**Programming Paradigms**

Programming Paradigms describe the approach and system that is used when writing code and planning out programs. It has been used to classify programming languages based on their features however it more accurately describes the style a programmer adopts for creating a project.

To use football as a real-world example (that’s soccer to you not hand-egg) let’s use the sport as an analogy for a program written in code. In the 2000’s Barcelona FC adopted a pretty fancy tactic called Tiki Taka whereby the players constantly passed the ball back and forth amongst themselves. A player would never hold onto the ball for more than a second. This caused the opposition to move out of place to intercept the ball which is when Barcelona would exploit this weakness to score. In the 90’s the Republic of Ireland national team used a much uglier tactic called Route One which involved kicking the ball up to the opposition goal where the tallest guy on the team would be with the hope of him heading the ball into the net.

Both these teams played football. They both coded a program however they used different styles and tactics which follow their own rules and systems. Tiki Taka worked great in Spain and has its own strengths and weaknesses while Route One did the job for the Irish. Programming Paradigms have their strengths and weaknesses as well as their fans and detractors. You may be forced to use one based on the language you’re using, or you may opt to adopt one based on the needs and scale of the project you’re working on.

Certain languages do limit themselves to one paradigm over another such as Java and Object-Oriented Programming. Certain people say that JavaScript and Python are OOP languages because of the way they are built (using objects as pretty much everything) however that is not correct because we can write programs with them in whatever fashion we would like.

Fundamentally programming paradigms come down to how you write and organize your code.

**Useful Terms**

Before discussing programming paradigms in detail two terms that you’ll come across a lot are declarative and imperative.

**Declarative** (declare) means to express the logic of a computation without describing its control flow. Say what it does without going into detail about how to get there. It describes *what* you do.

*It’s like bringing your car to the mechanic to fix. You don’t know how it got back to working but it does.*

**Imperative** uses statements that changes a program’s data. It involves writing a list of instructions to follow. It describes *how* you do something.

*It is like calling a parent to help you fix your car. They will give you instructions describing what you need to do to get it working.*

**State** is used quite a bit to refer to data. **Mutable** means changeable and **immutable** means it can’t or won’t change.

**Types of Programming Paradigms**

If you were to google “types of programming paradigms” you will find that there isn’t a huge consensus on the topic. There are three main paradigms that come up however: procedural, object-oriented, and functional programming.

Below is a good link that goes through each type using a JavaScript example. There’s a long 45-minute video accompanying this however the short article itself will hopefully be good enough:

<https://academind.com/tutorials/functional-vs-oop-vs-procedural>

**Procedural Programming**

This is an ***imperative*** paradigm is probably the way you code now and was the first real paradigm ever. Simply put, Procedural Programming involves writing down a ***list of instructions*** (procedures) to tell the computer what it should do step-by-step to finish the task at hand. The list is usually sequential and is written top to bottom. Procedural Programming involves using variables, both local and global which has the potential to introduce side effects.

* Very verbose
* Can be lengthy
* Can be hard to read
* Hard to reuse often requiring copy and paste

**Object-Oriented Programming (OOP)**

Everything is based around objects. We breakdown the tasks, separate them into smaller chunks or entities (***encapsulation***) called objects. Most of the code, including data (properties/attributes) and logic (methods) are hidden inside objects. We can reuse and maintain code better because of this. The attributes and methods that work closely together should live in the same class/object and ***inheritance*** is used to share data and logic between classes and objects. When errors occur in your code, they are confined to their respective objects, therefore they are easier to track down and isolate.

OBJECT METHOD

console.log(‘Hello World’)

In the above JavaScript example, we don’t need to know what the **console** object does. All the code is abstracted away from us. All we need to know is how to use the **.log()** method for our own purpose.

**Functional Programming**

Functional code is very ***declarative*** and is composed of lots and lots of functions. The logic is described using functions, ideally ***pure functions*** (functions that return the same result if given the same parameters), which describe what task is being accomplished. Data is passed around as parameters in functions and functions are the way in which the code is organized. Each function should ideally work on its own, on its own merits. Functional Programming does not like using variables (local or global) as this introduces the potential of ***side effects*** brought about by other areas of the code accessing or modifying these values unintentionally. Loops are also avoided in favour of functional programming methods like **map()**, **filter()** and **reduce()** or using ***recursion*** which is where a function will call itself continuously until a condition is met to stop this self-created loop.

+ Easy to test

|  |  |  |
| --- | --- | --- |
| **Procedural Programming** | **Object-Oriented Programming** | **Functional Programming** |
| Procedural programming is built around the idea that programs are sequences of instructions to be executed. They focus heavily on splitting up programs into named sets of instructions called procedures, analogous to functions. A procedure can store local data that is not accessible from outside the procedure’s scope and can also access and modify global data variables. | Object-oriented programming is a programming paradigm built on the concept of objects that contain both data and code to modify the data. Object-oriented programming mimics a lot of the real-world attributes of objects e.g. a cat has properties/attributes like eye colour and fur but also perform actions/methods like purring and running. | Using immutable data structures and avoid side effects by doing all computation by using mathematical expressions. Basically, no creating/modifying variables. Don’t use loops and instead use map and filter functions |
| Programs are composed of sequences of instructions. | Objects comprise data that define its state and methods that define its behavior. Each object encapsulates these two entities. | Use pure functions which return same result when given same parameters. Higher-order functions are functions that can either take other functions as arguments or return them as results. |
| Procedures, which are logical blocks consisting of groups of instructions, can be invoked from other places in the code. | Objects are instances of classes. Classes are blueprints to build objects. | Avoid for/while loops using map and filter functions or recursion (a function that keeps calling itself until it doesn’t need to anymore). |
| Functions can access and modify variables in the global scope. | Classes can inherit both state and behavior from other classes | Reduces bugs, easier to test |