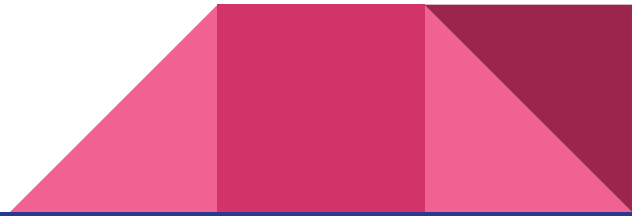
# #CodeYork

Session 3

# Motivation

- We've seen 'iterative' solutions before
  - Using loops to iterate over many sub-parts of a problem, eg. loop through a list to find its largest value or to sort the list
- Iterative solutions are generally not elegant (see most sorting algorithms)
  - Code can have elegance too - you'll find this out!
- Instead, we could define how to solve both the simplest case and how to reduce the difficulty of solving a problem, bit by bit



# What’s Recursion?

- Recursion is defining a solution in terms of itself
  - Our problem needs to have some cases where the solution is immediately obvious, and others where the problem can be simplified to a smaller one
  - Keep simplifying our problem again and again until we get a case with an obvious solution
- The simple cases are the ‘base cases’
- The other cases, where the problem must be simplified, are the ‘recursive cases’

# Recursion Example

- Recursion can allow us to do elegant list processing:

```
def print_in_order(lst):
    if len(lst) > 0:
        print(lst[0])
        print_in_order(lst[1:])

def print_in_reverse(lst):
    if len(lst) > 0:
        print_in_reverse(lst[1:])
        print(lst[0])
```

# Palindromes

- A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward.
  - eg. “madam” or “kayak”
- There will be an exercise to write a function to check if a string is a palindrome. We’ll assume the input is a single lowercase word.

# Factorials

- A number's factorial is all the non-negative integers up to that one multiplied together
  - Notation is an exclamation mark, eg. 10!
- 10! = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 3628800

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n \cdot (n-1)! & \text{if } n>0 \end{cases}$$

“To understand recursion, one must first understand recursion.”

- Stephen Hawking

# Thanks!

Contact us:

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

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