https://www.educative.io/blog/declarative-vs-imperative-programming

Imperative: oldest and most basic, step by step process.

Declarative: what to want to achieve, define the results but without description of control flow.

Escala de tiempo

Descripción generada automáticamente

|  |  |
| --- | --- |
| **Imperative Programming** | **Declarative Programming** |
| **1. Computation** | |
| You describe the **step-by-step instructions** **for how** an executed program achieves the desired results. | You set the conditions that trigger the program execution to produce the desired results. |
| **2. Readability and complexity** | |
| With the emphasis on the control flow, you can often follow the step-by-step process fairly easily. However, as you add more features and code to your program, it can become **longer and more complex**, making it increasingly confusing and time-consuming to read. | Step-by-step processes are eschewed. You’ll discover that this paradigm is **less complex and requires less code**, making it easier to read. |
| **3. Customization** | |
| A straightforward way to customize and edit code and structure is offered. You have **complete control** and can easily adapt the structure of your program to your needs. However, because you might have to deal with more code, you're more likely to run into editing errors than with declarative programming. | **Customizing the source code is more difficult** because of complicated syntax and the paradigm’s dependence on implementing a pre-configured algorithm. Some declarative programming programs may require more specificity to execute complex algorithms and functions |
| **4. Optimization** | |
| Adding extensions and making upgrades are supported, but doing so is **significantly more challenging** than with declarative programming, making it harder to optimize. This owes to the step-by-step structure of the paradigm and the fact that simple tasks require more code to process. The longer the code, the more likely you will run into errors. | You can **easily optimize code** because an algorithm controls the implementation. Furthermore, you can add extensions and make upgrades. |
| **5. Structure** | |
| The code structure can be **long and complex**. The code itself specifies how it should run and in what order. Due to the increased complexity, the code can sometimes be confusing because it may perform more than one task. | The code structure is **concise and precise**, and it lacks detail. Not only does this paradigm vastly limit the complexity of your code, but the code is more efficient. |

mylist = [1,2,3,4,5]

# set total to the sum of numbers in mylist

total = sum(mylist)

print(total)

Prolog:

write('Hello World').

LISP:

(defun hello ()

(format t "Hello, World!~%"))

Diagrama

Descripción generada automáticamente