12.1 Functional programming paradigm

12.1.1 Function type

What is a function?
Loosely speaking, a function is a rule that, for each element in some set \( A \) of inputs, assigns an output chosen from set \( B \) but without necessarily using every member of \( B \).

For example, the function \( f \):

\[
f: \{0,1,2,3\} \rightarrow \{0,1,2,3,4,5,6,7,8,9\}
\]

maps 0 to 0, 1 to 1, 2 to 4 and 3 to 9 when the rule is: output the square of the input.

Function as process

In function as process, a function is a rule that tells us how to transform some information into some other information, e.g. the integer 2 into its square 4.

Function as object

In function as object, the function is a thing in its own right.

For example, a pencil sharpener is an object. If the focus of attention is a pencil then the pencil sharpener just represents a process - sharpening pencils, input: unsharpened pencil; output: sharpened pencil.

In the function as process view, we are applying the function \( \text{sharpen} \) to pencils; it's the pencil that counts. But we can also think about the pencil sharpener as a thing in its own right, when we empty it of pencil shavings, or worry about whether its blade is sharp enough. This is the function as object view.

Questions

1. A function \( f \):
   
   \[
f: \{0,1,2,3\} \rightarrow \{0,1,2,3, ..., 25, 26, 27\}
   \]
   
   maps 0 to 0, 1 to 1, 2 to 8, 3 to 27.

   What is the rule?

2. A function \( f \):
   
   \[
f: \{0,1,2,3\} \rightarrow \{0,1,2,3, 4, 5, 6\}
   \]
   
   maps 0 to 0, 1 to 2, 2 to 4, 3 to 6.

   What is the rule?
What is a function type?

Just as data values (e.g. 6, 9.1, True) have types (integer, real, Boolean respectively) so do functions. Function types are important because they state what type of argument a function requires and what type of result it will return.

A function $f$ which takes an argument of type $A$ and returns a result of type $B$ has a function type which is written $A \rightarrow B$.

To state that $f$ has this type, we write $f : A \rightarrow B$.

For example,

1) $\text{squareroot} : \text{real} \rightarrow \text{real}$
2) $\text{square} : \text{integer} \rightarrow \text{integer}$

The function named $\text{squareroot}$ applied to an argument of data type $\text{real}$ produces a result of data type $\text{real}$, e.g.

$\text{squareroot}(4.0) \rightarrow 2.0$

The function named $\text{square}$ applied to an argument of data type $\text{integer}$ produces a result of data type $\text{integer}$, e.g.

$\text{square}(2) \rightarrow 4$
**Domain and co-domain**

If \( f : A \rightarrow B \) is a function from \( A \) to \( B \), we call the set \( A \), the domain of \( f \), and the set \( B \) the co-domain of \( f \). The domain and co-domain are always subsets of objects in some data type. For example, if \( A \) is a subset of domain data type \( \text{integer} \) then its values might be \( 0, 1, 2, 3, ..., 149, 150 \). Often it is just convenient to use the data type directly,

\[
\text{square} : \text{integer} \rightarrow \text{integer}
\]

The function \( \text{square} \) then has an argument type, \( \text{integer} \) and a result type, \( \text{integer} \) even though in practice a subset of integers only will be used.

**Practical Activity**

Use a text editor such as NotePad++ to write Haskell programs. Save these Haskell programs using extension .hs.

*Figure 12.1.1.1* shows NotePad++ being used to create a function named \( \text{square} \) with one parameter \( x \) of data type \( \text{Integer} \) and a body \( x \times x \). This file has been saved with filename \( \text{square.hs} \) in folder \( c:\\book\\haskell \).

The :: operator (read as *has type*) is used in Haskell to express what type an expression has.

\( \text{Integer} \) is the type of mathematical integers (\( \text{int} \) could have been used and is the type of integers that fit into a word on the computer - this will vary from computer to computer).

Launch WinGHci if you are using a machine running the Windows operating system (\( \text{ghci} \) on Linux-based machines). The WinGHci window is shown in *Figure 12.1.1.2*. 

---

**Key concept**

**Domain and co-domain:**

If \( f : A \rightarrow B \) is a function from \( A \) to \( B \), we call the set \( A \), the domain of \( f \), and the set \( B \) the co-domain of \( f \).
At the Prelude prompt (Prelude>) type the command to change to a specified folder.

:cd c:\book\haskell followed by <return>.

Commands begin with a colon, i.e. :

Now load the file containing the program defining the function square.

At the Prelude prompt type

:load square.hs followed by <return>.

WinGHCi will perform a compilation of a module called Main in order to run square.hs interactively.

If there are no errors loading and compiling the Prelude prompt will be replaced by the prompt *Main.

At the *Main prompt, type

square 4 followed by <return>.

The correct answer, 16, is displayed.

To return to the Prelude prompt, type :module or :m

In this chapter you have covered:

- Function as process
- Function as object
- Function, $f$, has a function type, $f : A \rightarrow B$ where the type is $A \rightarrow B$.
- $A$ is the argument type, and $B$ is the result type.
- $A$ is called the domain and $B$ is called the co-domain.
- The domain and co-domain are always subsets of objects in some data type.