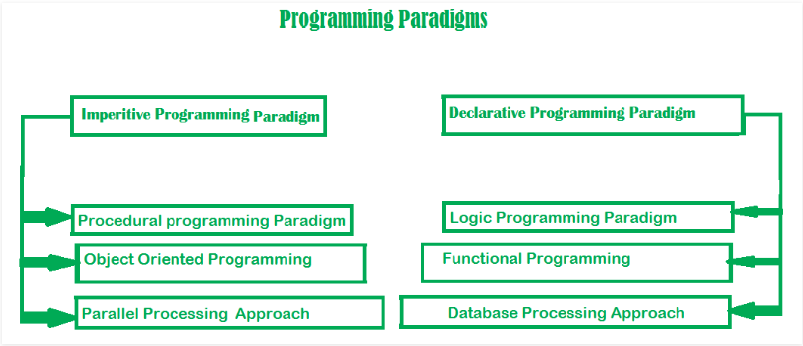
Programming paradigms:

<https://www.geeksforgeeks.org/introduction-of-programming-paradigms/>



* **Imperative (instruction) programming: !!procedure!!** how to do things
  + **Procedural programming:** same as imperative programming; a set of instructions/orders to achieve specific functionalities
    - Eg: C; ColdFusion; Pascal
  + **OOP:** written as a collection of **Classes** and **Objects** which are meant for communication
    - Eg: Simula (1st OOP); Java; C++; Python; JavaScript; Obj-C; Ruby

***4 Features***

<https://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/>

* + - **Abstraction** (real world 🡪 object world; **decoupling**): only show public relevant data & hide unnecessary details
      * Eg1: login page: show account name & password; how it is verified is abstracted from you
      * Eg2: car object: show manipulation system to you; subsystem (engine, gearing) are all hided from you
    - **Encapsulation (**group data into ***class*;** Data **Security/info hiding; decoupling)**: bind related data & methods; keep them from other interfaces
      * Eg1: classes in Java
      * Eg2: car steering system: complex inside, simple external interface; independent functionality, do not affect other system
    - **Inheritance** (**reusability**): we can derive our ***subclass*** from the ***existing superclass***. By doing this, we are **reusing** the fields and methods of the ***existing superclass***.
      * Eg1: Java AWT (parent) 🡨 Java Swing (child)
    - **Polymorphism: one object has more is-a relationship: Dog d = new Dog(); d is Dog and animal;**
      * **Static Polymorphism: Overloading**
        + one method with multiple implementation;
        + 2 (or more) methods with the same name but different arguments
        + **Static:** because which one to invoke is decided during ***compilation***
      * **Dynamic Polymorphism: Overriding**

https://www.tutorialspoint.com/java/java\_polymorphism.htm

* + - * + ***Subclass*** implementing a method from its ***superclass***
        + **Dynamic**: overridden method is resolved at ***runtime***, thus called runtime polymorphism

***5 Design OOP Principles: S.O.L.I.D***

<https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>

* + - Single-responsibility
    - Open-closed
    - Listov substitution
    - Interface segregation
    - Dependency Inversion
  + **Parallel processing programming**: program instructions by dividing them among multiple processors
* **Declarative programming:** **!!Goal/Result!!** What to do (**NOT** how to compute (**NOT** procedure))
  + Functional programming: building software by pure functions; NO shared state, NO mutable data, NO side-effects
    - **Principle**: execution of series of math functions
    - **4 features like OOP**:
      * **Abstraction**
      * **Encapsulation**
      * **Inheritance**
      * **Polymorphism**
    - **Feature**: data are loosely coupled to **functions**; function hide their implementation
      * Avoids **changing state** & **mutating data**
      * Using **expression** or **declaration** instead of **statements**
    - **Eg**: **JavaScript**; **Python**; Lisp (1st func prog; impure: functional + imperative); Haskell (pure functional), Scala
  + Logical programming: emphasize knowledge base & the problem; solve logical problem (puzzle)
    - Eg: Prolog
  + DB driven programming: based on DB and its movement

