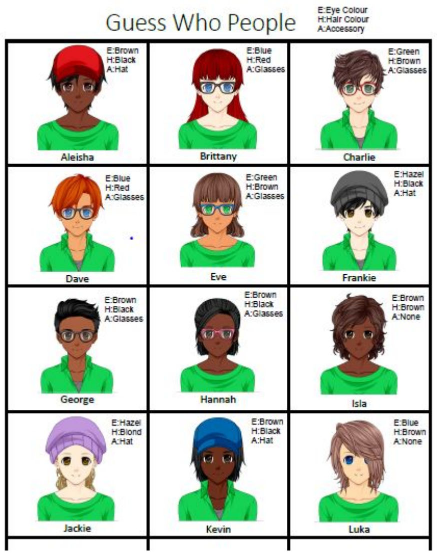
Guess Who!

Welcome to the Labs



Welcome to the Labs

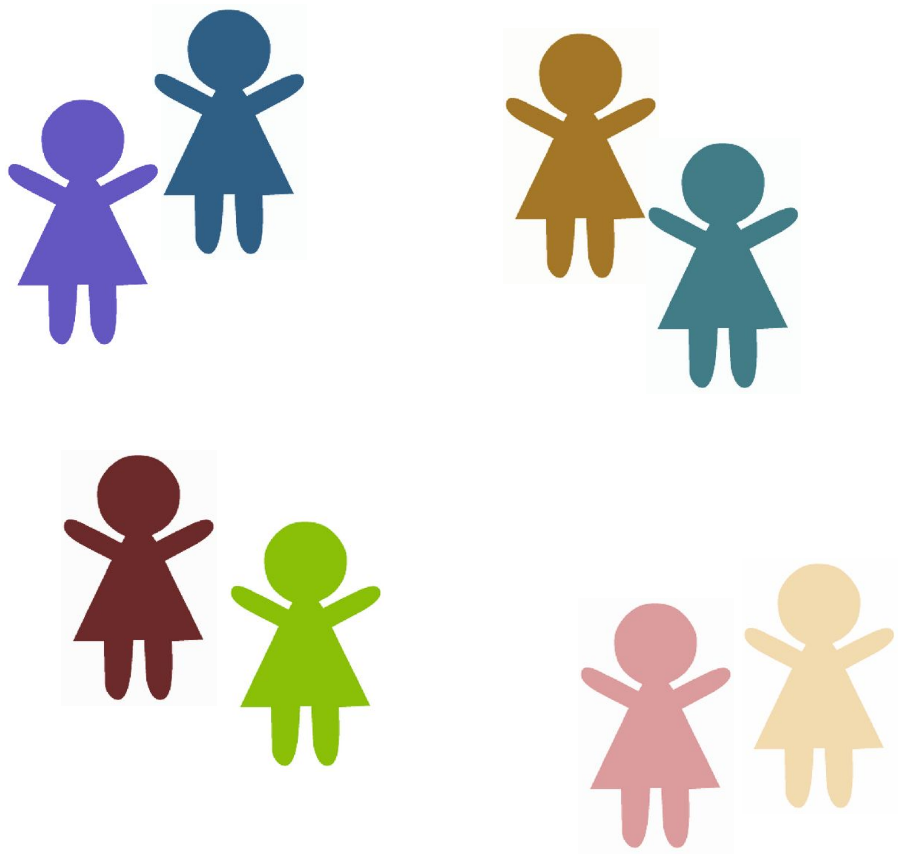
Guess Who!

Who are the tutors?

Who are you?

Introduce your partner

1. Find a partner (someone you’ve never met before)
2. Find out:
   a. Their name
   b. What (school) year they are in
   c. A fun fact about them!
3. Introduce them to the rest of the group!



Log on

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!

Guess Who?

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.

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

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.

★ BONUS 4.3: Do some extra!

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

Files

Girls' Programming Network
School of Information Technologies
University of Sydney

Filing it away!

What if we want to play Guess Who! But with different characters?

We'd have to change our code!!

It would be better if we could keep all our people 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!

```
my_file = open("test.txt")
```

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

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')
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

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

```
>>> print(my_stirng)
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!
```

You can also read in one line at a time

You can use a for loop to only get 1 line at a time!

```
my_file = open('haiku.txt')
for line in f:
   print(line)
```

```
Wanna go outside.

Oh NO! Help! I got outside!

Let me back inside!
```

Why is there an extra blank line each time?

Chomping off the newline

The newline character is represented by '\n':

```
print('Hello\nWorld')
Hello
World
```

We can remove it from the lines we read with .strip()

```
x = 'abc\n'
x.strip()
'abc'
```

x.strip() is safe as lines without newlines will be unaffected

Reading and stripping!

```
for line in open('haiku.txt'):
    line = line.strip()
    print(line)
```

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

No extra lines!

Using with!

This is a special trick for opening files!

```
with open(“words.txt”) as f:
  for line in f:
    print(line.strip())
```

It automatically closes your file for you!

It’s good when you are writing files in python!

Object Oriented Programming

Objects and Attributes

Outline

1. Objects

2. Classes

Thinking about objects

Thinking about objects

weight	height
level	type
attack	defense
speed	moves

Thinking about objects

weight	height
level	type
attack	defense
speed	moves

```
.level          .moves          .attack_stat
.type           .weight         .defense_stat
```

A new structure

```
class Pokemon:
    max_moves = 4
```

```
    def __init__(self):
        self.attack = 120
```

```
pikachu = Pokemon()
pikachu.attack
```

```
>>> 120
```

INSTANCE NAME	pikachu
attack	120

__init__

```
class Pokemon:
    max_moves = 4

    def __init__(self):
        self.attack = 120

pikachu = Pokemon()
print(pikachu.attack)
```

```
>>> 120
```

INSTANCE NAME	pikachu
attack	120

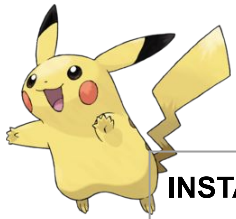
__init__

```
class Pokemon:
    max_moves = 4

    def __init__(self, new_attack, new_defense):
        self.attack = new_attack
        self.defense = new_defense
```

```
pikachu = Pokemon(120, 345)
print(pikachu.attack)
print(pikachu.defense)
```

```
>>> 120
>>> 345
```



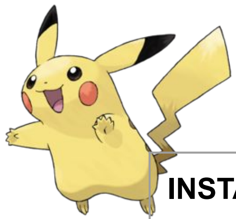
INSTANCE NAME	pikachu
attack	120
defense	345

More Pokemon!

```
class Pokemon:
    max_moves = 4

    def __init__(self, pokename,
            new_attack, new_defense):
        self.pokename = pokename
        self.attack = new_attack
        self.defense = new_defense

pok1= Pokemon(‘pikachu’,120, 345)
pok2= Pokemon(‘squirtle’,156, 125)
```



INSTANCE NAME	pok1
pokename	pikachu
attack	120
defense	345

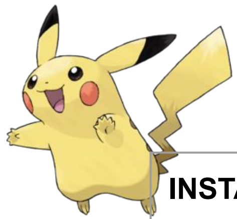
INSTANCE NAME	pok2
pokename	squirtle
attack	156
defense	125

Static class variables

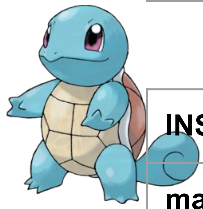
```python
class Pokemon:
    max_moves = 4
```

```python
    def __init__(self, pokename, ...
        self.pokename = pokename
        self.attack = new_attack
        self.defense = new_defense
```

```python
pok1= Pokemon(‘pikachu’, 120, 345)
pok2= Pokemon(‘squirtle’, 156, 125)
print(pok1.max_moves)
print(pok2.max_moves)
>>> 4
>>> 4
```



INSTANCE NAME	pok1
max_moves	4
pokename	pikachu
attack	120
defense	345



INSTANCE NAME	pok2
max_moves	4
pokename	squirtle
attack	156
defense	125

Instances can be different

```
class Pokemon:
    max_moves = 4
```

```
    def __init__(self, attack, defense, moves):
        self.attack = attack
        self.defense = defense
        self.moves = moves
```

```
pikachu = Pokemon()
pikachu.attack_stat
>>> 140
```

A new structure

Creating a class:

```
>>> class Pokemon:
...       attack_stat = 140
...
>>> pikachu = Pokemon()
>>> pikachu.attack_stat
140
```

Pokemon is a class

`pikachu` is an *instance* of the class `Pokemon`

Add whatever data you like!

```
>>> class Pokemon:
...         attack_stat = 140
...         moves = [
               ’thunderbolt’, ’tail whip’]
```

```
...
>>> pikachu = Pokemon()
>>> pikachu.attack_stat
140
>>> pikachu.moves
[‘thunderbolt’, ‘tail whip’]
```

Instances can be different

```
>>> class Pokemon:
...     def __init__(self, attack, defense, moves):
...         self.attack = attack
...         self.defense = defense
...          self.moves = moves
...
>>> pikachu = Pokemon(140, 136, [‘thunderbolt’, ‘tail
whip’])
>>> pikachu.attack_stat
140
>>> pikachu.moves
[‘thunderbolt’, ‘tail whip’]
```

Instances can be different

```
>>> class Pokemon:
...     def __init__(self, attack, defense, moves):
...         self.attack_stat = attack
...         self.defense_stat = defense
...          self.moves = moves
...
>>> squirtle = Pokemon(96, 160, [‘bubble’, ‘tackle’])
>>> squirtle.attack_stat
96
>>> squirtle.moves
[‘bubble’, ‘tackle’]
```

Now it's your turn!

- Create a Pokemon class (make a new Pokemon if you like!)
- In the init function, add properties for its
  - type,
  - level,
  - attack stat,
  - HP,
  - moves (e.g. {‘splash’: 0}),
  - and anything else you like.
- Now, try creating a few different types of Pokemon.

Object Oriented Programming
Methods

Outline

1. Methods

2. Modules

There's a method for this madness!

```
level_up()
attack()
run_away()
```

```
Level_up(): increases the level of the pokemon by 1.
run_away(): Prints "<name> ran away!"
attack():  Selects a random move, prints "<name> used <move>!"
```

Class methods

```
>>> class Pokemon:
...
...     def __init__(self, level, type):
...         self.level = level
...         self.type = type

...     def level_up(self):
...         self.level += 1

...     def set_type(self, new_type):
...         self.type = new_type

>>> p = Pokemon(50, ‘normal’)
>>> p.type
‘Normal’
>>> p.set_type(‘Fairy’)
>>> p.type
‘Fairy’
```

Class methods

```
>>> class Pokemon:
...
...     def __init__(self, level, type):
...         self.level = level
...         self.type = type

...     def level_up(self):
...         self.level += 1

...     def set_type(self, new_type):
...         self.type = new_type

>>> p = Pokemon(50, ‘normal’)
>>> p.level
45
>>> p.level_up()
>>> p.level
46
```

Now it's your turn!

- Modify your Pokemon class to have some new methods.
- In the init function, add methods for:
  - levelling up,
  - attacking,
  - defending,
  - using an item (might need another class!),
  - and anything else you like.
- Now, try instantiating a few different types of Pokemon.
- See if you can make them interact!

Syntax cheatsheet

```
class MyClassName:
  staticVariable = someValueForEveryInstance
  def __init__(self, param1, param2...):
     # Set the instance variables
     self.myParam1 = param1
     self.someOtherValue = param2
  def someFunc(self, otherParam1, otherParam2...):
     # Do stuff here
     # You can even return values if you like!
```

Syntax cheatsheet

```
# Access static variables
MyClassName.staticVariable


# Create new instance of a class
mine = MyClassName(param1, param2...)


# Access an instance variable or function
mine.myParam1
mine.someFunc(otherParam1, otherParam2...)


# Store values from functions that return something
someValue = mine.someFunc(otherParam1, otherParam2...)
```