**ASSIGNMENT 003 DATE: 28/08/2023**

**TITLE: Write about the paradigms of programming language.**

**TUTOR: G-WHYTE.**

Programming paradigms are different ways or styles in which a given program or programming language can be organized. Each paradigm consists of certain structures, features, and opinions about how common programming problems should be tackled.

Programming paradigms are not languages or tools. You can't "build" anything with a paradigm. They're more like a set of ideals and guidelines that many people have agreed on, followed, and expanded upon.

Programming languages aren't always tied to a specific paradigm. There are languages that have been built with a certain paradigm in mind and have features that facilitate that kind of programming more than others ([Haskel](https://www.haskell.org/) and functional programming is a good example).

But there are also "multi-paradigm" languages, meaning you can adapt your code to fit a certain paradigm or another (JavaScript and Python are good examples).

At the same time, programming paradigms aren't mutually exclusive, in the sense that you could use practices from different paradigms at the same time with no problem at all.

**Imperative Programming**

Imperative programming consists of sets of detailed instructions that are given to the computer to execute in a given order. It's called "imperative" because as programmers we dictate exactly what the computer has to do, in a very specific way.

Imperative programming focuses on describing *how* a program operates, step by step.

Using an actual code example, let's say we want to filter an array of numbers to only keep the elements bigger than 5. Our imperative code might look like this:

const nums = [1,4,3,6,7,8,9,2]

const result = []

for (let i = 0; i < nums.length; i++) {}

if (nums[i] > 5) result.push(nums[i])

console.log(result) // Output: [ 6, 7, 8, 9 ]

See that we're telling the program to iterate through each element in the array, compare the item value with 5, and if the item is bigger than 5, push it into an array.

We're being detailed and specific in our instructions, and that's what imperative programming stands for.

## Procedural Programming

Procedural programming is a derivation of imperative programming, adding to it the feature of functions (also known as "procedures" or "subroutines").

In procedural programming, the user is encouraged to subdivide the program execution into functions, as a way of improving modularity and organization.

## Functional Programming

Functional programming takes the concept of functions a little bit further.

In functional programming, functions are treated as **first-class citizens**, meaning that they can be assigned to variables, passed as arguments, and returned from other functions.

Another key concept is the idea of **pure functions**. A **pure** function is one that relies only on its inputs to generate its result. And given the same input, it will always produce the same result. Besides, it produces no side effects (any change outside the function's environment).

With these concepts in mind, functional programming encourages programs written mostly with functions (surprise 😲). It also defends the idea that code modularity and the absence of side effects makes it easier to identify and separate responsibilities within the codebase. This therefore improves the code maintainability.

Going back to the array filtering example, we can see that with the imperative paradigm we might use an external variable to store the function's result, which can be considered a side effect.

const nums = [1,4,3,6,7,8,9,2]

const result = [] // External variable

for (let i = 0; i < nums.length; i++) {

if (nums[i] > 5) result.push(nums[i])

}

console.log(result) // Output: [ 6, 7, 8, 9 ]

To transform this into functional programming, we could do it like this:

const nums = [1,4,3,6,7,8,9,2]

function filterNums() {

const result = [] // Internal variable

for (let i = 0; i < nums.length; i++) {

if (nums[i] > 5) result.push(nums[i])

}

return result

}

console.log(filterNums()) // Output: [ 6, 7, 8, 9 ]

It's almost the same code, but we wrap our iteration within a function, in which we also store the result array. In this way, we can assure the function doesn't modify anything outside its scope. It only creates a variable to process its own information, and once the execution is finished, the variable is gone too.

## Declarative Programming

Declarative programming is all about hiding away complexity and bringing programming languages closer to human language and thinking. It's the direct opposite of imperative programming in the sense that the programmer doesn't give instructions about how the computer should execute the task, but rather on what result is needed.

This will be much clearer with an example. Following the same array filtering story, a declarative approach might be:

const nums = [1,4,3,6,7,8,9,2]

console.log(nums.filter(num => num > 5)) // Output: [ 6, 7, 8, 9 ]

See that with the filter function, we're not explicitly telling the computer to iterate over the array or store the values in a separate array. We just say what we want ("filter") and the condition to be met ("num > 5").

What's nice about this is that it's easier to read and comprehend, and often shorter to write. JavaScript's filter, map, reduce and sort functions are good examples of declarative code.

Another good example are modern JS frameworks/libraries like React. Take this code for example:

<button onClick={() => console.log('You clicked me!')}>Click me</button>

Here we have a button element, with an event listener that fires a console.log function when the button is clicked.

JSX syntax (what React uses) mixes HTML and JS in the same thing, which makes it easier and faster to write apps. But that's not what browsers read and execute. React code is later on transpiled into regular HTML and JS, and that's what browsers run in reality.

JSX is declarative, in the sense that its purpose is to give developers a friendlier and more efficient interface to work with.

An important thing to notice about declarative programming is that under the hood, the computer processes this information as imperative code anyway.

Following the array example, the computer still iterates over the array like in a for loop, but as programmers we don't need to code that directly. What declarative programming does is to **hide away** that complexity from the direct view of the programmer.

## Object-Oriented Programming

One of the most popular programming paradigms is object-oriented programming (OOP).

The core concept of OOP is to separate concerns into entities which are coded as objects. Each entity will group a given set of information (properties) and actions (methods) that can be performed by the entity.

OOP makes heavy usage of classes (which are a way of creating new objects starting out from a blueprint or boilerplate that the programmer sets). Objects that are created from a class are called instances.