

Welcome to GPN

Thank you to our Sponsors!

Platinum Sponsor:



Who are the tutors?

Who are you?

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!

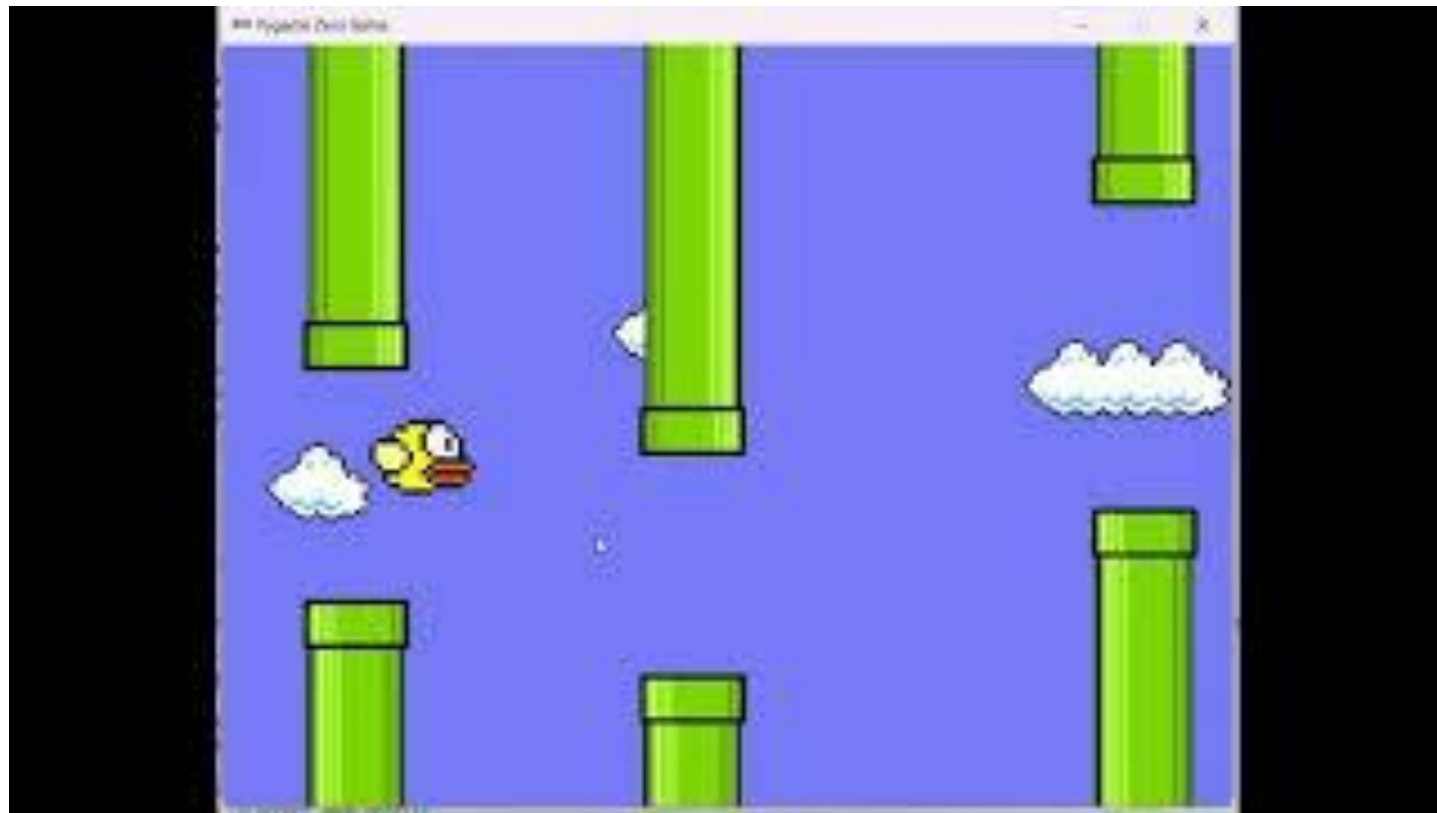
Tell us you're here!

Click on the
Start of Day Survey
and fill it in now!

Today's Project!

Flappy Bird!

What will the game look like?



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!

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.



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!

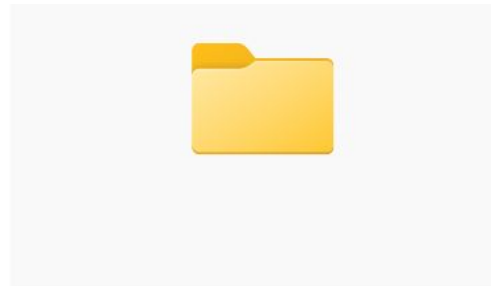


Intro to Python

Let's get coding!

Getting set up

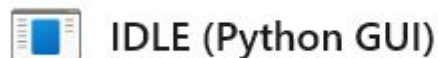
Go to your desktop and open the Flappy bird python



Flappy Bird Python

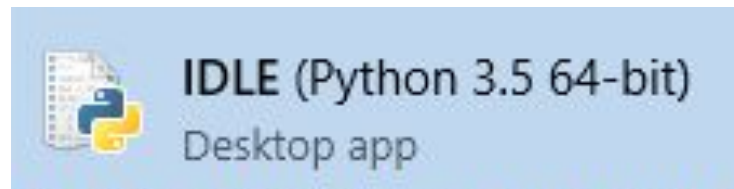
Double click the IDLE(Python GUI).exe file.
(This will download IDLE onto your desktop)

It should look like this

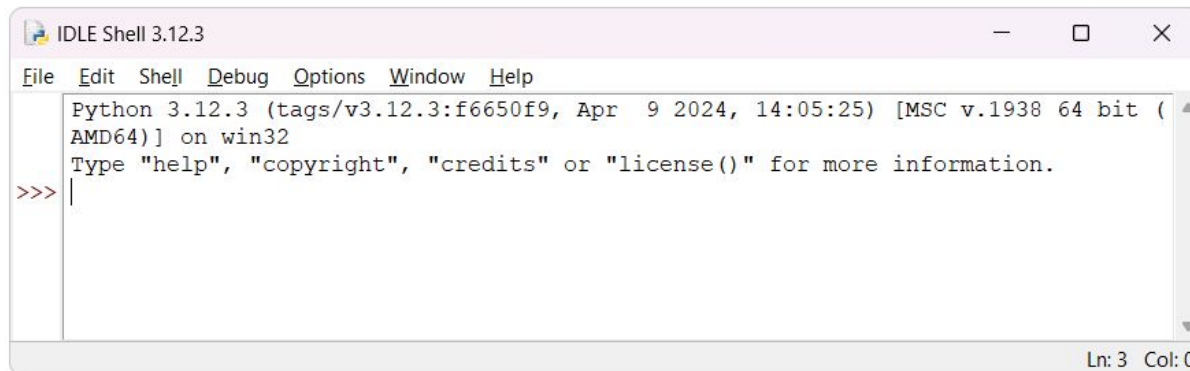


Where do we program? In IDLE

Once it's downloaded open IDLE.



You should get a screen that looks like this!

A screenshot of the IDLE Shell 3.12.3 window. The window title is "IDLE Shell 3.12.3". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area displays the following text: "Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32", "Type 'help', 'copyright', 'credits' or 'license()' for more information.", and a prompt ">>> |". The status bar at the bottom right shows "Ln: 3 Col: 0".

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?

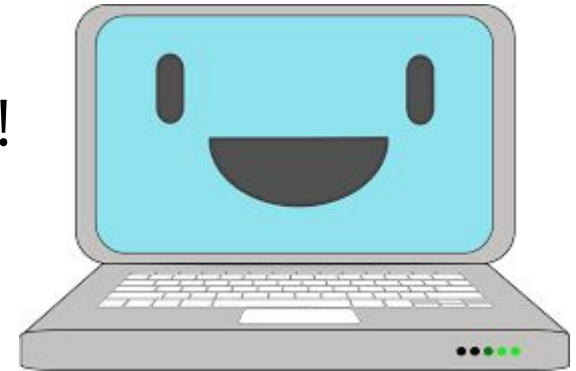
Mistakes are great!

*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!



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

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

*KeyError:
'Hairy Potter'*



We can learn from our mistakes!

Error messages help us fix our mistakes!

We read error messages from bottom to top

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

3. Where that code is

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!")
```

Write some code!!

This is the first bit of code we will do. What do you think it does?

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

Write some code!!

This is the first bit of code we will do. What do you think it does?

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

It prints the words “hello world” onto the screen!

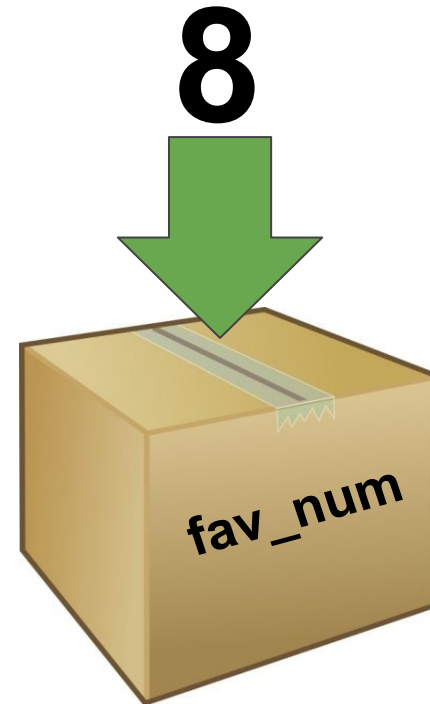
No Storing is Boring!

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

=>

fav_num + 21

=>

fav_num * 2

=>

fav_num / 2

=>

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow 2$$

$$\text{fav_num} + 21 \\ \Rightarrow$$

$$\text{fav_num} * 2 \\ \Rightarrow$$

$$\text{fav_num} / 2 \\ \Rightarrow$$

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow 2$$

$$\text{fav_num} + 21 \\ \Rightarrow 29$$

$$\text{fav_num} * 2 \\ \Rightarrow$$

$$\text{fav_num} / 2 \\ \Rightarrow$$

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow \mathbf{2}$$

$$\text{fav_num} + 21 \\ \Rightarrow \mathbf{29}$$

$$\text{fav_num} * 2 \\ \Rightarrow \mathbf{16}$$

$$\text{fav_num} / 2 \\ \Rightarrow$$

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow \mathbf{2}$$

$$\text{fav_num} + 21 \\ \Rightarrow \mathbf{29}$$

$$\text{fav_num} * 2 \\ \Rightarrow \mathbf{16}$$

$$\text{fav_num} / 2 \\ \Rightarrow \mathbf{4}$$

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow 2$$

$$\text{fav_num} + 21 \\ \Rightarrow 29$$

$$\text{fav_num} * 2 \\ \Rightarrow 16$$

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

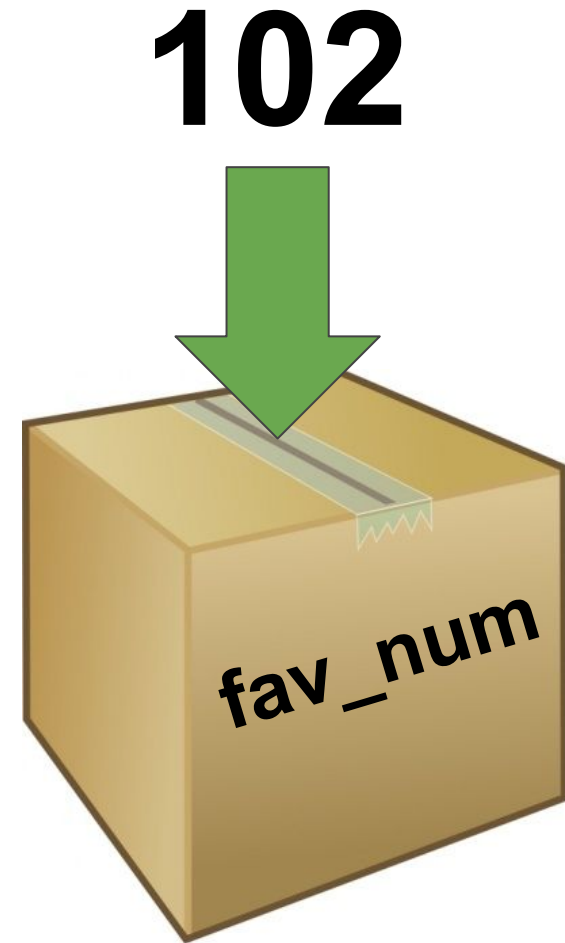


Variables

**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`

`=>`

`fav_num + 21`

`=>`

`fav_num * 2?`

`=>`

`fav_num / 2?`

`=>`

Variables

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



`fav_num - 6`
=> 96

`fav_num + 21`
=>

`fav_num * 2?`
=>

`fav_num / 2?`
=>

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow \mathbf{96}$$

$$\text{fav_num} + 21 \\ \Rightarrow \mathbf{123}$$

$$\text{fav_num} * 2? \\ \Rightarrow$$

$$\text{fav_num} / 2? \\ \Rightarrow$$

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow \mathbf{96}$$

$$\text{fav_num} + 21 \\ \Rightarrow \mathbf{123}$$

$$\text{fav_num} * 2? \\ \Rightarrow \mathbf{204}$$

$$\text{fav_num} / 2? \\ \Rightarrow$$

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow \mathbf{96}$$

$$\text{fav_num} + 21 \\ \Rightarrow \mathbf{123}$$

$$\text{fav_num} * 2? \\ \Rightarrow \mathbf{204}$$

$$\text{fav_num} / 2? \\ \Rightarrow \mathbf{51}$$

No variables VS using variables



4
Changes

| | | |
|--------|---|-----------------|
| 8 - 6 | → | 102 - 6 |
| 8 * 2 | → | 102 * 2 |
| 8 + 21 | → | 102 + 21 |
| 8 / 2 | → | 102 / 2 |



1
Change

| | | |
|--------------|---|----------------------|
| 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?

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)
```

```
My favourite animal is a dog  
My favourite animal is a cat  
My favourite animal is a catdog
```



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?

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)
```

10

9

Delicious



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)
```

Variables

Your turn!

Can you guess what each `print` will do?

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

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

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

Variables

Your turn!

Can you guess what each `print` will do?

```
>>> x = 3
>>> print(x)
3
>>> print(x + x)
6
>>> y = x
>>> print(y)

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


Variables

Your turn!

Can you guess what each `print` will do?

```
>>> x = 3
>>> print(x)
3
>>> print(x + x)
6
>>> y = x
>>> print(y)
3
>>> y = y + 1
>>> print(y)
```

Variables

Your turn!

Can you guess what each `print` will do?

```
>>> x = 3
>>> print(x)
3
>>> print(x + x)
6
>>> y = x
>>> print(y)
3
>>> y = y + 1
>>> print(y)
4
```

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

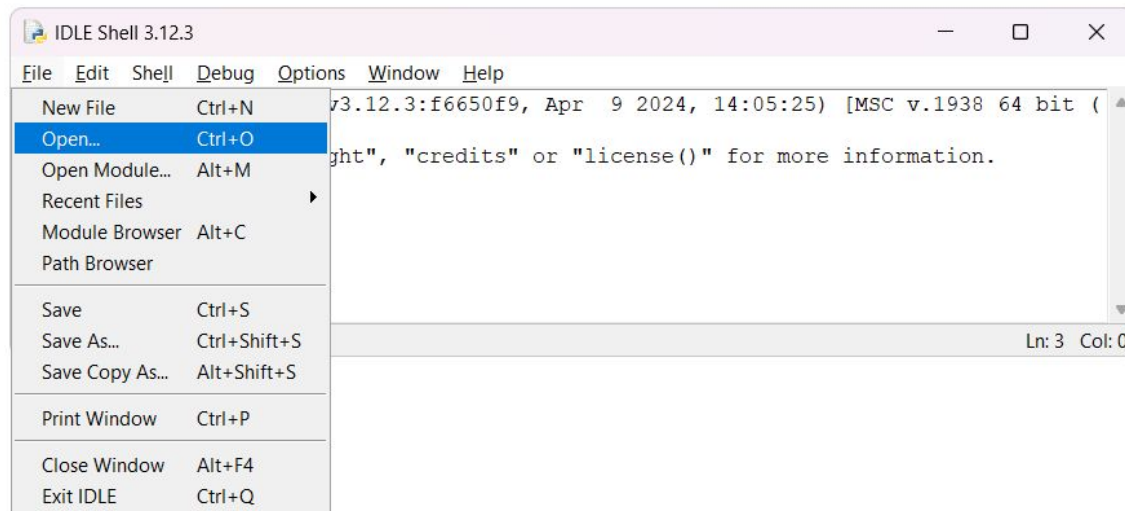
For **True** and **False**

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



Coding in a file!

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



1. Open a file called “flappy_bird.py” (it’s in your folder)
2. Put your `print('hello world')` code in it
3. Run your file using the F5 key

Project time!

You now know all about printing and variables and input!

Let's put what we learnt into our project
Try to do Part 0

The tutors will be around to help!

Intro to PyGame Zero

Making it into a game!



What is Pygame Zero?

We use pygame zero to allow our code to do some cool things.



Pygame Zero Setup

The first thing we need to do to use pygame zero is to write this at the top of your file

```
>>> import pgzrun
```

Pygame Zero Setup

The first thing we need to do to use pygame zero is to write this at the top of your file

```
>>> import pgzrun
```

Now to make sure PyGame Zero runs our code we also need another line at the end of our code

```
>>> pgzrun.go()
```



Some Pygame Zero basics

Here's some of the basics of Pygame Zero that you'll need for your game.

Screen:

Your main screen for the game will be a screen that pops up whenever you run your game. You can create a screen by setting its size using the keywords WIDTH and HEIGHT

1. Try making a 100 x 100 screen and running your file!

The screen should be blank for now

Some Pygame Zero basics

Here's some of the basics of Pygame Zero that you'll need for your game.

Screen:

Your main screen for the game will be a screen that pops up whenever you run your game. You can create a screen by setting its size using the keywords WIDTH and HEIGHT

1. Try making a 100 x 100 screen and running your file!

```
>>> WIDTH = 100
```

```
>>> HEIGHT = 100
```

The screen should be blank for now

Project time!

You now know all about the basics of
Pygame Zero!

Let's put what we learnt into our project
Try to do Part 1

The tutors will be around to help!

PyGame Zero images

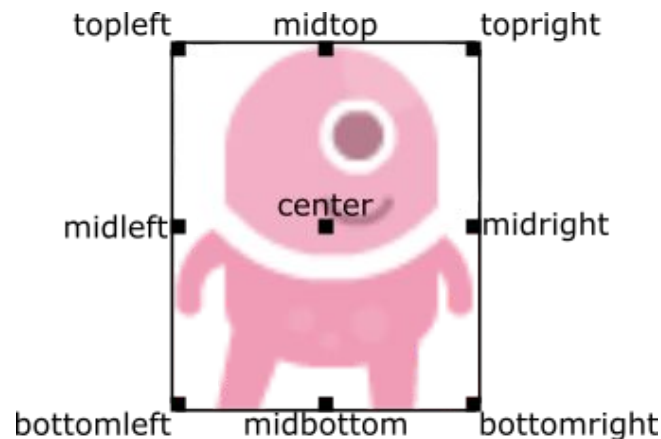
Adding things to our screen!



Images in Pygame zero

Images in Pygame zero are called **Actors**

This is because you can make them move around and do things like actors in a play. Pygame zero stores some information about each of the actors in our game like their position on the screen and what image the actor is.

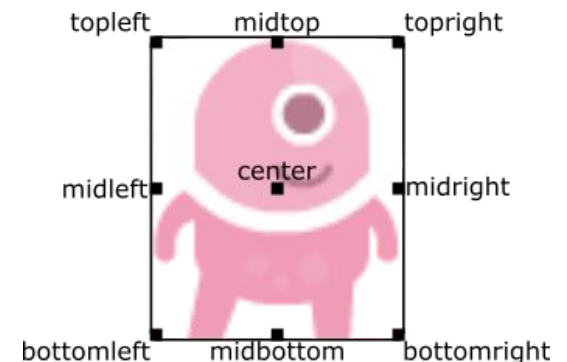


How to make an actor

To make a new actor and tell Pygame zero what image it is you need to write the code:

```
>>> myActor = Actor("myImage")
```

Here the name of our actor is **myActor** and if we need to change anything about it we have to use it's name



How to make an actor

To make a new actor and tell Pygame zero what image it is you need to write the code:

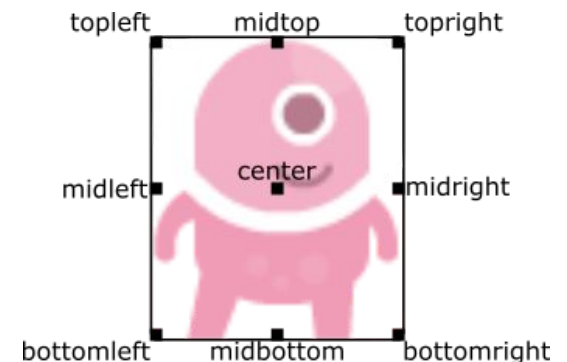
```
>>> myActor = Actor("myImage")
```

Here the name of our actor is **myActor** and if we need to change anything about it we have to use it's name

To set our actor's x and y position you use the code:

```
>>> myActor.x = 50
```

```
>>> myActor.y = 50
```



Some important code

Pygame zero needs some pretty specific things in order to make our game work. To do these there are three main functions:

```
def draw():  
    # This function is to add things to the screen every frame  
  
def update():  
    # This function is to change things every frame  
  
def on_mouse_down():  
    # This function's code runs every time the player clicks their mouse
```

What is a function?

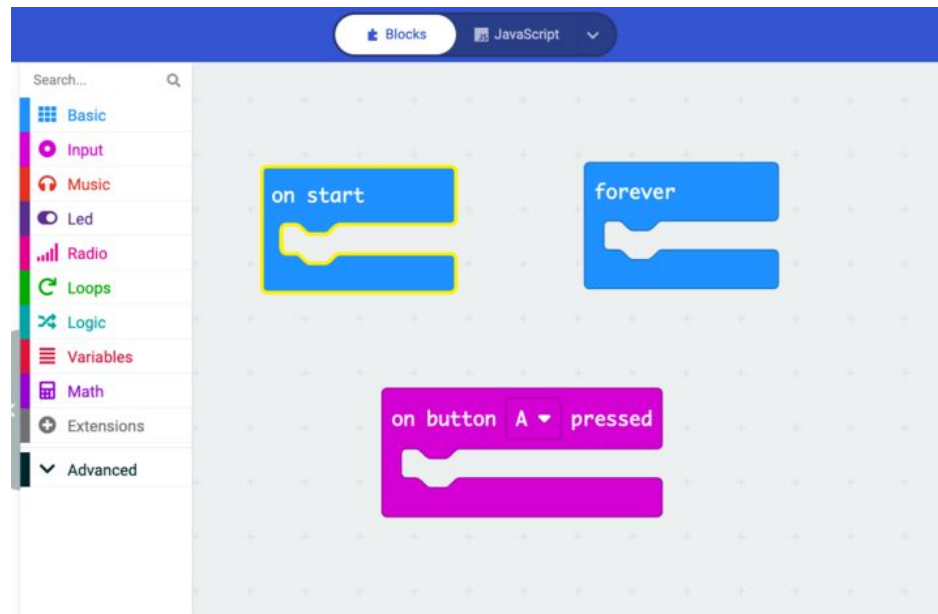
What you need to know about functions:

They are a piece of code that gets run a lot! These functions get run everytime you say their name.



Functions in Blockly

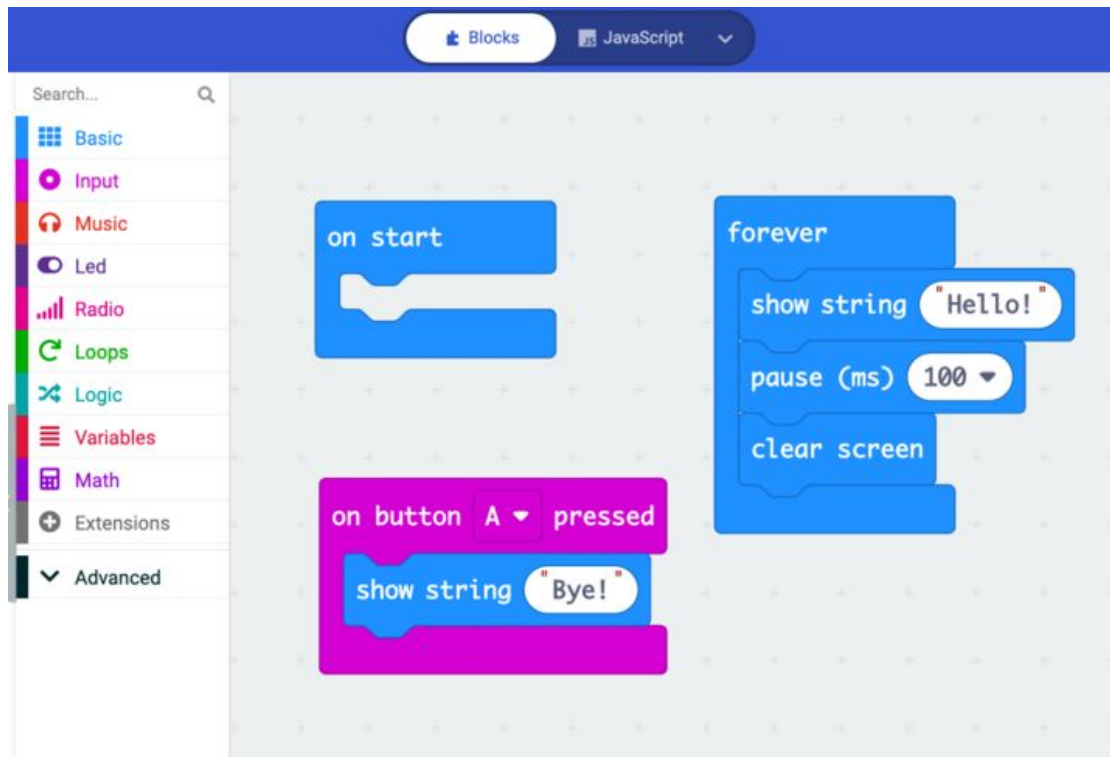
Here are some functions in Blockly - maybe they seem familiar from school



Here any code you put in these boxes will get run every time they do

Functions in Blockly

This is what the functions can look like with code in it...



We can do the same thing with code!

Some important code

Our special Pygame Zero functions are just like the blocks!

```
def draw():  
    # This function is to add things to the screen every frame  
  
def update():  
    # This function is to change things every frame  
  
def on_mouse_down():  
    # This function's code runs every time the player clicks their mouse
```

We'll put our code inside and Pygame Zero will run them to make the game work!

Getting an actor on screen!

The first function we need in Pygame Zero is the `draw()` function. The `draw()` function tells Pygame Zero what things need to appear on screen.

You can use it to “draw” an actor on the screen by using these lines of code:

```
>>> def draw():  
...     myActor.draw()
```

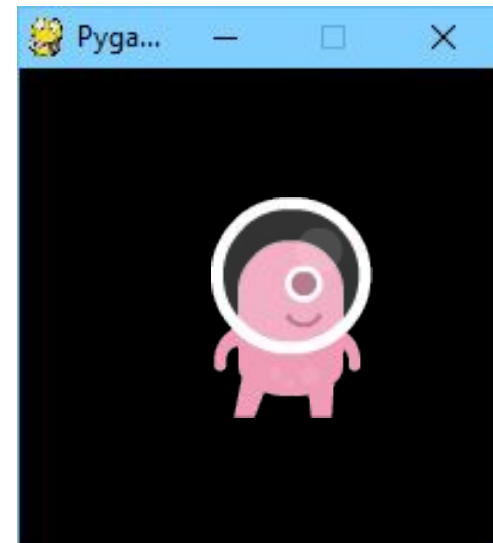


Changing the actor

The `update()` function tells Pygame Zero what things need to change so that it can “animate” the game frame by frame

You can use it to do things like update an actor’s image or x or y coordinates:

```
>>> def update():  
...     myActor.x = myActor.x + 5
```



When the mouse clicks

The `on_mouse_down()` function only runs when the player has clicked. This means that you can make changes to your character when the player clicks their mouse.

You can use it to do things like change an actor's image or x or y coordinates when the player clicks the mouse:

```
>>> def on_mouse_down():  
...     myActor.image("image2")
```



Project time!

You now know all about how to put a character on the screen and how to animate it!

Let's put what we learnt into our project
Try to do Part 2

The tutors will be around to help!

Events and If Statements

Conditions!

Conditions let us make decision.

First we test if the condition is met!

Then maybe we'll do the thing



If it's raining take an umbrella

Yep it's raining

..... take an umbrella

Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

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

`5 < 10`

`3 + 2 == 5`

`5 != 5`

`"Dog" == "dog"`

`"D" in "Dog"`

`"Q" not in "Cat"`

Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

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

| | | |
|-------------------------|-------------------|-------------------------------|
| <code>5 < 10</code> | <code>True</code> | <code>"Dog" == "dog"</code> |
| <code>3 + 2 == 5</code> | | <code>"D" in "Dog"</code> |
| <code>5 != 5</code> | | <code>"Q" not in "Cat"</code> |

Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

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

| | | |
|-------------------------|-------------------|-------------------------------|
| <code>5 < 10</code> | <code>True</code> | <code>"Dog" == "dog"</code> |
| <code>3 + 2 == 5</code> | <code>True</code> | <code>"D" in "Dog"</code> |
| <code>5 != 5</code> | | <code>"Q" not in "Cat"</code> |

Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

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

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

Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

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

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

Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

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

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

Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

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

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

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?

```
>>>
```



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?

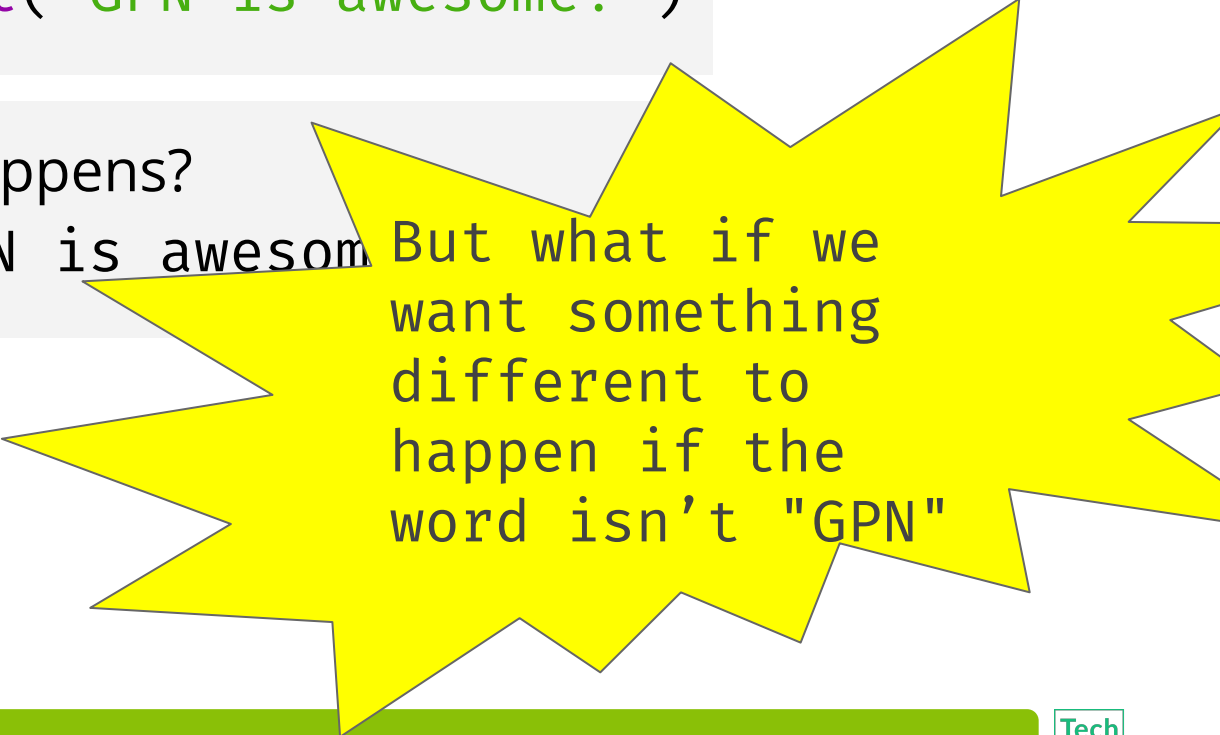
```
>>> GPN is awesome!
```

If statements

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

What happens?

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



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
statements
means we can
give specific
instructions for
other scenarios

```
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
statements
means we can
give specific
instructions for
other scenarios

```
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**!

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

The tutors will be around to help!