Topic 12 Fundamentals of Functional Programming

Functions

A function can be defined as:

"A rule that, for each element in some set of inputs, assigns an output chosen from set B, but without necessarily using every member of B."

Data (called an argument) is passed to the function, which applies a rule to it to create an argument. The argument could be any data type such as the number 1, the letter P, an entire file or even another function. Functions usually require a specific data type to work correctly and this is specified when the function is created.

Example Function

As an example, imagine a function named TimesFour, which applies a rule to multiply its input by four. The function could be given a set of natural numbers [1,2,3,4] as an argument and would produce the output numbers [4,8,12,16]

Function Types

All functions have a declared type, if we use f to represent the function, A to represent the input and B to represent the output, the function type can be defined as:

$$f: A \rightarrow B$$

The input (A) is also called the argument type or the domain, and the output (B) is also called the result type or the co-domain. We can talk about the formula above by saying that function f maps A to B

Example Function Type

Our example function above returns a number which is four times the input. The function will also only accept natural numbers, meaning we could write the function type as:

$$F: \{1, 2, 3, 4\} \rightarrow \{4, 8, 12, 16\}$$
$$F: NaturalNumbers \rightarrow NaturalNumbers$$

This means that the function will not accept a negative number, even though these can be times by four.

First-Class Object

First Class objects can be defined as:

First-class objects (or values) are objects which may
Appear in expressions
Be assigned to a variable
Be assigned as arguments
Be returned in function calls

Examples of first-class objects include functions, integers, characters and strings.

Function Application

Function application means applying the functions' rule to the arguments it is presented with.

Function Application Example

As an example, imagine the function Addition will add two integers and return the result. Two integers added together will always give an integer so we the function type could be:

```
Addition ::: int → int → int
```

Partial Function Application

As shown in the examples above, a function can accept more than one argument, allowing it to process more than one piece of data at a time. Partial function application fixes the number of arguments the function takes, meaning not all arguments are provided.

This can be thought of as creating a new function, which carries out only part of the calculation in the original function.

Function Composition

Function composition creates a new function by combining two existing functions. The functional programming concept relies on the idea of using function composition to build complex functions from smaller individual functions.

This allows the original functions to be used as they were previously along with the new function. For functions to be combined the domain of one of the functions must be the same as the co-domain of the other. The ○ symbol is used to show two functions being combined as shown in the example below:

```
Subtract4 ○ Triple
```

When combined, functions work from the inside out, meaning that in the example above the input would be tripled then 4 subtracted from the result.

Higher Order Functions

A higher order functions is one which takes a function as an argument, returns a function as an output, or both. The exam requires you to know about three higher order functions, map, filter and fold.

Map Function

The map function takes two arguments, a function and a list. It applies the function to all elements in the list and returns a new list. It is called the map function, since it maps one element in the original list, to a corresponding element in the new list which has had the function applied to it.

The example below shows how the map function might apply a function to subtract 5 to a list of three numbers.

```
Map (-5) {10,15,20} >> {5,10,15}
```

Filter Function

The filter function takes a list and a criteria as arguments, it returns only elements from the list which match the criteria.

The example below shows how the filter function might filter a list to include only numbers greater than 20.

```
Filter (>20) {10,25,6,50,2,5,30} >> {25,50,30}
```

Fold Function

The fold function takes a list, starting number and a function as an argument, it uses the function to reduce the list to a single value.

The example below shows how the fold function could be used to add a list of numbers together, starting at 0:

```
Foldl (+) 0 {10,15,20} >> 45
```

This example shows how the fold function could be used to add numbers, but this time starting at 50:

```
Foldl (+) 50 {10,15,20) >> 95
```