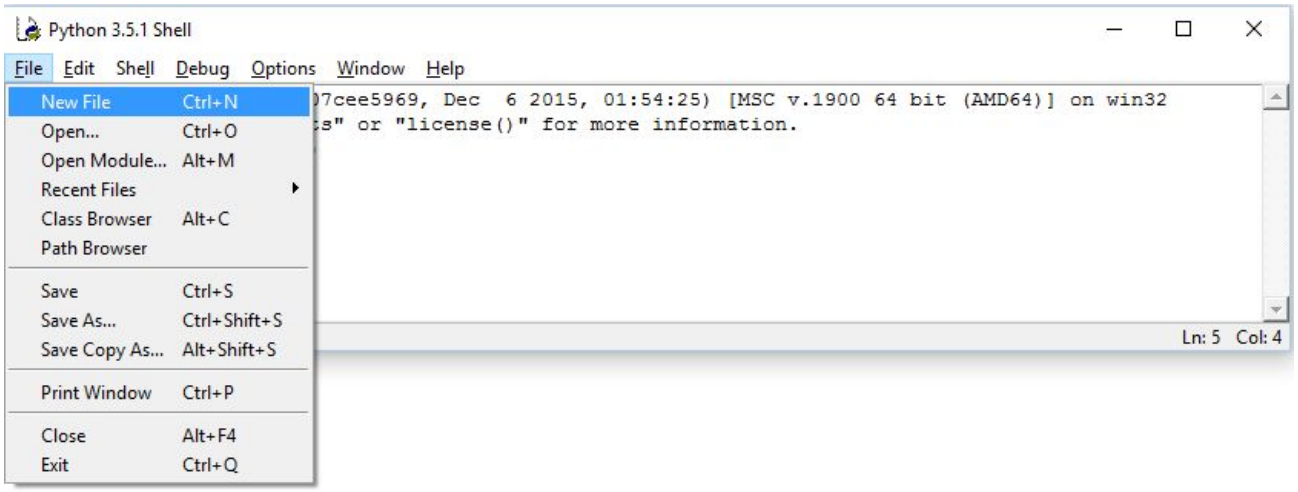
```
f                2
```

**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!

# Project time!

You now know all about printing and variables!

**Let’s put what we learnt into our project**

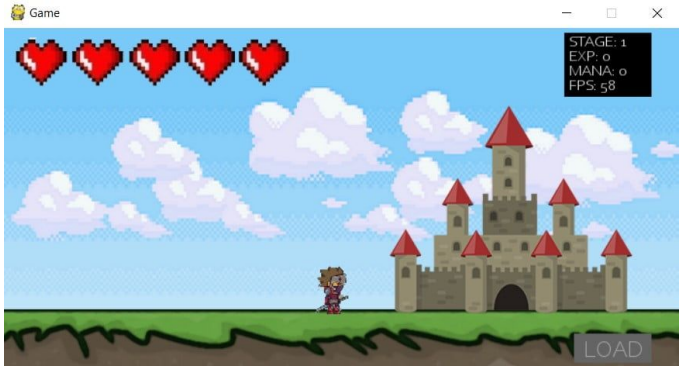
**Try to do the next Part!**

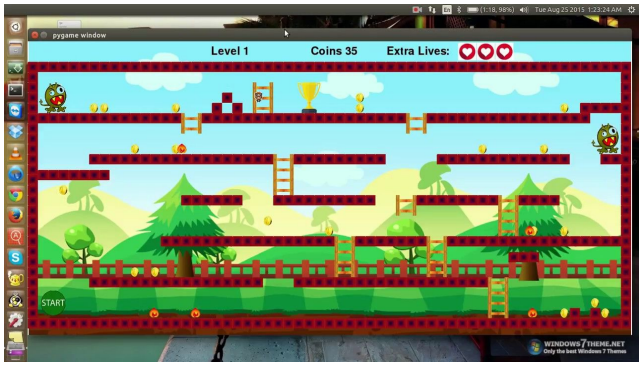
The tutors will be around to help!

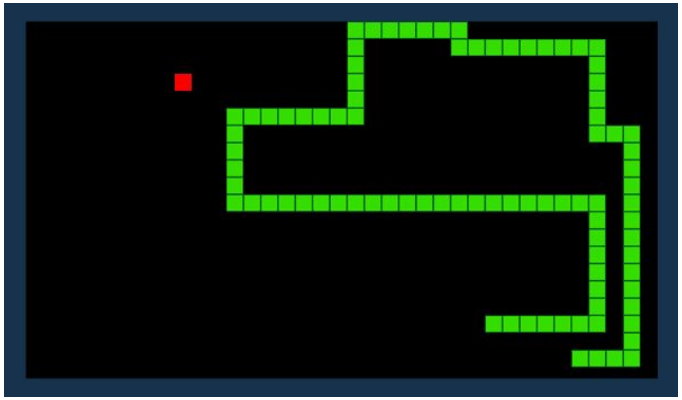
# Intro to Pygame

# What is Pygame?

Pygame is a tool we can use to make cool games using Python like these:



# How do we use it?

First we need to tell Python that we want to use Pygame, and tell it to start the game

```
from pygame import *

init()
```

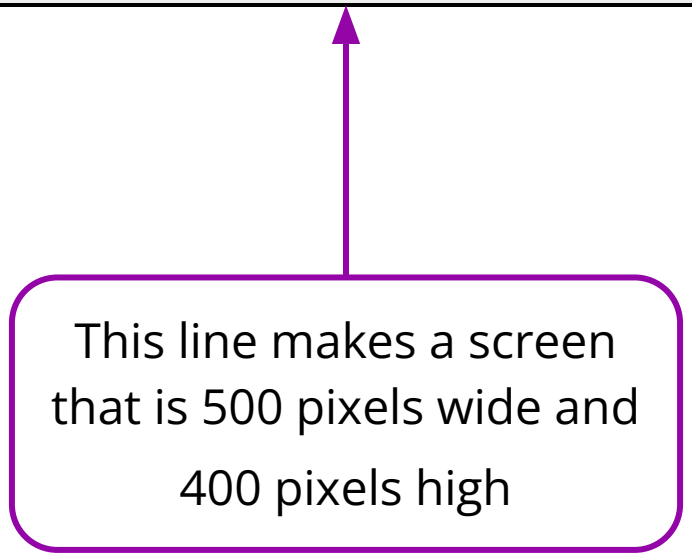
This line tells Python that we want to use pygame

This line starts Pygame

# Make a scene!

We can make a screen to show our game

```
screen = display.set_mode(500, 400)
```



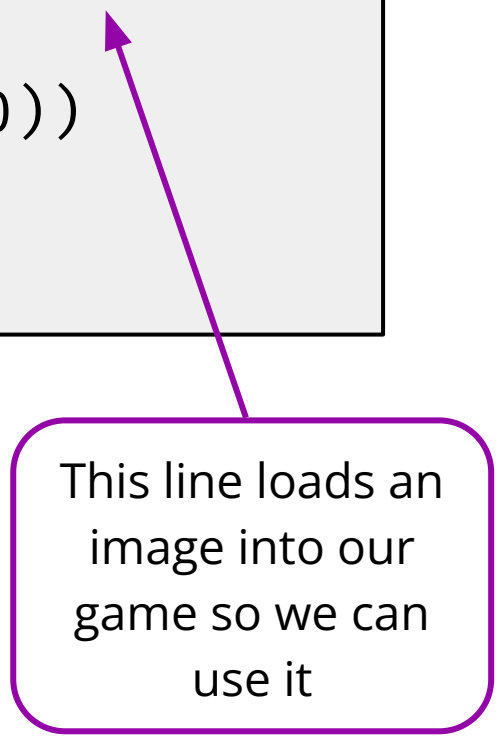
# Make a scene!

Once we have a screen we can put images on it!

```
cat_image = image.load("cat.png")

screen.blit(cat_image, (30, 40))

display.update()
```



# Make a scene!

Once we have a screen we can put images on it!

```
cat_image = image.load("cat.png")

screen.blit(cat_image, (30, 40))

display.update()
```

This line puts the image on the screen at the coordinates 30, 40

This line loads an image into our game so we can use it

# Make a scene!

Once we have a screen we can put images on it!

```
cat_image = image.load("cat.png")

screen.blit(cat_image, (30, 40))

display.update()
```

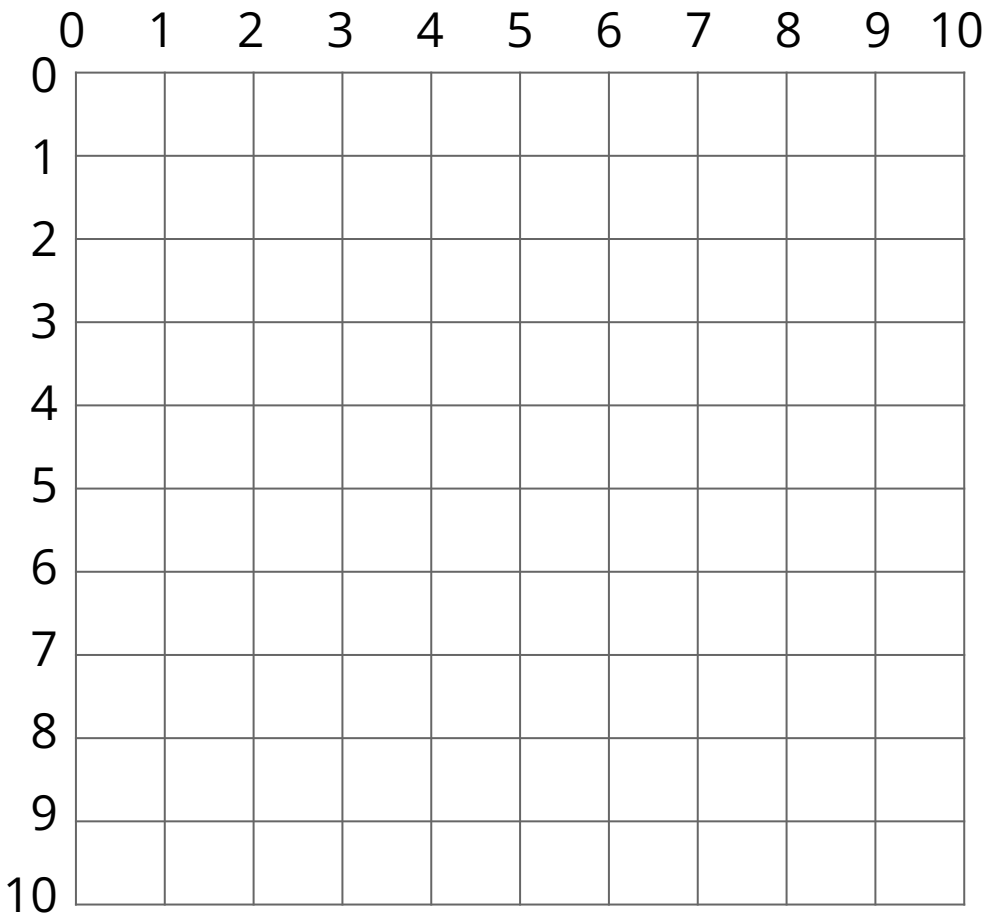
This line tells Pygame to update our display, which will show our new image

This line puts the image on the screen at the coordinates 30, 40

This line loads an image into our game so we can use it

# Pygame Coordinates

Coordinates in Pygame are a little strange...




Because they start at the top left instead of the bottom left

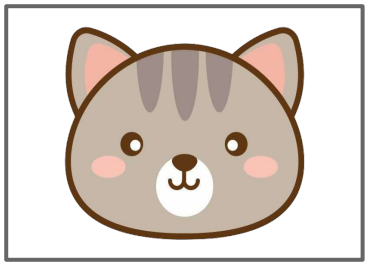
# Pygame Coordinates

When you have an image in pygame like this:

# Pygame Coordinates

When you have an image in pygame like this:

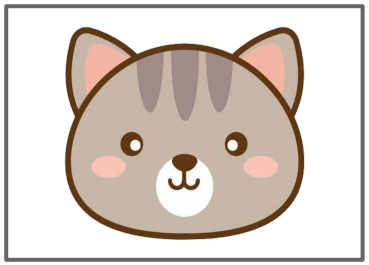
Pygame thinks of it like a rectangle like this:



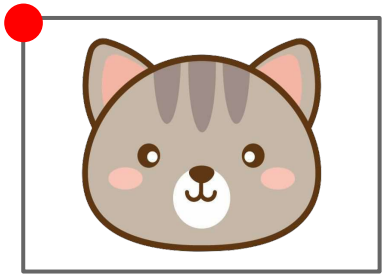
# Pygame Coordinates

When you have an image in pygame like this:

Pygame thinks of it like a rectangle like this:

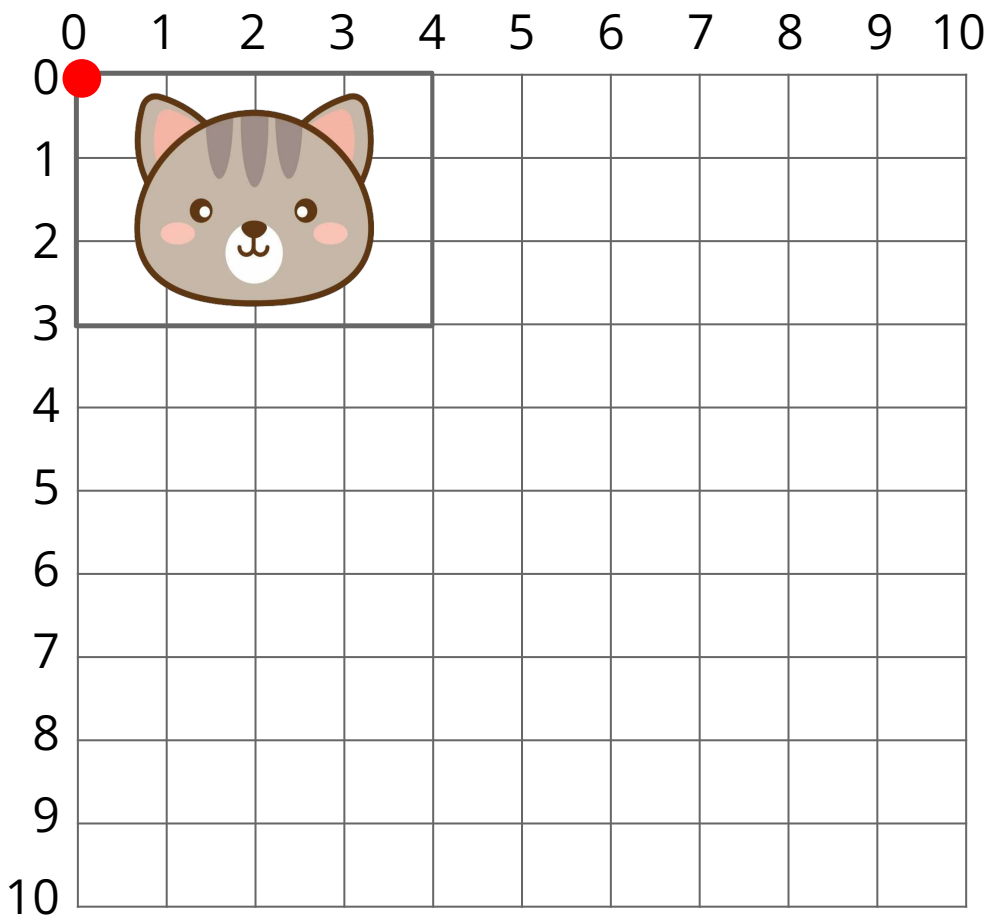


And the coordinates for the image are the top left corner like this:



# Pygame Coordinates

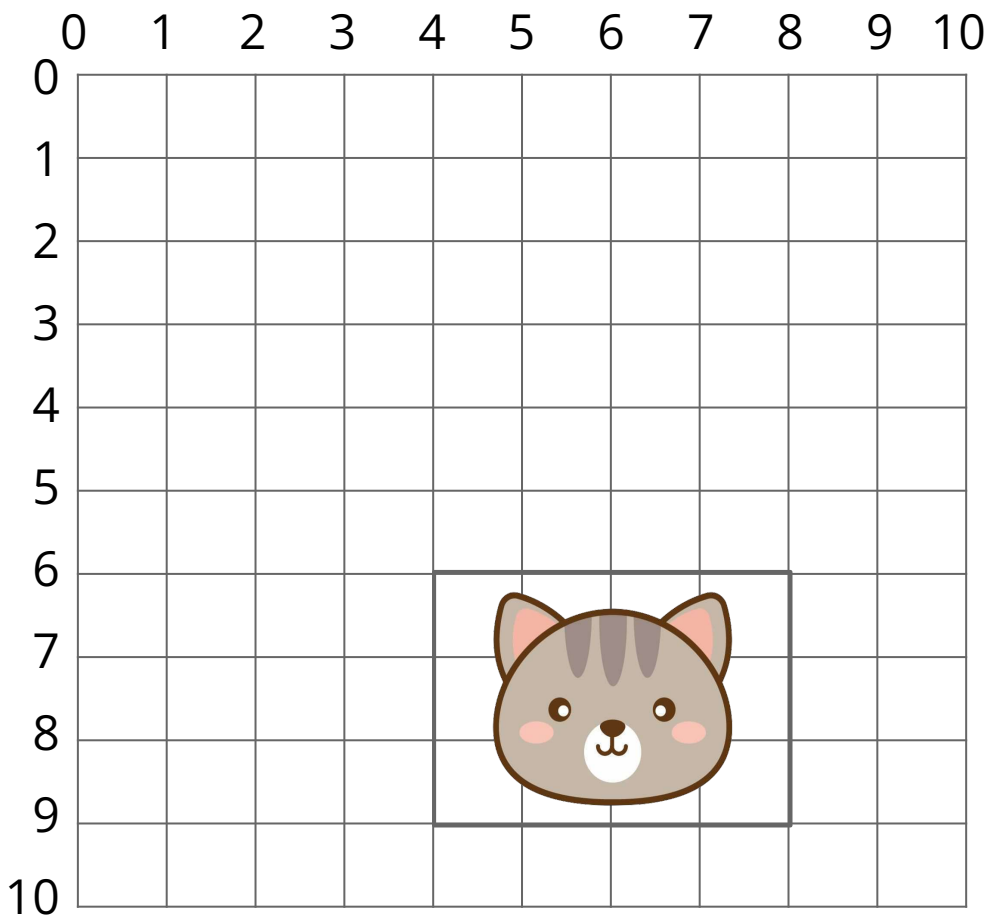
So to if we wanted our cat image to be on the top left we would use the coordinates (0, 0)




That means that the top left corner of the image will be in the top left corner of the screen

# Pygame Coordinates

What are the coordinates of our image now?



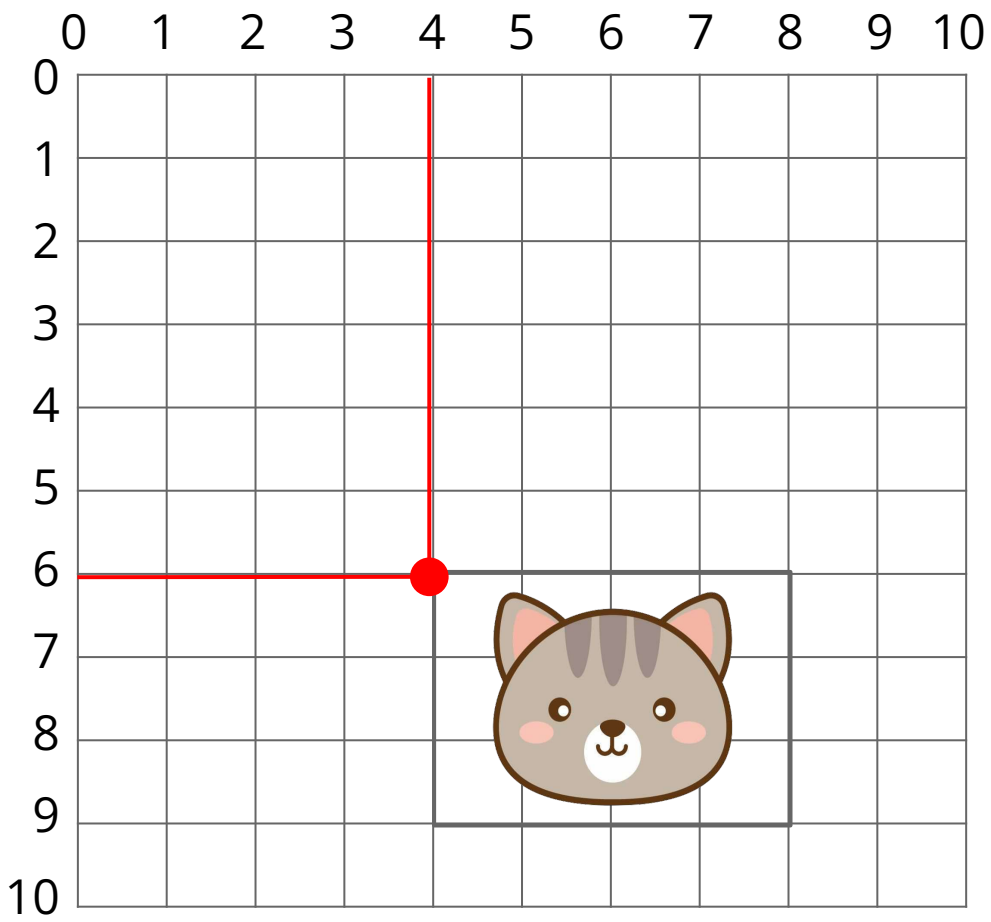

Remember that when we write coordinates we say the x (horizontal) number first and the y (vertical) number second

# Pygame Coordinates

What are the coordinates of our image now?

They are:

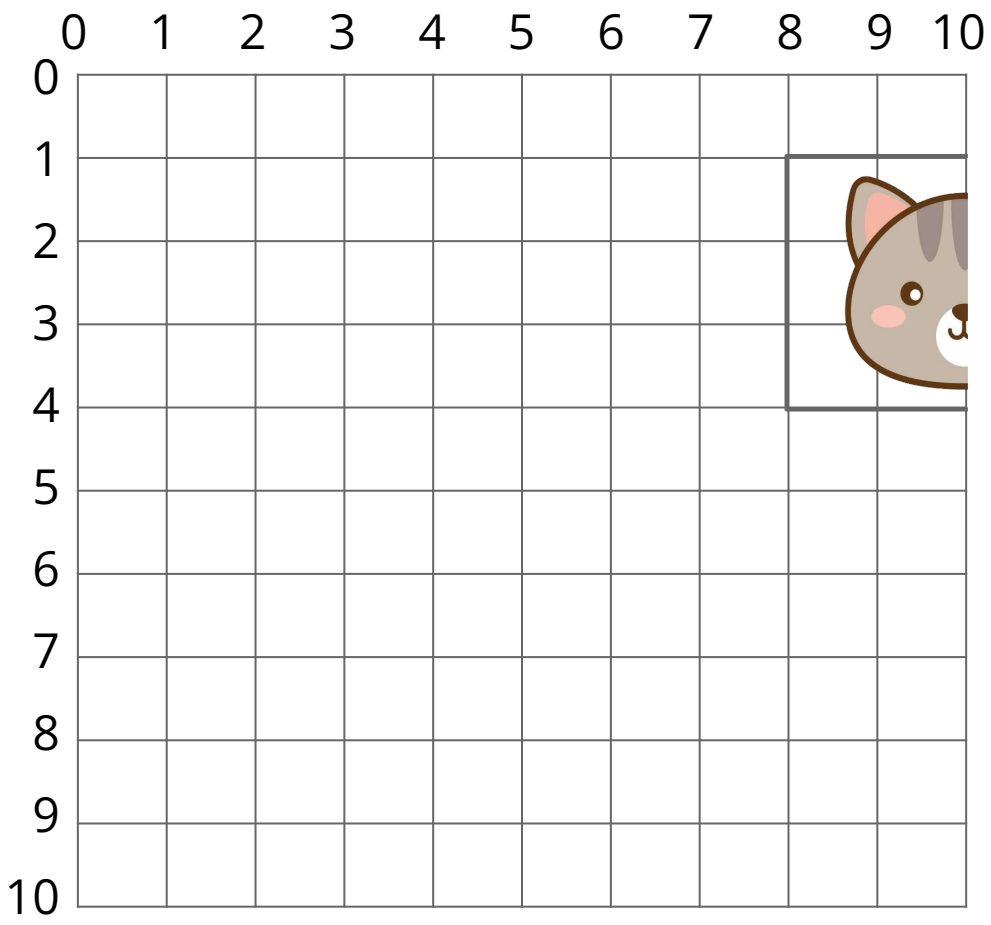
(4, 6)




Remember that when we write coordinates we say the x (horizontal) number first and the y (vertical) number second

# Pygame Coordinates

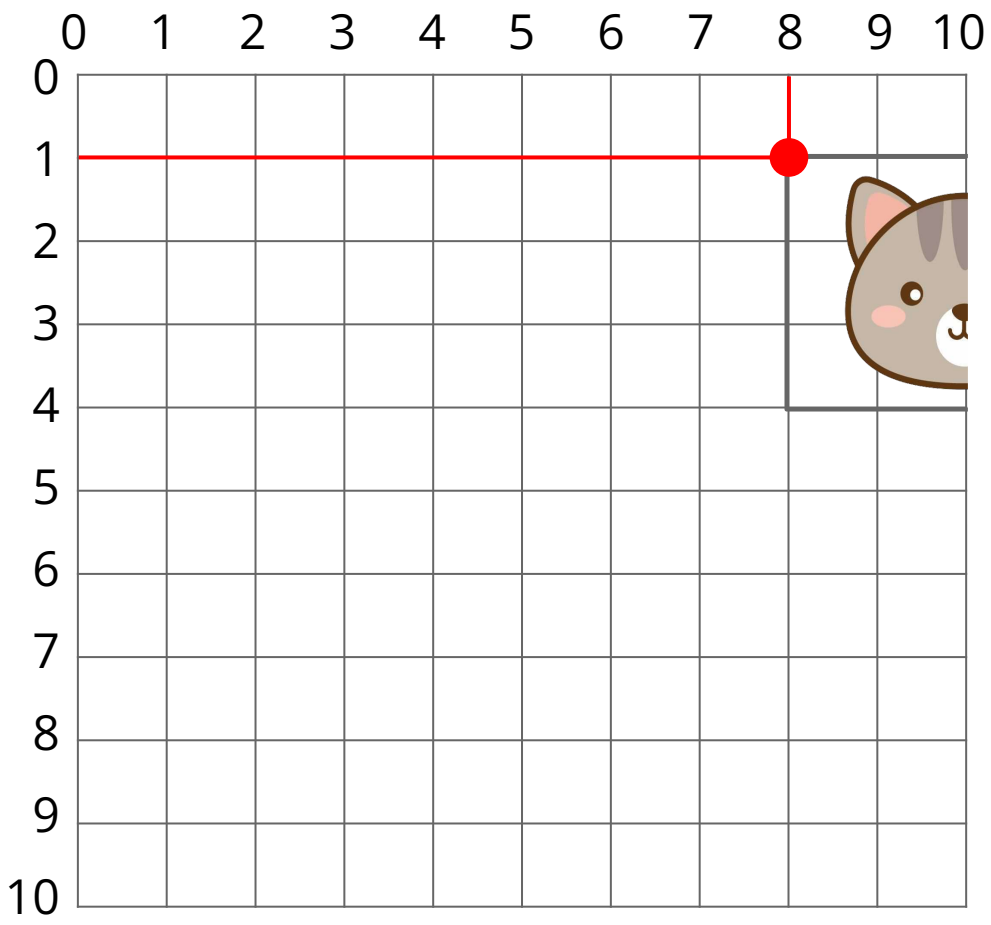
What are the coordinates of our image now?




Sometimes we can put the image off the screen by giving coordinates that are far to the bottom or the right

# Pygame Coordinates

What are the coordinates of our image now?




They are:

(8, 1)

Sometimes we can put the image off the screen by giving coordinates that are far to the bottom or the right

# Project time!

Now that you've **update**d your knowledge...

**Try to do the next Part!**

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

```
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 |

# 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
```

# 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")
```

What do you think happens?

```
>>>
```

# Conditions

## How about a different number???

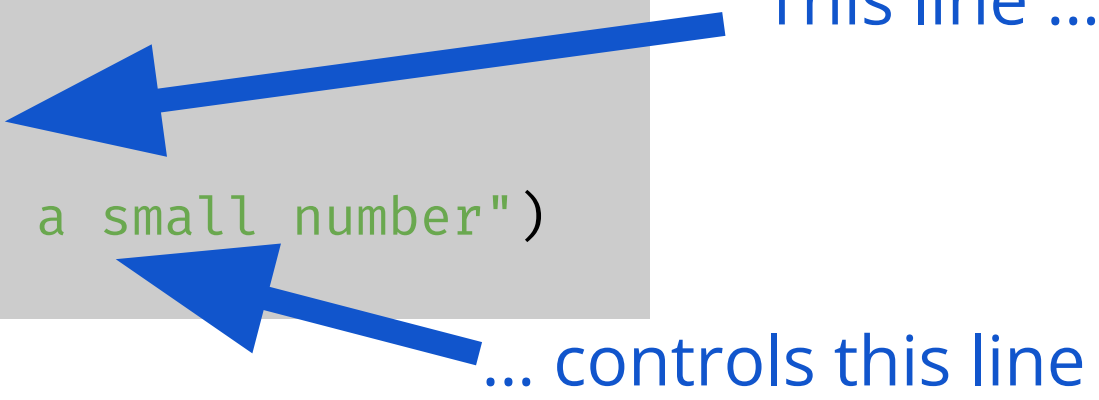
```
fave_num = 9000
if fave_num < 10:
    print("that's a small number")
```

What do you think happens?

```
>>>
```

# If statements

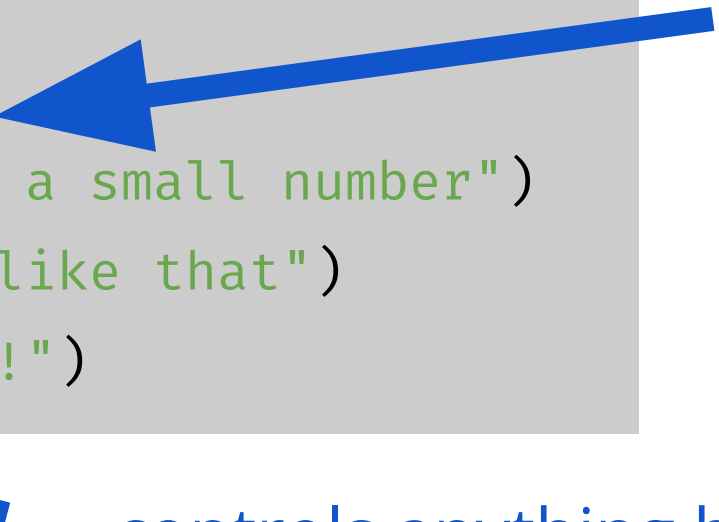
```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
```



# If statements

## Actually .....

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



# 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 the next Part**

The tutors will be around to help!

# Pygame Events!

# Pygame Events

Pygame can do more than just show images on a screen!

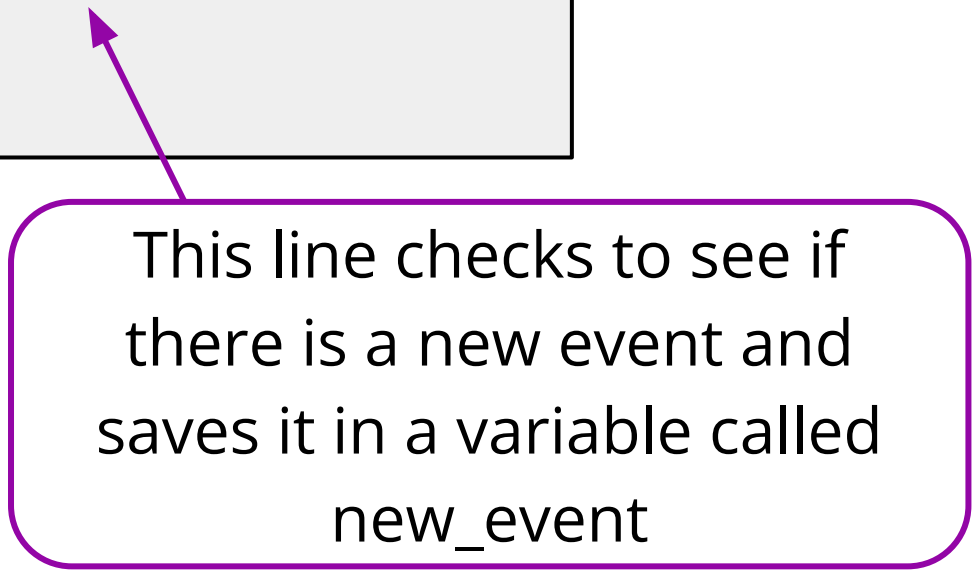
We can use it to figure out if the mouse moved or if a keyboard button was pressed!

These actions are called “events”! Let’s learn how to use them

# How do we use it?

First we need to ask Pygame to tell us what has happened recently

```
while True:
  new_event = event.poll()
```



# How do we use it?

First we need to ask Pygame to tell us what has happened recently

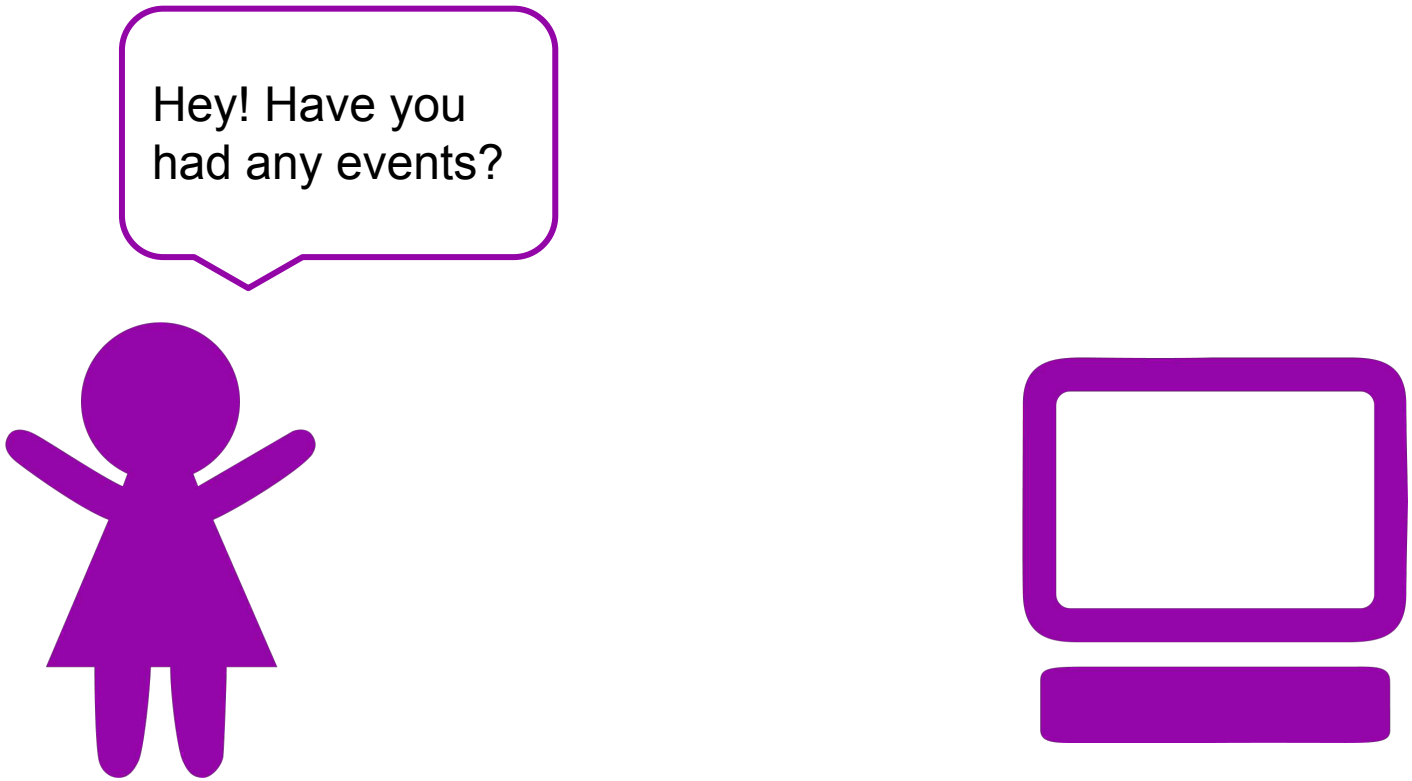
```
while True:
    new_event = event.poll()
```

But what does this line do??

This line checks to see if there is a new event and saves it in a variable called new_event

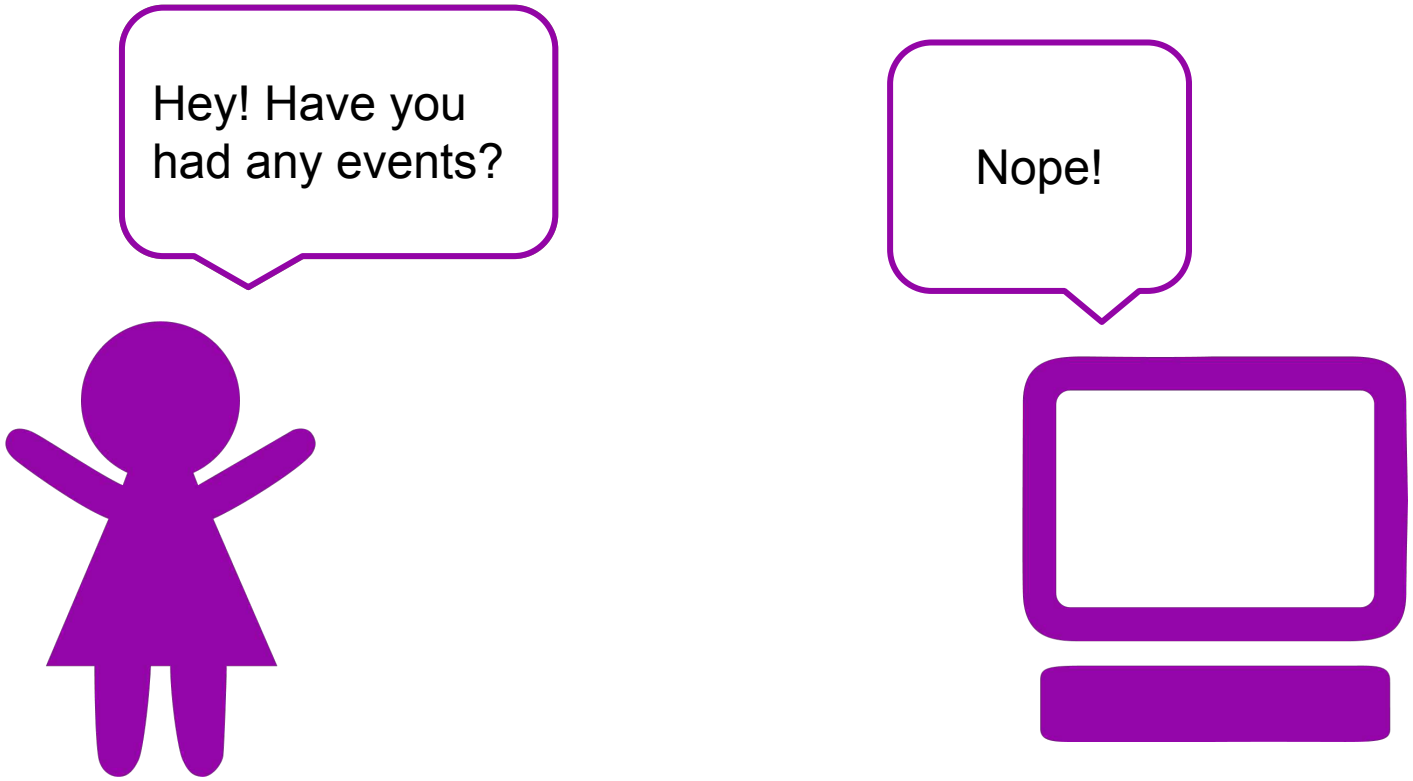
# Looking for Events

Let's think of how Pygame checks for events like this:



# Looking for Events

Let's think of how Pygame checks for events like this:

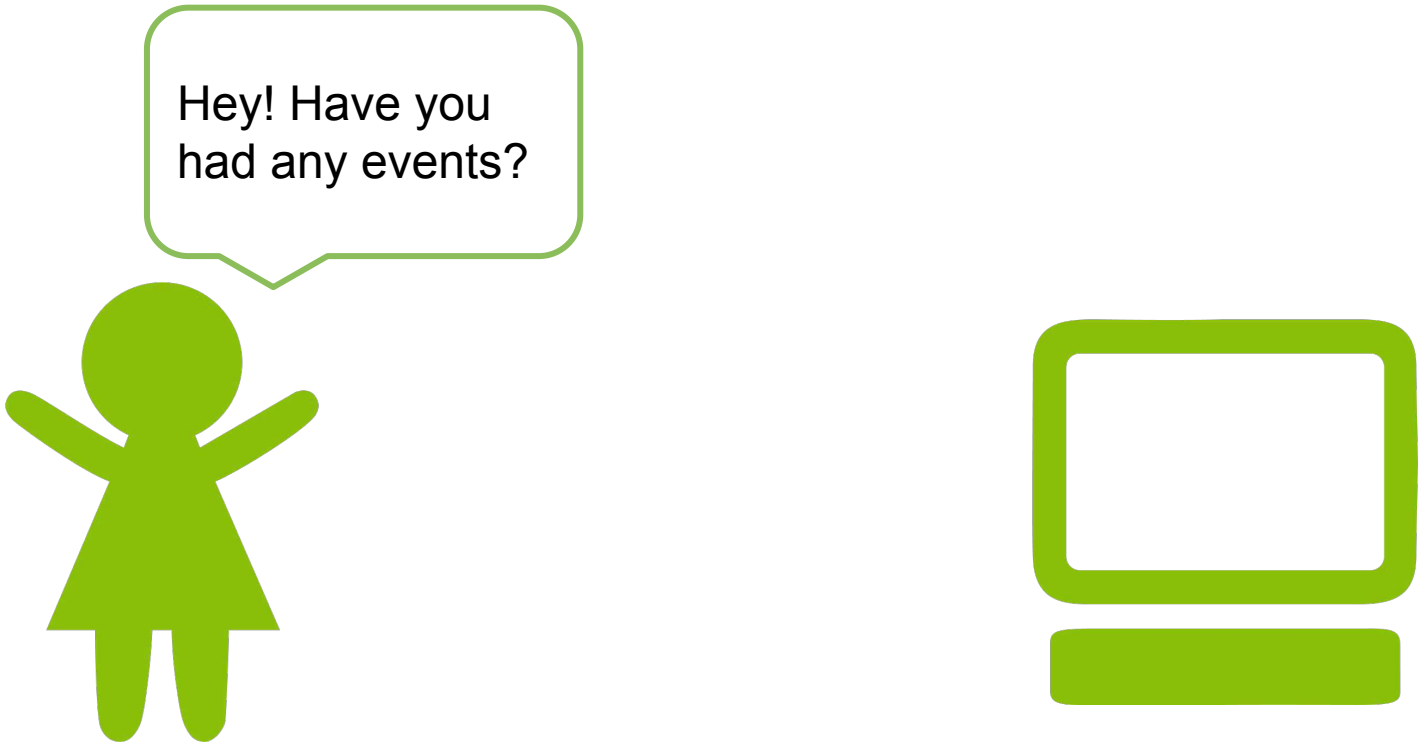


# Looking for Events

If we only ask once then we won't know if an event happens later

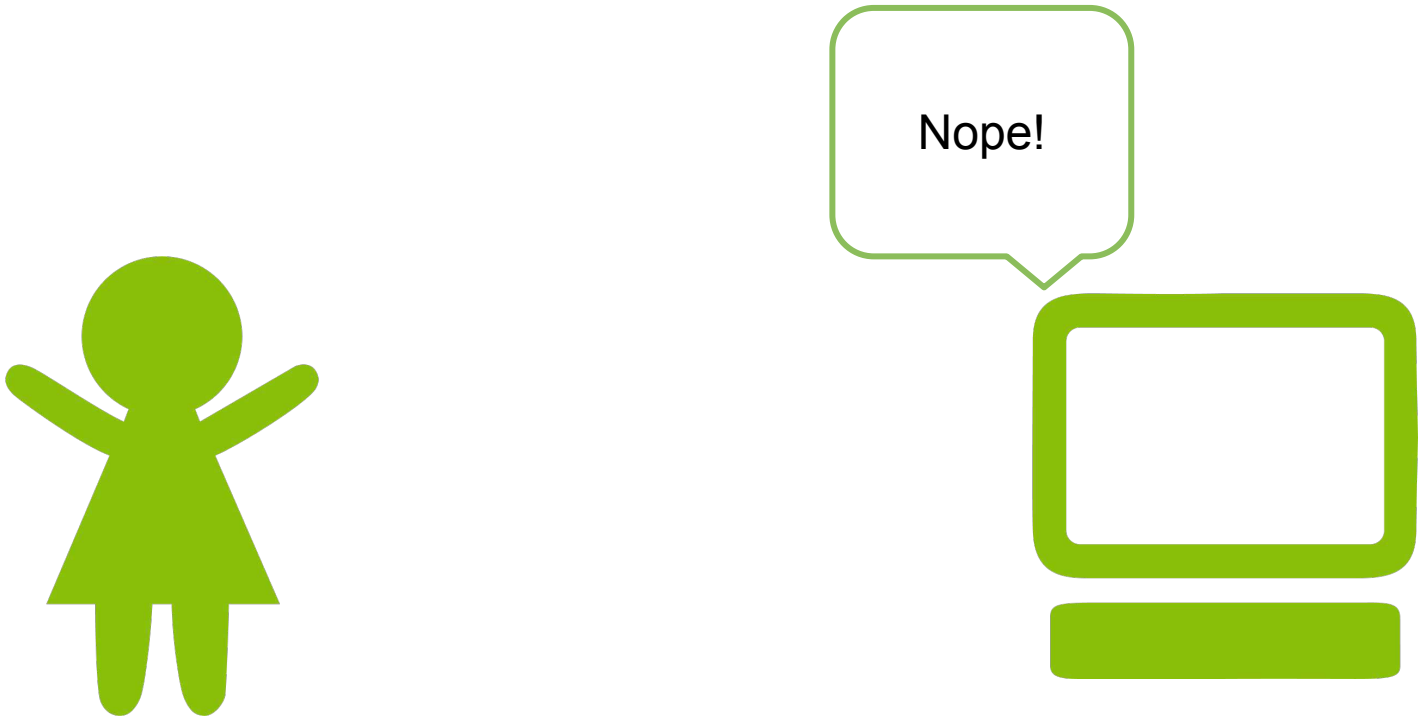
# Looking for Events

We need to keep asking over and over again!



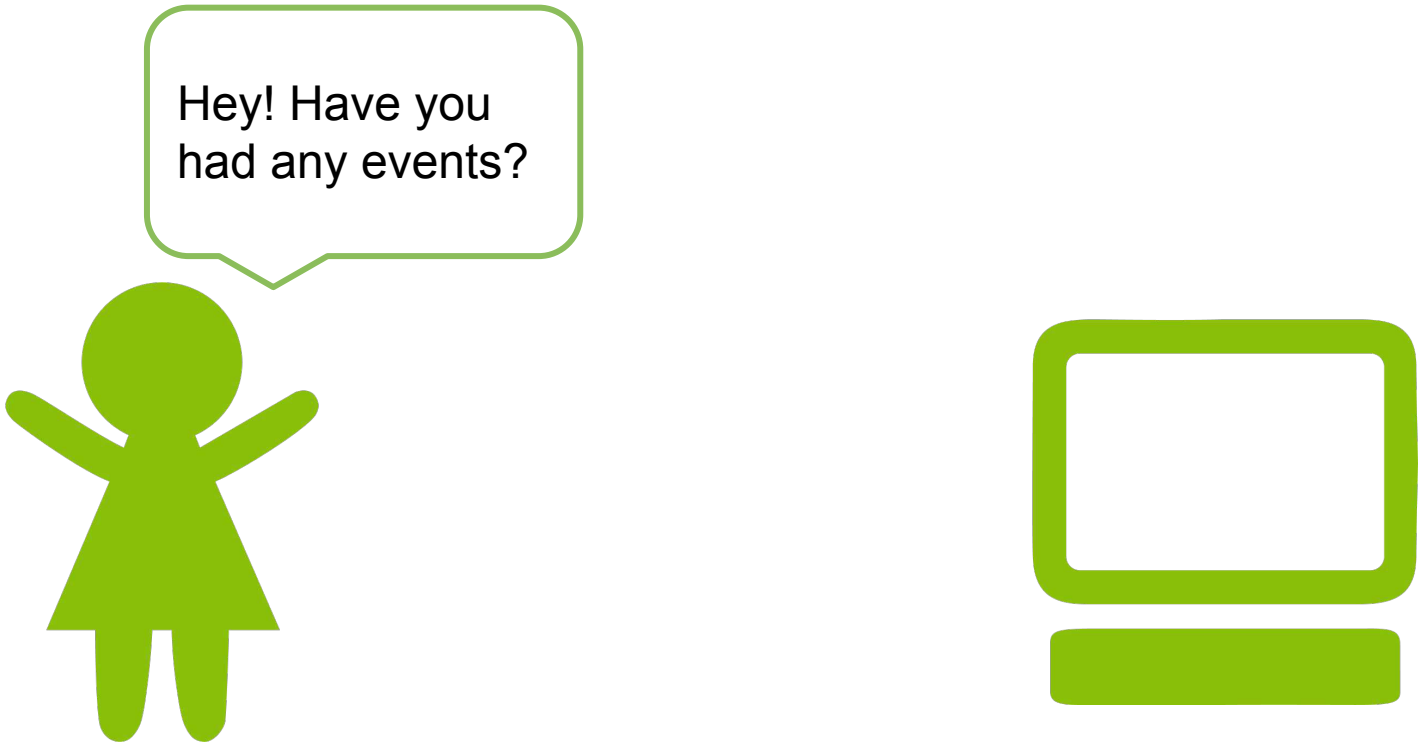
# Looking for Events

We need to keep asking over and over again!



# Looking for Events

We need to keep asking over and over again!

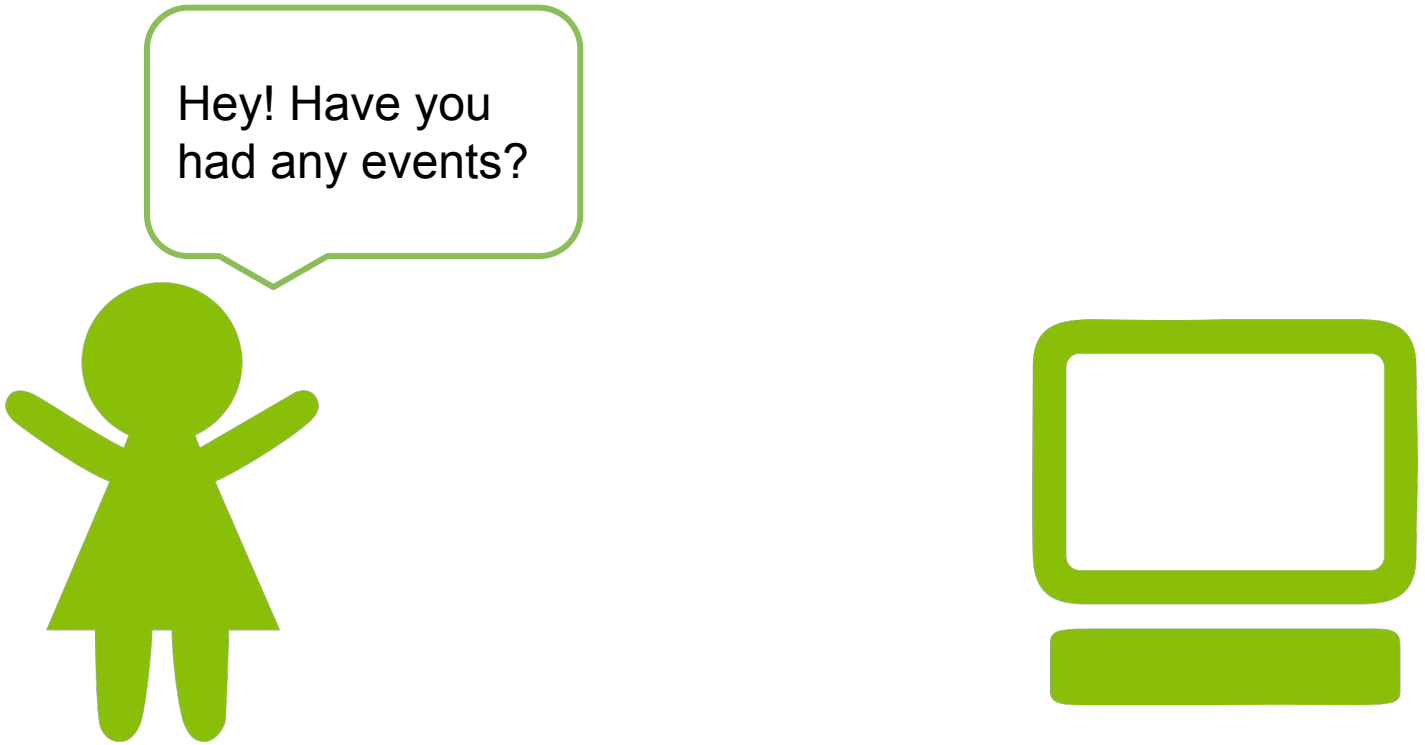


# Looking for Events

We need to keep asking over and over again!

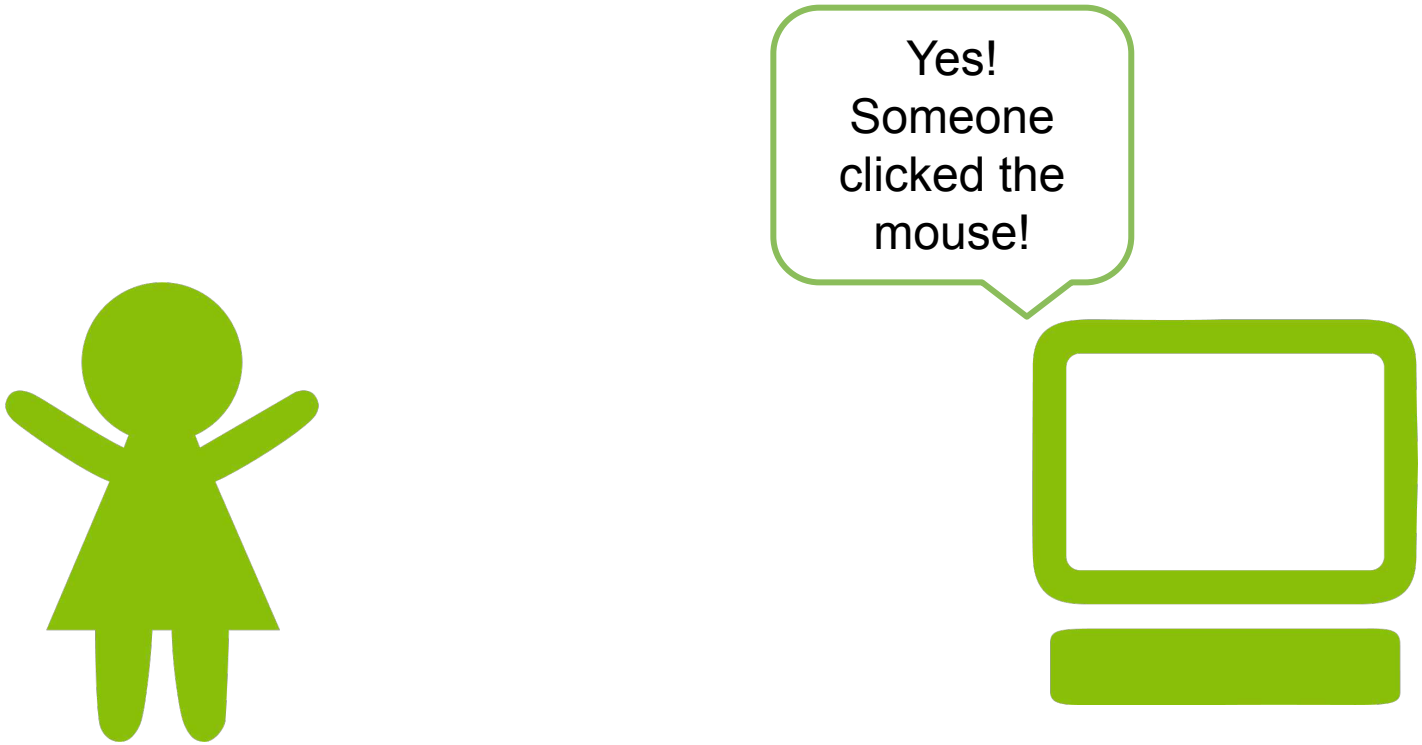
# Looking for Events

We need to keep asking over and over again!



# Looking for Events

We need to keep asking over and over again!



# Loops

We can do something over and over again in our code using Loops! Like this:

```
while True:
  print(“Hello”)
```

What do you think this code does?

# Loops

We can do something over and over again in our code using Loops! Like this:

```
while True:
  print(“Hello”)
```

What do you think this code does?

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

# Loops

We can do something over and over again in our code using Loops! Like this:

```
while True:
  print(“Hello”)
```

It prints “Hello” forever! Let’s have a look at what it’s doing

# Loops

```
while True:
  print("Hello")
```

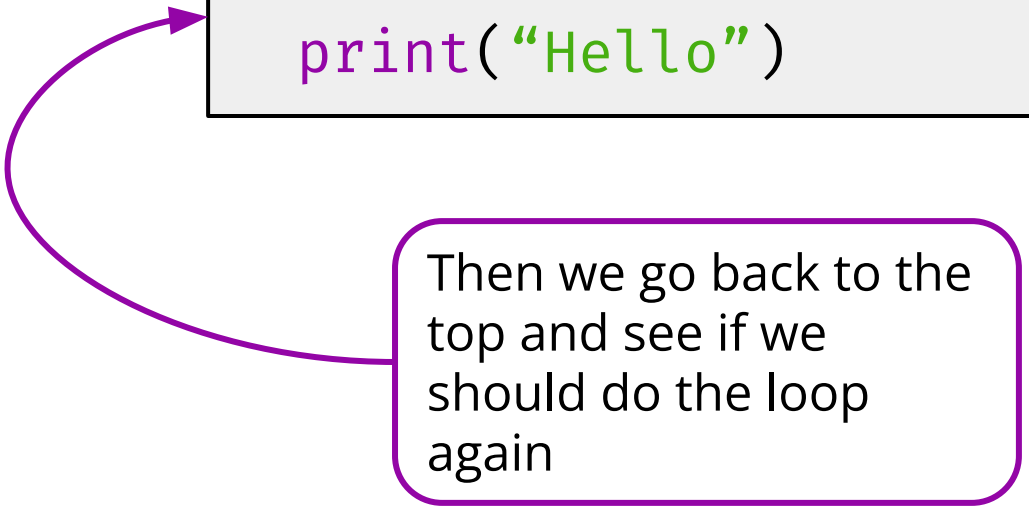
First, it checks to see if it should go into the loop. Because we wrote True here it will **always** go into the loop

# Loops

```
while True:
   print(“Hello”)
```

# Loops

```
while True:
  print("Hello")
```



# Loops

```
while True:
  print("Hello")
```

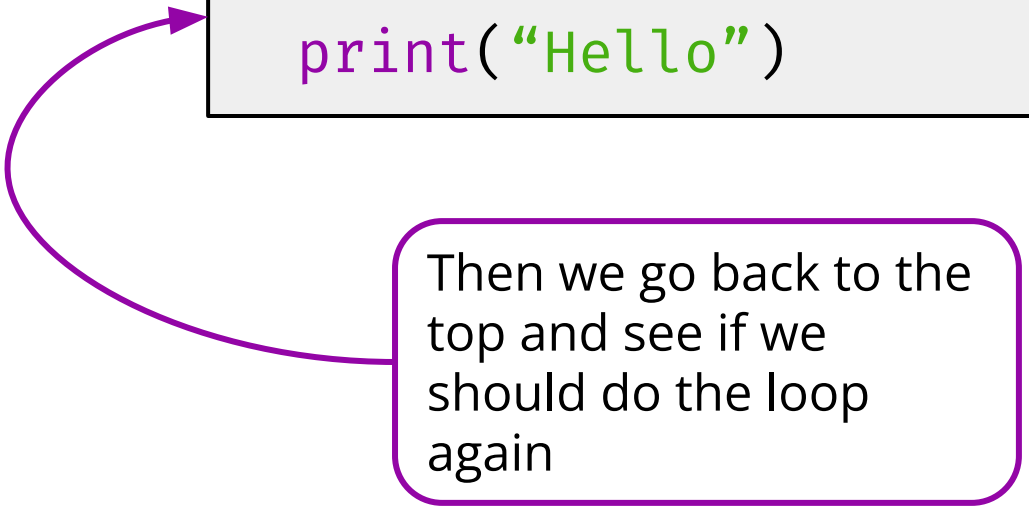
Because we wrote True here it will **always** go into the loop

# Loops

```
while True:
   print(“Hello”)
```

# Loops

```
while True:
  print("Hello")
```



# Loops

This pattern keeps going on and on forever! (or until you quit the program)

```
while True:
  print(“Hello”)
```

# Looking for Events

Now that we understand that we need to keep asking over and over, let's have another look at that code!

# Looking for Events

Now that we understand that we need to keep asking over and over, let's have another look at that code!

```
while True:
  new_event = event.poll()
```

This line checks to see if there is a new event and saves it in a variable called new_event

# Looking for Events

Now that we understand that we need to keep asking over and over, let's have another look at that code!

```
while True:
    new_event = event.poll()
```

This line tells python to do it over and over. It's called a loop!

This line checks to see if there is a new event and saves it in a variable called new_event

# Looking for Events

```
while True:
    new_event = event.poll()
```

First we enter the loop here

# Looking for Events

```
while True:
  new_event = event.poll()
```

Hey! Have you had any events?

Then we ask if there is a new event

# Looking for Events

```
while True:
  new_event = event.poll()
```

Then we ask if there is a new event

Nope!

# Looking for Events

```
while True:
  new_event = event.poll()
```

Then we go back to the top and do it again

# Looking for Events

```
while True:
  new_event = event.poll()
```

Hey! Have you had any events?

Now we're doing this line again!

# Looking for Events

```
while True:
  new_event = event.poll()
```

Nope!

Now we're doing this line again!

# Finding Events

Okay so now we know how to ask for new Events. But what do we do when we find one?

# Finding Events

Okay so now we know how to ask for new Events. But what do we do when we find one?

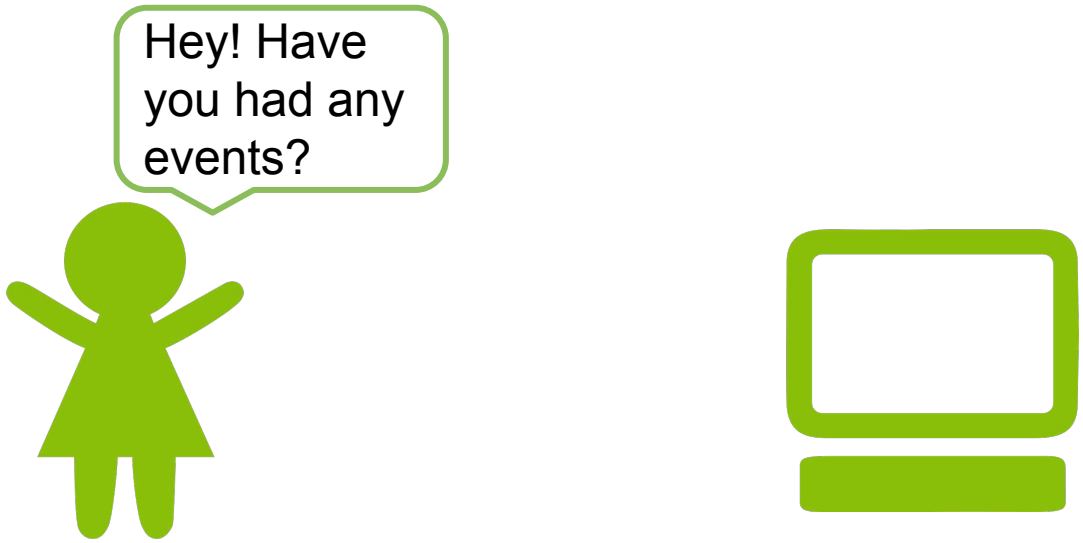
```
while True:
  new_event = event.poll()
  if new_event.type == KEYDOWN:
    print("You pressed a key!")
```

This if statement checks if the type of event was a KEY on the keyboard being pressed DOWN

# Finding Events

```
while True:
    new_event = event.poll()
    if new_event.type == KEYDOWN:
        print("You pressed a key!")
```

First we check if there are any events



# Finding Events

```
while True:
   new_event = event.poll()
   if new_event.type == KEYDOWN:
      print("You pressed a key!")
```

Then we check what type of event it was

Yep! It was a KEYDOWN event

# Finding Events

```
while True:
    new_event = event.poll()
    if new_event.type == KEYDOWN:
        print("You pressed a key!")
```

If it's the event we want then we print this line


# Pressing Keys

But we want to know *which* key they pressed! Not just if they pressed any key on the keyboard!

# Pressing Keys

But we want to know *which* key they pressed! Not just if they pressed any key on the keyboard!

```
while True:
  new_event = event.poll()
  if new_event.type == KEYDOWN and new_event.key == K_SPACE:
    print("You pressed the space key!")
```

This now also checks if they key was the SPACE key

# Finding Events

```
while True:
  new_event = event.poll()
  if new_event.type == KEYDOWN and new_event.key == K_SPACE:
    print("You pressed the space key!")
```

First we check if there are any events


# Finding Events

```
while True:
  new_event = event.poll()
  if new_event.type == KEYDOWN and new_event.key == K_SPACE:
    print("You pressed the space key!")
```

Then we check what type of event it was

And we check what key was pressed

Yep! It was a KEYDOWN event using the SPACE key!

# Finding Events

```
while True:
  new_event = event.poll()
  if new_event.type == KEYDOWN and new_event.key == K_SPACE:
    print("You pressed the space key!")
```

Now we know what the event is we can print


# Project time!

The **key** to doing the next part was all in these slides

**Try to do the next Part**

In the **event** of confusion, the tutors will be around to help!

# Pygame Collisions!

# Pygame Collisions

Our game is looking great so far! But it would be even better if the things in our game could react when something collides with them!

Pygame already knows how to work this out, let's learn how to do it!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

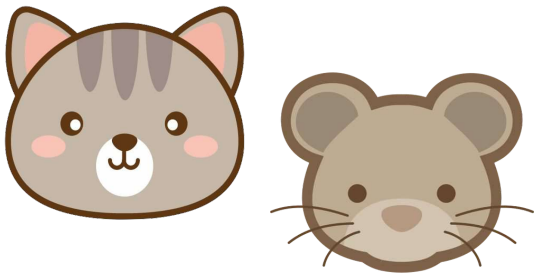
Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!



# Cat & Mouse

Let’s have a look at this example where we have a game where the cat has to catch the mouse!

# Cat & Mouse

Let's have a look at a little bit of the code from this game. We've left out some of it to keep it short

```
while True:
  cat = screen.blit(cat_image, (cat_x, cat_y))
  mouse = screen.blit(mouse_image, (mouse_x, mouse_y))
  display.update()
```

# Cat & Mouse

Let's have a look at a little bit of the code from this game. We've left out some of it to keep it short

```
while True:
  cat = screen.blit(cat_image, (cat_x, cat_y))
  mouse = screen.blit(mouse_image, (mouse_x, mouse_y))
  display.update()
```

What does this code do?

# Cat & Mouse

Let's have a look at a little bit of the code from this game. We've left out some of it to keep it short

```
while True:
  cat = screen.blit(cat_image, (cat_x, cat_y))
  mouse = screen.blit(mouse_image, (mouse_x, mouse_y))
  display.update()
```

What does this code do?

It blits the cat image and the mouse image to the screen and then updates the display!

# Colliderect

We need to be able to tell when our cat collides with our mouse! Let's have a look at the code to do that

```
while True:
  cat = screen.blit(cat_image, (cat_x, cat_y))
  mouse = screen.blit(mouse_image, (mouse_x, mouse_y))
  display.update()

  if cat.colliderect(mouse):
    print("I win!")
```

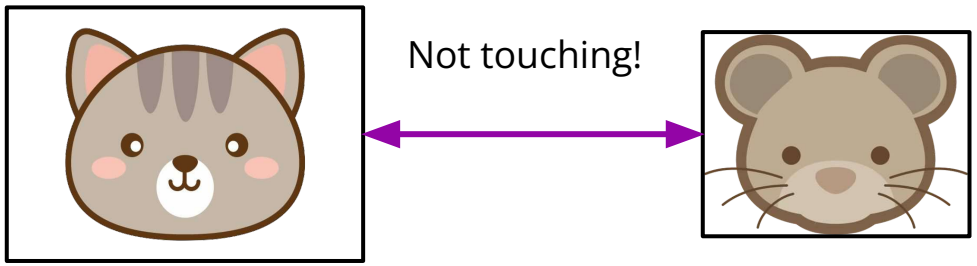
# Colliderect

Let's take a closer look!

```
while True:
  cat = screen.blit(cat_image, (cat_x, cat_y))
  mouse = screen.blit(mouse_image, (mouse_x, mouse_y))
  display.update()

  if cat.colliderect(mouse):
    print("I win!")
```

This if statement checks if the cat's rectangle collides (touches) the mouse's rectangle



# Colliderect

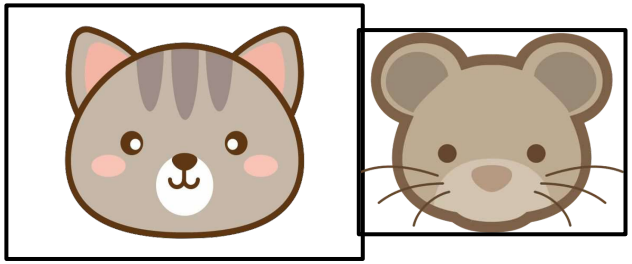
Let's take a closer look!

```
while True:
  cat = screen.blit(cat_image, (cat_x, cat_y))
  mouse = screen.blit(mouse_image, (mouse_x, mouse_y))
  display.update()

  if cat.colliderect(mouse):
    print("I win!")
```

This if statement checks if the cat's rectangle collides (touches) the mouse's rectangle

Touching!



# Project time!

Now we can **collide** our knowledge with the workbook and do the next part!

**Try to do the next Part**

The tutors will be around to help!