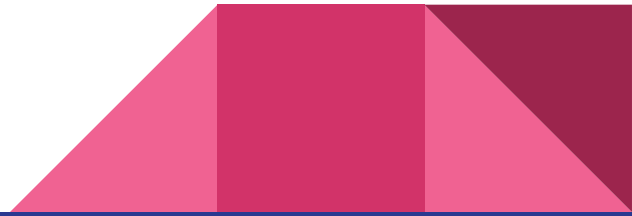
# #CodeYork

Session 2

# Functions

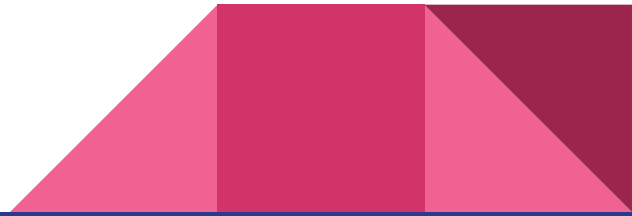
# Motivation

- Say you want to perform a specific action in your code many times throughout
  - Can’t put in a while loop, since it’s not always in the same place
  - Awful to copy and paste code that does this action everywhere, messy
    - What if you want to modify it later?
- We want to name an action and just tell Python ‘do that action’ whenever we want to



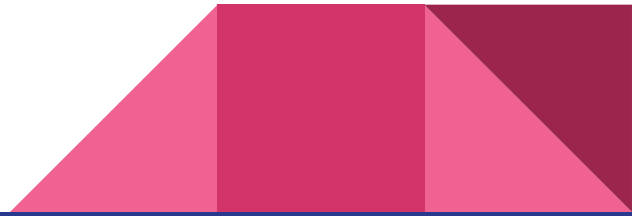
# What’s a function?

- The ‘verbs’ of a programming language
- Can define an ‘action’ in code and ‘perform’ it at any time
  - Proper terms are ‘function’ and ‘call’
- In english, can define what ‘speak’ means and perform it after, once everyone knows what it means
- In python, can define what ‘do_thing()’ means and perform it after, once Python knows what it means



# Parameters and Arguments

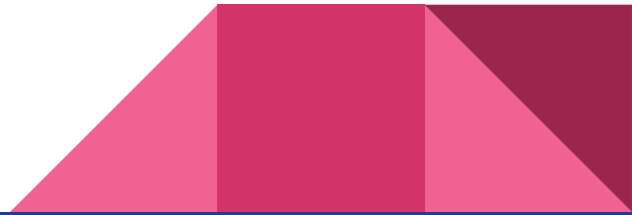
- Sometimes our actions need to know about the world they're in
  - eg. 'eating' needs us to know who's eating and what they're eating
- Functions can have parameters
- Parameters are the inputs a particular function wants in order to run
  - eg. print() wants a string to display, has one parameter
- Arguments are the specific values we end up giving to the function
  - This is why functions have parentheses after their names!
  - eg. print("Hello york!") -> "Hello york!" is the argument



# Lists and Iteration

# List Cheat Sheet

- List definition
  - ls = [1, 2, ‘hello’, 3.4, True]
- Indexing
  - ls[0] -> 1rst item (1), ls[2] -> 3rd item (‘hello’)
  - ls[-3] -> 3rd item from end (‘hello’)
  - “ls[1] = 4” changes 2nd item in ls to 4
  - ls[1: 4] -> sublist from ls[1] to ls[3], ie. [2, ‘hello’, 3.4]
- Common functions
  - len(ls) -> length of list
  - max(ls), min(ls) -> max and min of list
  - ls.append(5) -> appends 5 to end of list
  - ls.index(3.4) -> index of first element equal to 3.4 (ie. 3)



# Iteration

- Use ‘while’ loops when you want to repeat something until a condition stops being true
- Use ‘for’ loops to do something for every element in an “iterable”
- In Python 3, we have the “range” function which returns something that behaves like a list of numbers

```
>>> for x in range(0, 3):
        print(x)

0
1
2
```

# Iteration Examples

```
>>> for el in [1, 2, 3, 4]:
        print(el)

1
2
3
4
```

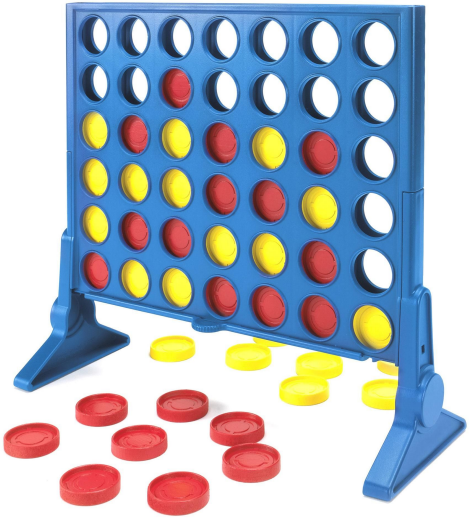
```
>>> for char in "hello":
        print(char)

h
e
l
l
o
```

# Connect Four

# Connect Four

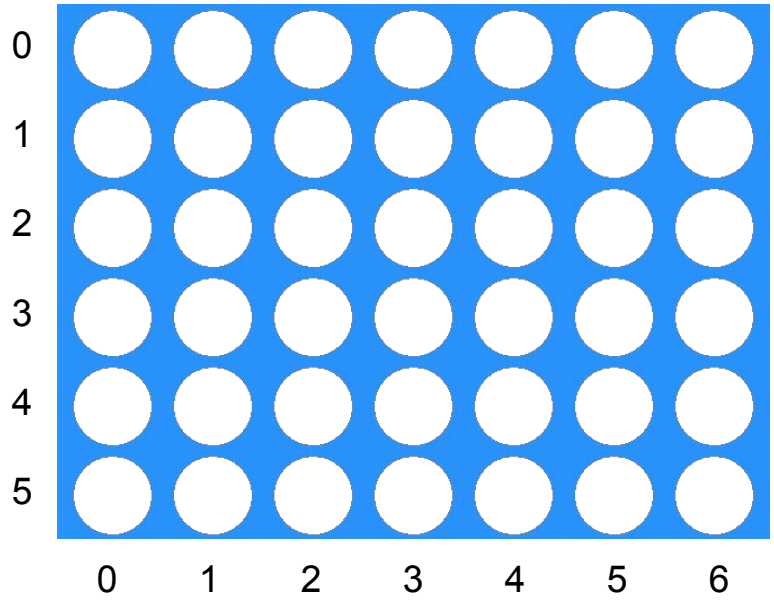
- A two-player connection game
- 6 squares high
- 7 squares wide
- First to get “4 in a row” wins



# Representing the Game Board

- We shall be using 2D arrays (lists of lists)
- Not as hard as you might think!
- Accessing “board[0]” will give you the 1st row (which is a list)
- Accessing “board[0][0] will thus give you the upper left element
- Accessing “board[0][6] will thus give you the upper right element

# Indexing the Game Board



# Using The Client

A link to download the client is available on the module webpage.

app.py - /Users/graham/GitHub/CYorkClient/app.py (3.5.2)

```
import client
import random
import sys


def main(args):
    """
    Instantiate app.
    """
    if input("AI player? (y/n) ") == 'y':
        movegen = client.MoveGeneratorAI()
    else:
        movegen = client.MoveGeneratorPlayer()
    app = client.Client(movegen)
    app.run()


if __name__ == "__main__":
    main(sys.argv)
```

```
def my_move(state):
    """
    Put your Connect 4 decision logic in here.

    Args:
        state (list): List of rows in game board, ie. the 'state' of the game.
        Element of a row is True if your token, False if opponent's, and None if

    Returns:
        int: Integer index of column to drop next token into.
    """
    # An example program to choose a random column for your move.
    num_columns = len(state[0])
    chosen_column = random.randint(0, num_columns)

    return chosen_column
```

Ln: 1 Col: 0

# Go write code!

# Thanks!

Contact us:

gjc510@york.ac.uk
jr1161@york.ac.uk

https://york.gjcampbell.co.uk/