Robert Szkutak

As programmers, every time we are faced with a problem to solve we are also faced with a nearly limitless number of ways with which it can be solved. One of the first choices toward crafting a solution which a programmer makes is choosing the programming language with which to solve the problem. It is crucial for a programmer to consider whether an imperative, functional, logical, or object oriented language would be the best approach for the problem. The best approach to writing a program to solve a problem is usually considered to be the solution which is most efficient and easily maintainable. Thus, a good programmer should conceptualize a problem with solutions in multiple paradigms of languages before picking and actually programming a solution. It's impossible to objectively say which type of programming language is the best as the answer varies between different problems.

As a programmer dealing with the storage of sensitive information such as passwords and credit card numbers you understand that it is important to keep such material secure. Storing such information in plaintext would be terribly insecure so you decide to store each piece of sensitive user information input into your program as a SHA-256 hash. So you will need to write a program which encrypts and decrypts information input by the user to and from SHA-256 hashes. The SHA-256 cryptographic hash function is a very straightforward and very well documented algorithm. Implementing it in an imperative language simply requires performing the algorithm step by step. Using this algorithm in an imperative programming language requires nothing but your basic math skills and your ability to follow an algorithm. Were you to implement the SHA-256 algorithm in a functional programming language it would require a some more carefully laid out thought as you would probably have a function perform one step of the algorithm and then call itself to perform the next step. Implementing the SHA-256 algorithm in a logical programming language is nearly impossible to even conceptualize. The only way I can see it being done would be by listing out every possible result of the algorithm and using some rather complex predicates to determine which possibility was the correct one. Implementing the SHA-256 algorithm as part in an object oriented programming language would require you to think of the information being encrypted or decrypted as well as the algorithm itself af an object. After creating an instantiation of each object, you would run through a similar set of steps as in programming in an imperative language. In this case, I believe an imperative programming paradigm would be the best choice because the algorithm being used is rather straightforward so it's implementation should also be kept straightforward and simple. I believe any other programming paradigm used to run the algorithm would result in many unavoidable inefficiencies resulted by taking an overly complex approach to perform a simple task.

As a programmer you're developing a program which will search a file system and output total number of all files with the extension "txt". You'll need to account for searching all subdirectories found as the filesystem is explored. Performing this task in a functional programming language would be trivial. You would simply write a function with an integer return value which counts all the files with the "txt" extension in a directory, calls itself to search any subdirectories found, and returns the number of files with the "txt" exension found when the current directory and all of it's subdirectories have been searched. Performing this task in an

imperative programming language would be a bit more tedious. You would use a loop to keep searching for files with the “txt” extension until no more files with the “txt” extension or subdirectories are found. Performing this task in a logical language would require you define rules which determine what a file with the “txt” extension is, and what a subdirectoy is. The program would need to return the full path name of each “txt” file when queried. Peforming this task in an object oriented programming language would require you to think of the filesystem, subdirectories, and files with the “txt” extension as objects. Furthermore, it might be useful to implement a doubly linked list object (consisting of node objects) within the filesystem object to perform search functions. As a programmer, I feel the logical implementation would be the most difficult conceptually, I feel the imperative implementation would not be the most efficient, I feel the object oriented approach would be a bit complex but still very clean and perhaps not worth the cost of memory depending on the implementation of the objects. I feel a solution implemented in a function programming language for this task would be the simplest and most efficient solution and thus it would be the one I would choose.

As a programmer you’ve been hired by a political science researcher to write a program which determines which political party the user is most likely affiliated with. The user will answer a few simple questions about their feelings on things such as the economy or the war overseas and will then be told which political party their views most strongly represent them as being a member of. Solving this problem in a logical programming language would require you to come up with a list of rules which would determine what views make one most affiliated with one political party. So for example you would write a rule saying “if the user believes in completely free market economy and if the user believes in legalization of all non-violent crime then the user is a libertarian”. Solving this problem in an imperative programming language would be done similarly. After receiving the users input you would then implement a possibly more complex mathematical algorithm to determine which political party the user is most likely to be affiliated with. Solving this problem in a functional programming language would require a function which receives one piece of input, decides which party the user is most similar to, and then calls itself again to receive another piece of input and reevaluate until all input as been evaluated. Solving this problem in an object oriented programming language would require you to think of the user and each politcal party as obejcts. After recieving input from the user you would compare the values of the fields of each object and determine the user to be a member of whichever party his fields most closely match. For this problem, I believe a logical programming language would be the easiest to implement a solution because it is the programming paradigm which most clearly defines a set of rules to accurately determine a logical result. While other types of programming languages could be manipulated to achieve an equally accurate result, no other programming paradigm is so inherently able to achieve this result so efficiently.

As a programmer you’ve been hired by a biological research company studying the evolution of species to write a program which presents a simulation of their results. In particular, they want you to show how the company believes life on Earth evolved from bacteria to dinosaurs to the plethora of species inhabiting our planet today. To accomplish this task in an object oriented language you would think of the most basic level of biological material, bacteria, as an object. From there, you would think of other objects which come from it such as mammals

and vegetation as objects too. These objects would inherit the properties of the bacteria. You would go on and on creating an object for each species and appropriately giving it the polymorphic qualities of it's ancestors from whence it came. Solving this problem in an imperative language would be done by writing out the entire simulation for one species, then the next, then the next, and so on and so forth until every species is fully represented. It would take a very long to do this. Solving this in a functional language would require a function which simulates one species of life and then calls itself to demonstrate the next species again and again until all species have been demonstrated. Solving this in a logical programming language would require a large number of rules clearly defining the heritage of each species which would allow queries to demonstrate which species were derived from which species. I believe using an object oriented language is the best approach to solving this problem because an object oriented most clearly defines the relationships between objects and most likely can be used to contain the least redundant and cleanest code.

When solving a problem as a programmer one of the very choices you must make is which type of programming language to use to write your solution. Which type you choose depends on what you conceptualize to be the most efficient, clean, and maintainable to write your solution in. Each problem requires a different task to be solved and the best choice of language will vary for each problem. This step of choosing a type of language is perhaps the most crucial as it will dictate how the rest of your solution is handled. Choosing what type of programming language to use is never a decision that should be made with haste, but a decision which should be made only after careful planning.