- Function evaluation
- Pure functions
- Anonymous functions a.k.a. lambdas
- Lambda example
- Lambdas and parameters

FP is a style of programming characterised by...
- Treating computation as the evaluation of functions
- Use of higher-order functions and/or recursion
- Immutable (read-only) state
- Lazy evaluation

Why use FP?
- Very amenable to multi-threading
- Share complex algorithms across multiple threads, to maximise concurrency and increase performance

Any disadvantages?
- Quite a steep learning curve
- Not suitable for every problem

Functions depend only on their inputs, and not on other program state
For example, consider the following function...

```
def cube(x):
  return x * x * x
```

Ex1

- It always gives the same answer for the same input (so it has predictable behaviour - you can reason about its operation)
- It has no local state, side-effects, or changes to any other program state (so it can be safely executed by multiple threads)

A lambda expression is a 1-line inline expression
- Like an anonymous function

To define a lambda expression:
- Use the lambda keyword...
- Followed by the argument list...
- Followed by a colon...
- Followed by a 1-line inline expression

```
my_lambda = lambda arg1, arg2, … argn  :  inline_expression
```

To invoke a lambda expression
- Same syntax as a regular function call

```
my_lambda(argvalue1, argvalue2, …, argvaluen)
```

Ex2 Ex3

- Passing a lambda to a function
- Returning a lambda from a function
- Closures

Higher-order functions can use other functions as arguments and return values
- You can pass a function as a parameter into another function
- You can return a function from a function

Let's explore both these techniques
- We'll use lambdas to represent the function parameters/returns

You can pass a lambda as a parameter into a function
- Allows you to write very generic functions

Example
- The apply() function applies the lambda that you pass in

Ex4

You can return a lambda from a function...

Consider this simple concat() function
- Concatenates its two parameters in the order specified

Now consider the flip() function
- Takes a binary operation
- Returns a lambda that performs the operation with args flipped

Ex5

A closure is a function whose behaviour depends on variables declared outside the scope in which it is then used
- This is often used when returning functions/lambdas
- The returned function/lambda remembers the original state in the enclosing function

Ex6

fib returns a function that calculates Fibonacci numbers, returns the next one each time called.

Note 1:
- nonlocal keyword lets you access a variable in external scope

Note 2:
- tup is a tuple, and you access its members using [0] and [1]

Ex7

- Recursion
- Tail recursion
- Reduction
- Partial functions

Recursion is commonly used instead of looping
- It avoids the mutable state associated with loop counters

Ex8

Tail recursion is where the very last thing you do in a function is call yourself
- The function calls can theoretically be executed in a simple loop

Ex9

The functools module has several useful utility functions for functional programming
- E.g. reduce(), which reduces the elements in a collection to a single result

Ex10

The functools module also allows you to create partial functions, i.e. functions with one or more args already filled in.
- Via partial()
- Our own version

Ex11 Ex12