FUNCTIONAL PROGRAMMING

Functional programming: a language where each line of code is made up of calls to a function, which in turn may be made up of other functions or result in a value. Pros:

- Broader abstraction leading to fewer errors
- Functions have no side effects so they only need testing once
- No concurrency issues
- In multi-processor environments the sequence in which functions are evaluated is not critical

Function types

Function: rule that, for each element in some set A of inputs, assigns an output chosen from set B. Any function, f, has a function type f: A → B (where the type is A → B, A is the argument type, and B is the result type).

- Domain: set from which the function's input values are chosen.
- Co-domain: set from which the function's output values are chosen. Not all of the codomain's members need to be outputs. The values that are used are referred to as the *range*.

First class objects and higher order functions

In functional programming languages, a function is a first-class object.

- First class object: are objects which may: appear in expressions, be assigned to a variable, be assigned as arguments or be returned in function calls. For example, integers, floating-point values, characters and strings are first class objects in many programming languages.

Function application

Process of calculating the result of a function by passing it some data to produce a result.

Partial function application: the process of applying a function by creating an intermediate function by fixing some of the arguments to the function

- add: int x int → int (x is the Cartesian product)

  full application of the function which takes two integers as arguments passed at the same time and adds them together

- add: int → int → int

  partial application where a new function is created which always add the first argument value onto a number. Add 4 returns a function which when applied to another integer adds 4 to that integer

Function composition

Combining two or more functions together to create more complex functions

Given two functions f: A → B and g: B → C, the function g ∘ f, called the composition of g and f, is a function whose domain is A and co-domain is C. f is applied first and then g is applied to the result returned by f.

Writing functional programs (Haskell)

Higher order functions: a function that takes a function as its input or creates a function as its outputs.

- Map

  Higher-order function that applies a given function to each element of a list, returning a list of results.

  ```
  square x = x * x
  map square [1,2,3,4]
  ```

- Filter

  Higher-order function that processes a data structure, typically a list, in some order to produce a new data structure containing exactly those elements of the original data structure that match a given condition.

  ```
  filter odd [1,2,3,4]
  ```

- Reduce or fold

  Higher-order function which reduces a list of values to a single value by repeatedly applying a combining function to the list values.

  ```
  foldl (/) 64 [4,2,4]       →    2.0
  ```

List processing

List: collection of data items of the same type that can be stored using a single identifier. It can be divided in head, the first element, and tail, all the items apart from the head.

Standard processes

- `let SetA = [1,2,3,4]` — *construct a list*
- `head SetA` — *return the head*
- `tail SetA` — *return the tail*
- `null SetA` — *test for an empty list*
- `length SetA` — *return the length*
- `let SetB = [0] ++ SetA` — *prepend an item*
- `let SetC = SetA ++ [0]` — *append an item*