Principios de Diseño de Software

—

con POO en Python

Algoritmos

En informática, un algoritmo corresponde al **conjunto de instrucciones** definidas, ordenadas y acotadas que permiten resolver un problema, hacer un cálculo o desarrollar una tarea determinada.

No son restrictivos de la matemática, lógica o computación, sino que se utilizan constantemente en la resolución de problemas de nuestra vida cotidiana. Por ejemplo: instrucciones ensamblado, recetas, etc.

La definición de los algoritmos es el paso previo a la programación.

Podemos dividir los algoritmos informáticos en tres partes:

Principios de diseño

La consideración de algunos princpios al momento de diseñar algoritmos favorece enormemente la calidad general del software.

~~ Principios generales ~~

1) **DRY: Don’t Repeat Yourself**
   Evitar repetir instrucciones, procedimientos, o cualquier pieza de información definida.

2) **KISS: Keep it Simple and Stupid**
   Buscar la simplicidad como objetivo central del diseño.

3) **RFTM**
   Lamentablemente el conocimiento suele estar en el manual.

Algoritmos: estrategias de diseño

Pseudocódigo

El pseudocódigo es una **representación de alto nivel** en texto libre informal de un algoritmo o proceso. Utiliza una estructura similar a la de un lenguaje de programación, pero se diseña para ser leído por humanos en lugar de máquinas.

Puede incorporar **palabras del lenguaje natural** para representar las instrucciones. Por ejemplo: LEER, ESCRIBIR, BUCLE, VARIABLE, etc.

Idealmente se debe usan instrucciones claras, que suelen ser *semejantes* a las disponibles en el lenguaje de programación en el que se implementará finalmente el algoritmo.

Debe ser **claro** y **breve**.

Ejemplo:

```
INICIO
LEER número
SI número MOD 2 = 0 ENTONCES
        ESCRIBIR "El número es par"
SINO
        ESCRIBIR "El número es impar"
FIN SI
FIN
```

https://es.wikipedia.org/wiki/Pseudoc%C3%B3digo

Algoritmos: estrategias de diseño

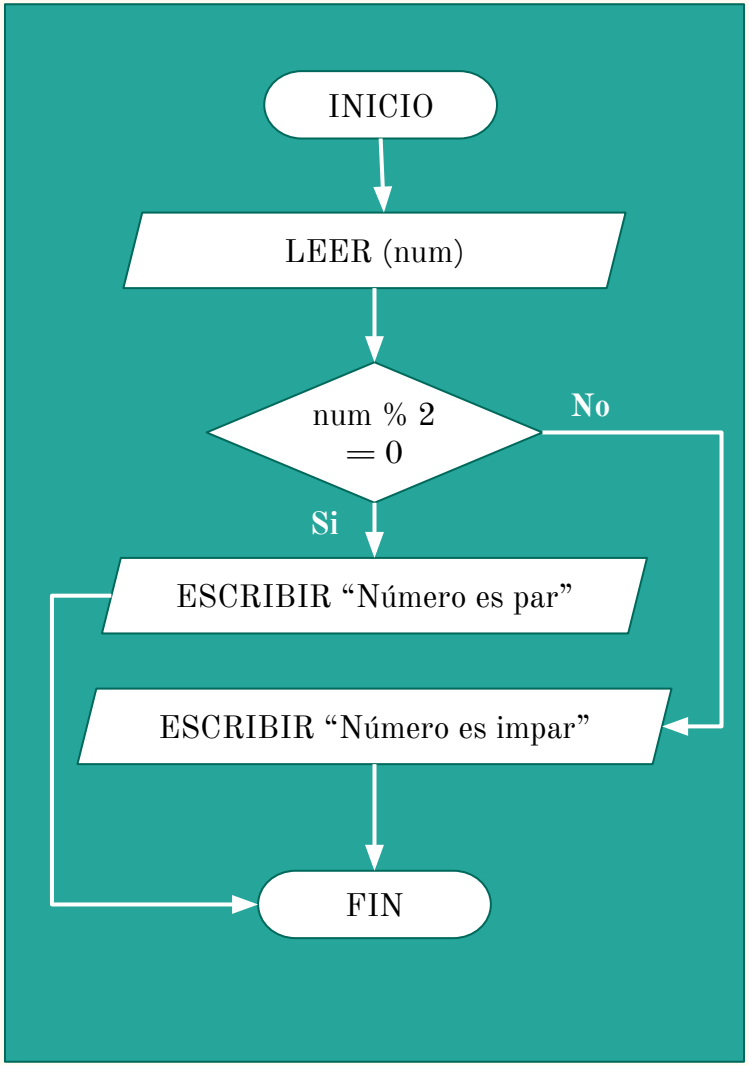
Diagramas de flujo (ordinograma)

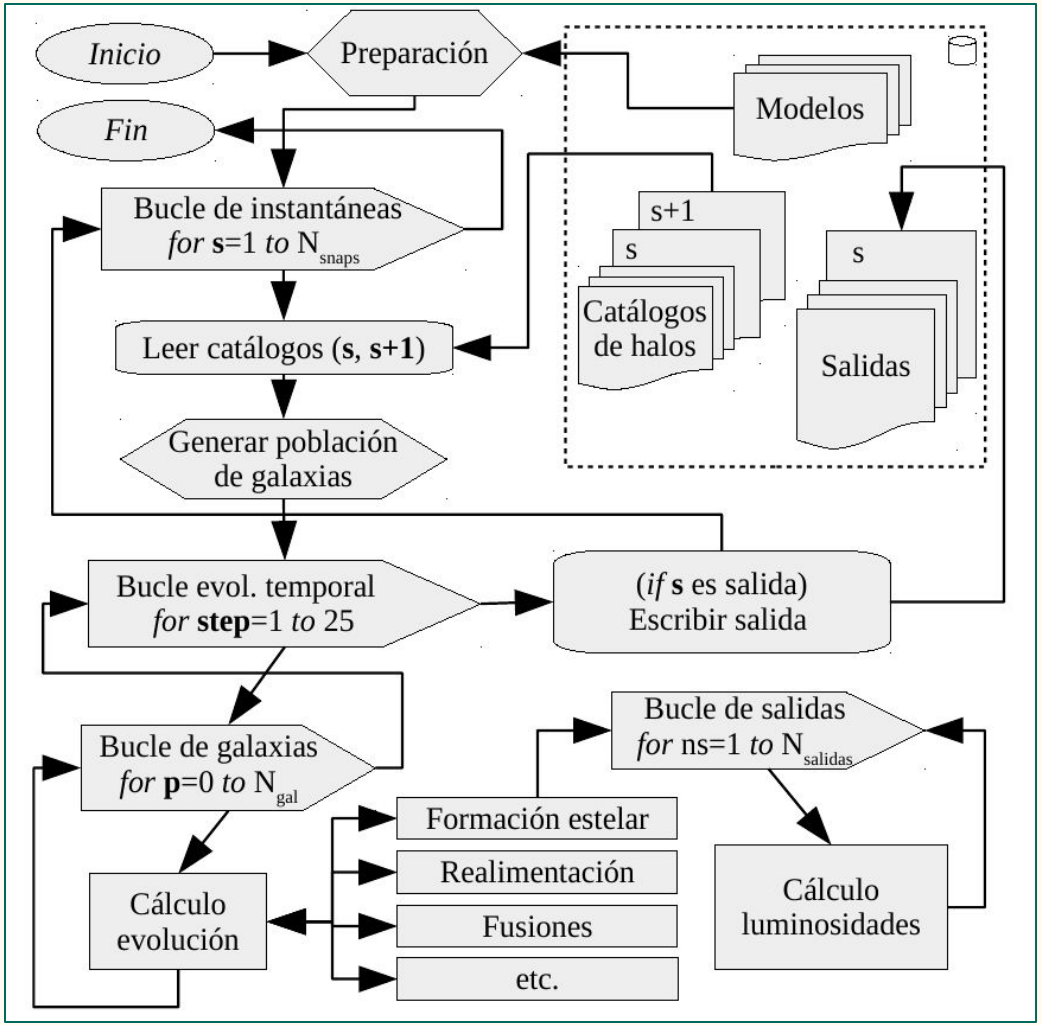
Es una representación de alto nivel del algoritmo o proceso en **formato de diagrama/esquema.**

Permite visualizar de manera esquemática las etapas, decisiones y flujos de un proceso determinado. Incorpora de forma explícita el I/O (input/ouput), las decisiones (condicionales), los bucles (iteraciones) y las asignaciones (variables/objetos) relevantes del algoritmo.

Utilizan **símbolos** con significados definidos que representan los pasos del algoritmo. Representan el flujo de ejecución mediante **flechas** que conectan los puntos de inicio y de fin del proceso.

https://es.wikipedia.org/wiki/Diagrama_de_flujo





Ejemplo aplicación

Diagrama de flujo personalizado

Este diagrama se realizó como análisis del flujo del código semi-analítico de formación y evolución de galaxias SAG.

Escrito en C
~ 100 archivos (.c y .h)
> 60.000 líneas de código

Solo la realización de este diagrama permitió identificar:

- las partes del cálculo que más tiempo de cómputo consumían
- procedimientos mal llamados
- niveles de anidación de blucles
- flujo por optimizar

Algoritmos en POO vs procedimental

La POO considera una forma distinta para abordar los diseños de los algoritmos:

- **Programación procedimental**
  El diseño de los algoritmos se enfoca en descomponer el problema siguiendo las acciones a realizar para abordarlo.

  → El algoritmo se diseña a partir de los **verbos** identificados.

- **Programación Orientada a Objetos**
  El diseño de los algoritmos se enfoca en descomponer el problema privilegiando los datos a ser representados virtualmente en el programa.

  → El algoritmo se diseña a partir de los **sustantivos** identificados.

Principios SOLID sugeridos para POO

1) **S: Principio de responsabilidad única (SRP)**
   Cada clase creada debe tener una única razón para cambiar (responsabilidad).

2) **O: Principio de abierto/cerrado (OCP)**
   Las entidades virtuales deben estar abiertas para extensión, pero cerradas para modificación.

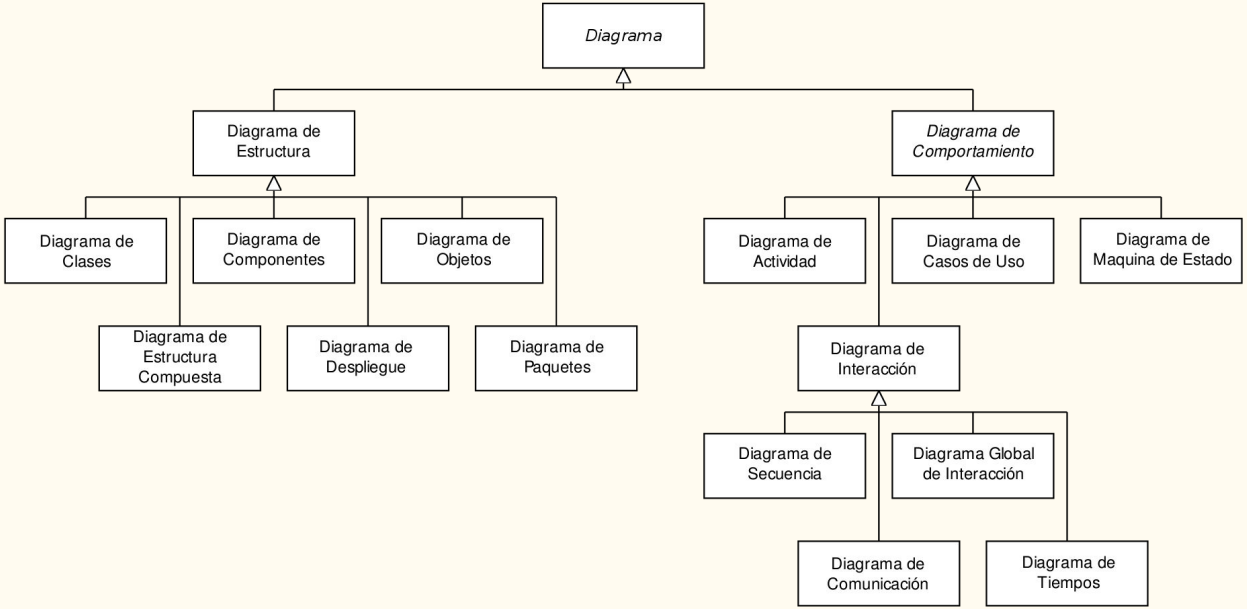
3) **L: Principio de sustitución de Liskov (LSP)**
   Los objetos deberían ser reemplazables por instancias de sus subtipos sin alterar el correcto funcionamiento del programa

4) **I: Principio de segregación de interfaces (ISP)**
   Clientes de un programa solo deberían conocer aquellos métodos que realmente usan, y no aquellos que no necesitan usar.

5) **D: Principio de inversión de la dependencia (DIP)**
   a) Los módulos de alto nivel no deberían depender de los módulos de bajo nivel; ambos deberían depender de abstracciones (p.ej., interfaces). b) Las abstracciones no deberían depender de los detalles; los detalles (implementaciones concretas) deben depender de abstracciones.

Unified Modeling Language (UML)

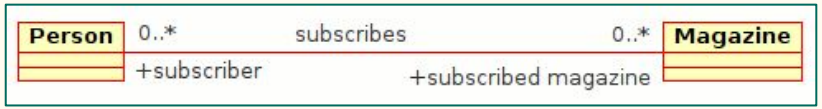
Es un **lenguaje gráfico** para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados. Se definen varios tipos:



UML: diagrama de clases

Diagrama de estructura estática que describe la **organización de un sistema** mostrando las clases definidas, sus atributos, operaciones (o métodos), y las relaciones entre los objetos.

Las **asociaciones** entre instancias de las clases se representan mediante líneas rectas, las que pueden incorporar la forma de asociación y la múltiplicidad (cúantas instancias espera):



También puede indicar las **herencias** (flechas vacías), la **agregación** de componentes temporales (rombos vacíos) y la **composición** (rombos llenos).