AQA Computer Science A-Level

4.12.2 Writing functional programs

Advanced Notes

<u>**Specification:**</u>

4.12.2.1 Functional language programs

Show experience of constructing simple programs in a functional programming language.

Higher-order functions: a function is higher-order if it takes a function as an argument or returns a function as a result, or does both.

Have experience of using the following in a functional programming language:

• map. *Map* is the name of a higher-order function that applies a given function to each element of a list, returning a list of results.

• filter. *Filter* is the name of a 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.

• reduce or fold. *Reduce* or *fold* is the name of a higher-order function which reduces a list of values to a single value by repeatedly applying a combining function to the list values.

Functional Programming

Resources to help you learn a functional programming language are available in the extra resources.

Synoptic Link

Parameters are the **variables/items** in the **declaration** of the **function**. The **argument** is the **actual value** passed into the **parameter** when the function is called.

Arguments and parameters are covered in **Functional Programming Paradigm** under **Fundamentals of Functional Programming**.

Higher Order Functions

Higher order functions either take a function as an argument, returns a function as its result, or does both. The three examples you need to be aware of are map, filter and fold.

```
{8,2,3,7,12,1}
```

```
Map (+1) {8,2,3,7,12,1}
>> {9,3,4,8,13,2}
```

```
Filter (>4) {8,2,3,7,12,1}
>> {8,7,12}
```

```
Foldl (+) 100 {8,2,3,7,12,1}
>> 133
```

Map

The map function takes a second function and applies it to a list of elements before returning the new list.

Map Example 1

We have the following list:

```
{1,2,3}
```

However, we want to minus 2 from each element. To do this, we can use the map function.

Function as argument

List as argument

```
Map (-2) {1,2,3}

>> {-1,0,1}
```

Outputted list

Map Example 2

We have the following list:

```
{2,4,6,8,10,12,14,16,18,20}
```

We want each element to be squared.

```
Map (^2) {2,4,6,8,10,12,14,16,18,20}

>> {4,16,36,64,100,144,196,256,324,400}
```

Map Example 3

There is a list of single fruit items.

```
{"Orange", "Pear", "Apple", "Banana"}
```
:

We want to change the list so each element is plural. The easiest way of doing this is adding an "s" on the end of each word.

```
Map (++ "s") {"Orange", "Pear", "Apple", "Banana"}

>> {"Oranges", "Pears", "Apples", "Bananas"}
```

Filter

The filter function returns the elements of the list which adhere to the condition given. It is a filter IN rather than a filter OUT.

Filter Example 1

We have the following list:

```
{1,2,3}
```

We only want the elements greater than 1.

Filter condition. Also a function passed as an argument.

List passed as argument

```
Filter (>1) {1,2,3}

>> {2,3}
```

List returned

Filter Example 2

We have the following list:

```
{2,4,6,8,10,12,14,16,18,20}
```

We only want the elements greater than 20.

```
Filter (>20) {2,4,6,8,10,12,14,16,18,20}

>> {}
```

<u>Fold</u>
By folding a list, you can reduce the list to a single value. Folding is also known as reducing. The fold function needs an operator (+,- etc), a starting number and a list. The type of fold - left or right - determines how the recursion will work. You will only be asked questions on fold left, but it is a good idea to have an understanding of fold right.

Synoptic Link

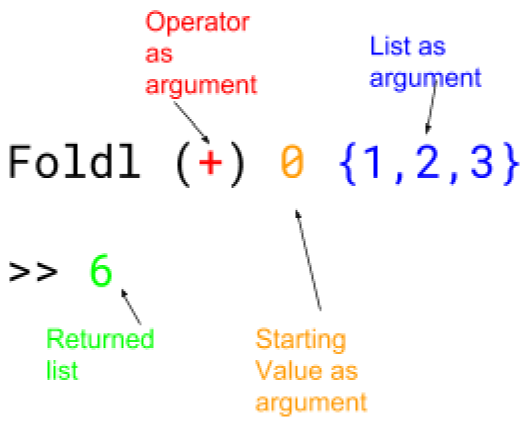
Recursion is when a **function** calls **itself** in order to **solve** a **problem**.

Recursion is covered in **Programming** under **Fundamentals of Programming**.

<u>Fold Example 1</u>
We have the following list:

```
{1,2,3}
```

We want to total this list.



```
Foldl (+) 0 {1,2,3}

>> 6
```

What is happening?

$$\begin{tikzcd}
\boxed{\texttt{Foldl (+) 0 \{1,2,3\}}} \arrow[d] \\
\boxed{\texttt{(((0+1)+2)+3)}} \arrow[d] \\
\boxed{\texttt{6}}
\end{tikzcd}$$

In this case, the result would be the same for fold right, it would just be calculated differently.

$$\begin{tikzcd}
\boxed{\texttt{Foldr (+) 0 \{1,2,3\}}} \arrow[d] \\
\boxed{\texttt{(1+(2+(3+0)))}} \arrow[d] \\
\boxed{\texttt{6}}
\end{tikzcd}$$

<u>Fold Example 2</u>

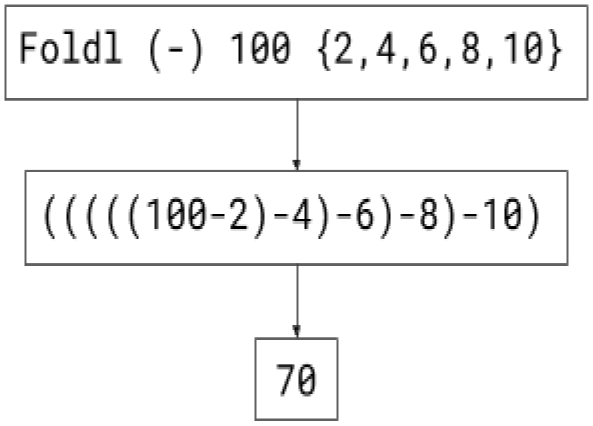
We have the following list:

```
{2,4,6,8,10}
```

We can use the fold function to reduce this list to a single number.

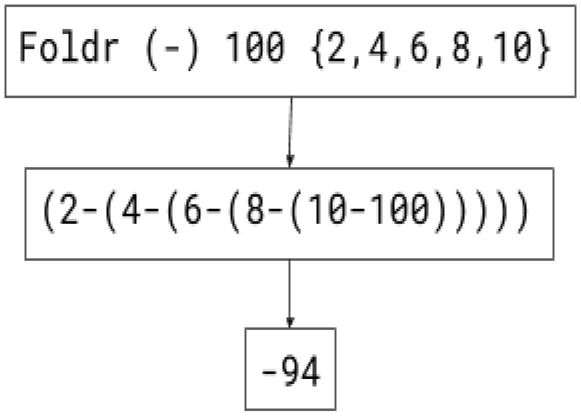
```
Foldl (-) 100 {2,4,6,8,10}

>> 70
```

What happened?



```
Foldl (-) 100 {2,4,6,8,10}
((((( 100-2)-4)-6)-8)-10)
70
```

Using fold right in this case, produces a very different answer.



```
Foldr (-) 100 {2,4,6,8,10}
(2-(4-(6-(8-(10-100)))))
-94
```