

## 2.1.3.3 Functional Programming

### [Return to Programming Paradigm](#)

The [Functional Programming](#) methodology focuses on results, not the process. Therefore, there can be many different ways to achieve the desired results and none of the different approaches are right or wrong. This is ideal when trying to integrate multiple existing technologies and/or products together when they are providing the same or very similar **Function**. Therefore, a key aspect of the Functional Programming methodology is the decomposition of the domain problem space (i.e., DIDs) into a set of well defined, deterministic functions, in other words, the same set of inputs always produce the same set of outputs regardless of when it is run or the the order the functions are run in.

The Functions are self contained and do not create any side effects or depend on any side effects. Simplistically it means that there is no interaction with external data from within the function. This is very similar to the mathematical definition of a function which uses relational expressions and recursion to do perform the calculations. It does not support iteration like loop statements and conditional statements like If-Else. The data that is passed into the function is [immutable](#) (it can not be changed).<sup>1)</sup>

Some examples of Functional [Programming Languages](#) are:

- Clean
- Clojure
- Erlang
- F#
- Haskell
- Make
- Mathematica
- ML/OCaml [Lisp](#) / Scheme
- Scala
- SML
- SQL
- XSLT

In Figure 1, the green boxes represent functions and the blue cylinders represent the data that flows into the function. Note, there is no data other than the data passed into the function used by the function. There are no decision points (i.e., diamonds) on the diagram.

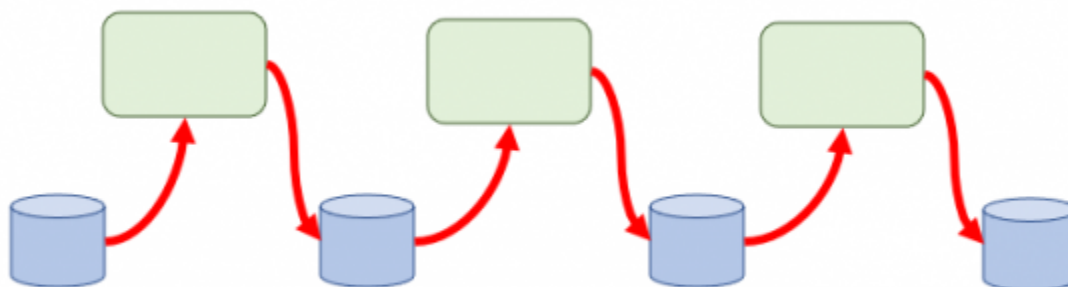


Figure 1: Function Programming - Data Flows

The DIDO [Command Line Interface](#) (DIDO-CL) is a hybrid of [Functional Language](#) and a [Procedural Language](#). The Functional aspects are built around operations that are similar to mathematical functions. For example, the summation of a column of numbers, the insertion, update or deletion of data into a RDBMS. In Functional languages the action comes first and the the thing the action is to be applied to:

Table 1: Examples of mapping of Functions in SQL, POSIX and MAKE.

CLI	Function	Data	Description
SQL	INSERT	DbTable	Insert data into DbTable
SQL	UPDATE	DbTable	Update data of DbTable
SQL	DELETE	DbTable	Delete data from DbTable
POSIX	ls	Filespec	list the files in Filespec directory
POSIX	rm	Filespec	remove files in the Filespec directory
POSIX	ps	Owner	Process Status (ps) for processes belonging to Owner
MAKE	clean	programs	Removes all the object files that had been created for for the
MAKE	install	programs	Compile the program and copy the executables, libraries, etc to the designated area
MAKE	all	programs	Compile the entire program

The Elixir language is also a functional language. The development of a Math Library is demonstrates of how a function based constructs are done. The following is an example of a Math Module:

### [| An Example of an Elixir <odule](#)

```
defmodule Math do
  def sum(a, b) do
    a + b
  end
end
```

The use of a Functional Programming approach used in a traditional Procedural Programming language like C/C++ would be some of the reusable libraries such as string, stderr, locale, etc. See [C Standard Library Reference Tutorial](#)<sup>2)</sup>.

1)

Guru99, Accessed: 13 April 2021, <https://www.guru99.com/functional-programming-tutorial.html>

2)

TutorialsPoint, Accessed: 15 April 2021, C Standard Library Reference Tutorial,  
[https://www.tutorialspoint.com/c\\_standard\\_library/index.htm](https://www.tutorialspoint.com/c_standard_library/index.htm)

From:

<https://www.omgwiki.org/dido/> - **DIDO Wiki**

Permanent link:

[https://www.omgwiki.org/dido/doku.php?id=dido:public:s\\_cli:05\\_contents:01\\_prt:02\\_basics:03\\_paradigim:funcpro](https://www.omgwiki.org/dido/doku.php?id=dido:public:s_cli:05_contents:01_prt:02_basics:03_paradigim:funcpro)

Last update: **2021/08/13 13:25**

