Recursive Functions

Dr Ian Cornelius

Hello

Hello (1)

Learning Outcomes

1. Understand the concept of recursion and recursive functions
2. Practice and implement recursive functions

Recursive Functions

Recursive Functions (1)

- Recursion is concerned with a function calling itself, either:
  - **directly**: the function contains a call to itself
  - **indirectly**: the function contains a call to another function, which in turn calls the recursive function
- There must be some sort of condition to ensure this process can be stopped
  - otherwise, known as *base case*

```
def recursion_fun():
    ...
    recursion_fun()
    ...

recursion_fun()
```

Recursive Functions (2)

Infinite Recursion

- When a function calls itself, then the called execution will also call a further execution and so on
  - this would result in an infinite number of calls made, known as *infinite recursion*
- To avoid this, the recursive function must be carefully constructed
- Ensure at some stage that the function can terminate without calling itself

Recursive Functions (3)

Why use Recursion?

- Recursive functions can make code look clean and elegant
- Generating sequences is easier with recursion instead of nested iteration
- Complex tasks can be broken down into simpler sub-problems

But...

- Following the logic of a recursive function can be difficult
- Recursive calls are inefficient and can take up a lot of memory and time
- Recursive functions can be difficult to debug

Recursive Functions (4)

Example: Factorial

- A simple example of recursion is the *factorial* function, $f(n) = n!$
- When called with a positive integer, it will call itself by decreasing the number
- Each function will multiply the number with the factorial of the number below it, until it is equal to one

```
def factorial(x):
    if x == 1:
        return 1
    return x * factorial(x - 1)
```

```
factorial(5) -> 120
```

Recursive Functions (5)

Example: String Reversal

- Another example of a recursive function is reversing a string
- When met with an empty string, the process is terminated
- However, if the list contains multiple elements, then a pattern needs to be found
  - the first character of the string is concatenated to the end of the remaining characters
  - this is repeated until the end of the string, i.e. when it is empty

```
def reverse_string(s):
    if s == "":
        return s
    return reverse_string(s[1:]) + s[0]
```

```
reverse_string('hello') -> olleh
```

Goodbye

Goodbye (1)

Questions and Support

- Questions? Post them on the **Community Page** on Aula
- Additional Support? Visit the Module Support Page
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk