PROGRAMACION IMPERATIVA

PARCIAL 3

JENNYFER NIÑO TOBON

Profesor:

GUSTAVO MOJICA

UNIVERSIDAD NACIONAL DE COLOMBIA

PROGRAMACION DE COMPUTADORES

2025

PROGRAMACION IMPERATIVA

INTRODUCCION

Para la presentación del parcial #3 de PC, se escogió la programación Imperativa debido a que es interesante ver como a pesar de ser el paradigma más antiguo sigue vigente hoy en día, es compatible con múltiples paradigmas y programas además de ser ideal para principiantes, es por ello que en este documento se profundizara mas en este interesante paradigma.

PRESENTACION DEL PARADIGMA:

Proviene del latín imperare que significa orden y es el lenguaje de programación mas antiguo., se basa en una secuencia de instrucciones definidas a un ordenador que le dice que hacer para lograr el resultado deseado, para ello se integran estructuras de control.

Es un paradigma fácil de entender y sus múltiples lenguajes de programación hacen que sea accesible, pues es la base de C, Java, Python y JavaScript y utiliza tres estilos de programación: Estructurado, procedimental y modular.

En el estructurado se usan estructuras de control básicas como secuencias, selección e iteración con el fin de limitar las sentencias de salto; En el procedimental se divide la tarea en subtareas que se describen individualmente en el código dando como resultado módulos reutilizables; Y en el modular diseña, desarrolla y prueba componentes en módulos individuales que se combinan para crear un software real.

Como se ha dicho anteriormente, su mayor ventaja además de ser de fácil comprensión en el control detallado del flujo de ejecución, es decir se explica el proceso paso por paso, y una desventaja es que es mas propenso a errores debido a la alta manipulación y sus variables cambiantes.

IMPORTANCIA Y RELEVANCIA:

Gracias a la secuencia de instrucciones que maneja la programación imperativa sigue siendo relevante en la programación debido a la manipulación de datos que permite una mayor flexibilidad y un mayor control dentro del funcionamiento del programa.

Por ejemplo, en JavaScript se pueden construir aplicaciones web robustas, escalables y de alto rendimiento ya que haciendo uso de funciones, objetos y estructuras de control mutables permiten la reutilización de código, depuración y pruebas para corregir errores, optimización de rendimiento y legibilidad.

PARTE TEORICA

DEFINICION Y FUNDAMENTOS:

- Definición del Paradigma.

  También conocida como programación algorítmica, y es un paradigma de programación que utiliza fórmulas que alteran el estado de un programa, es decir, se basa en escribir programas que indican a la computadora que acciones deben realizar, su código fuente encadena una seria de instrucciones que determinan lo que el ordenador debe ejecutar, al ser un lenguaje concreto es fácil de entender, pero al ser tan explicito requiere de muchas líneas de código que muchas veces no tan necesarias.

- Principios Fundamentales que lo Caracterizan.

  - Secuencia de Instrucciones: Las instrucciones se ejecutan secuencialmente, una tras otra por lo que el orden importa y afecta el resultado del programa.
  - Estado Mutable: Hacen uso de variables y estructuras de datos mutables, estos pueden cambiar a lo largo de la ejecución del programa lo que permite realizar un seguimiento de cambios y manipular los datos según convenga.
  - Estructuras de Control: Estas construcciones controlan el flujo de ejecución del programa y permite tomar decisiones dependiendo de sus condiciones.

- Comparación con otros Paradigmas (Ventajas y Desventajas).

  Un paradigma de programación es un método para resolver un problema utilizando las herramientas adecuadas para cada ocasión, es por ello que para cada problema existe un paradigma ideal que resuelve esa necesidad, hay muchos paradigmas utilices y similares.
  Debido a que la programación imperativa funciona principalmente a través de funciones de asignación y realiza tareas paso a paso, haremos la comparación con la programación declarativa que se centra en describir la lógica del problema sin importar su implementación y con la programación funcional que tiene sus raíces en las matemáticas independientemente del programa.

- Imperativa vs Declarativa

Programación Imperativa	**Programación Declarativa**
Los programas especifican cómo debe hacerse.	Los programas especifican lo que se debe hacer.
Simplemente describe el flujo de control del cálculo.	Simplemente expresa la lógica del cálculo.
Su objetivo principal es describir cómo conseguirlo o lograrlo.	Su objetivo principal es describir el resultado deseado sin importar como conseguirlo.
Entre sus ventajas se encuentran la facilidad de aprendizaje y lectura.	Entre sus ventajas se incluyen un código efectivo, que puede aplicarse mediante métodos.
Su tipo incluye programación procedimental, programación orientada a objetos y enfoque de procesamiento paralelo.	Su tipo incluye la programación lógica y la programación funcional.
Se permite al usuario tomar decisiones y dar comandos al compilador.	Un compilador se le permite tomar decisiones.
Tiene muchos efectos secundarios e incluye variables mutables en comparación con la programación declarativa.	No tiene efectos secundarios y no incluye ninguna variable mutable en comparación con la programación imperativa.
Proporciona control total a los desarrolladores, lo cual es muy importante en la programación de bajo nivel.	Puede automatizar el flujo repetitivo y al mismo tiempo simplificar la estructura del código.

- Imperativa vs Funcional

Programación imperativa	***Programación funcional***
Generalmente es un proceso de describir pasos que simplemente cambian el estado de la computadora.	Generalmente es un proceso de desarrollo de software simplemente componiendo funciones puras, evitando o minimizando los efectos secundarios, los datos compartidos y los datos mutables.
Se centra principalmente en describir cómo se ejecuta u opera el programa, es decir, cómo procesa.	Se centra principalmente en qué programas se deben ejecutar u operar, es decir, los resultados.
Entre sus ventajas se incluyen que es fácil de aprender, fácil de leer, un	Sus ventajas incluyen código libre de errores, aumento del rendimiento, mejor encapsulación, aumento de la

modelo conceptual es fácil de aprender, etc.	reutilización, aumento de la capacidad de prueba, etc.
Sus características incluyen una secuencia de declaraciones, contiene estado y puede cambiar de estado, puede tener algunos efectos secundarios, modelo de programación con estado, etc.	Sus características incluyen baja importancia del orden de ejecución, modelo de programación sin estado, unidades de manipulación primarias, control de flujo primario, etc.
Estos programas están estructurados como asignaciones sucesivas de valores a nombres de variables.	Estos programas generalmente se estructuran como llamadas funcionales anidadas sucesivas.
Implica escribir programas como series de instrucciones o declaraciones que pueden modificar activamente la memoria.	Implica escribir programas en términos de funciones y estructuras matemáticas.
Requiere que la estructura de datos se represente como cambios de estado.	Requiere la representación explícita de las estructuras de datos que se utilizan.
Los nuevos valores suelen asociarse con nombres diferentes mediante la anidación de llamadas a funciones recursivas.	Los nuevos valores suelen asociarse al mismo nombre mediante la repetición de comandos.

Ambas tablas fueron sacadas del portal de Geeks for Geeks que realizó una comparativa muy interesantes sobre las ventajas y desventajas de los diferentes paradigmas de programación.

HISTORIA Y EVOLUCION:

- Reseña Histórica.

La programación imperativa es uno de los paradigmas mas antiguos y fundamentales en la historia de la computación, su origen se puede remontar a la década de 1940 y 1950, pues se basa en el modelo de la maquina de Von Neumann que conceptualiza una computadora con una memoria que almacena instrucciones; los lenguajes de programación eran imperativos escritos en código de máquina.

Posteriormente, con la aparición de Fortran en 1957 que era programa diseñado para cálculos científicos surgieron otros lenguajes como COBOL Y ALGOL.

En los años 70 surgieron los lenguajes de programación mas influyentes respecto a programación imperativa que fue Pascal y C.

A finales del Siglo XX y la programación imperativa evoluciono hacia la programación orientada a objetos (POO) con lenguajes como C++ y Java, pero aun así no pierde su base de programación imperativa al seguir usando instrucciones secuenciales y manipulación directa del programa.

En el siglo actual, es ampliamente utilizada en sistemas operativos y desarrollo de software de alto rendimiento, lenguajes como Python, JavaScript y C, aunque incorporan otros paradigmas sigue teniendo sus raíces en programación imperativa.

- Desarrollo y Mejoras a lo largo del Tiempo.

  Desde sus inicios con código de maquina hasta hoy, la programación imperativa ha evolucionado en mejoras estructurales y de seguridad, siendo hoy en día un pilar en el desarrollo de software y coexistiendo a la par con otros paradigmas de la programación.
  A continuación, se muestran los lenguajes a través del tiempo de la programación imperativa:

  - 1945: Maquina de Von Neumann y Código Maquina y Ensamblador.
  - 1957: Fortran, primer lenguaje de alto nivel.
  - 1959: COBOL, enfocado en comercio.
  - 1960: ALGOL, base de futuros lenguajes.
  - 1963: CPL, precursor de C.
  - 1966: BCPL, precursor de C.
  - 1970: Pascal, promueve la programación estructurada.
  - 1972: C, leguaje de sistemas de bajo nivel y estructuras de alto nivel.
  - 1983: C++, Expande C con clases y objetos.
  - 1983: Ada, lenguaje usado en sistemas críticos.
  - 1991: Python, lenguaje multiparadigma.
  - 1995: Java. Enfocado en POO.
  - 2000: JavaScript, adaptación del desarrollo web.
  - 2009: Go, simplicidad y eficiencia en sistemas concurrentes.
  - 2010: RUST, seguridad y eficiencia sin sacrificar control imperativo.

CASOS DE USO:

- Aplicaciones Prácticas y Escenarios donde el Paradigma es más Adecuado y Ejemplos de Software o Proyectos que lo utilizan.

  Hoy en día, se utiliza en lenguajes como C, C++, Python, Java, JavaScript, Go y C# debido a su control y flexibilidad.
  También se utiliza en los sistemas operativos como Windows, Linux y macOS debido a su control sobre procesos y memoria.
  En diferentes videojuegos ya que requieren alto rendimiento y control de flujo de ejecución.

Se utiliza en aplicaciones empresariales y web backend como Django en Python, .NET en C#, Node.js en JavaScript, entre otros.
En software de seguridad y redes (C), además de inteligencia artificial y ciencia de datos (Python).

LENGUAJES DE PROGRAMACION ASOCIADOS:

- Lenguajes de Programación que Soportan este Paradigma.

  A continuación, se mencionan los más conocidos:
  - Fortran
  - Java
  - Pascal
  - ALGOL
  - C, C++, C#
  - Ensambladores
  - BASIC
  - COBOL
  - Python
  - Ruby.

- Código Ilustrativo para demostrar su Implementación.

  Ejemplo mostrado en IONOS – Digital Guide:

  Los lenguajes de programación imperativa se distinguen por su carácter instructivo y para ello normalmente requieren considerablemente más líneas de código para expresar lo que en el estilo declarativo se puede describir con menos instrucciones. En el siguiente ejemplo, se debe proporcionar una lista con nombres:

  Programación imperativa (PHP):

```
$listaParticipantes = [1 => 'Peter', 2 => 'Hans', 3 => 'Sarah'];
$nombres = [];
foreach ($listaParticipantes as $id => $name) {
    $nombres[] = $name;
}
```

PARTE PRACTICA.

IMPLEMENTACION DE UN EJEMPLO PRACTICO:

Se realiza un programa que genere contraseñas seguras, en Python. Se hace uso de funciones y estructuras de control. Además de la flexibilidad al momento de generar los caracteres y su orden, hace que sea un ejemplo claro de programación imperativa.

```python
#Generador de Contraseñas Seguras

import string #Cadena de texto
import random #Genera datos aleatorios

def contraseña(longitud, numeros=True, mayusculas=True, caracteres=True): #Funcion Contraseña, pide el largo de esta, si tiene numeros, mayusculas y caracteres especiales.
    genera_contraseña = string.ascii_lowercase #inicia en minusculas
    if mayusculas: #Agrega mayusculas
        genera_contraseña += string.ascii_uppercase
    if numeros: #Agrega numeros
        genera_contraseña += string.digits
    if caracteres: #Agrega caracteres especiales
        genera_contraseña += string.punctuation

    contraseña = ''.join(random.choice(genera_contraseña) for _ in range(longitud)) #Genera la contraseña con las indicaciones anteriores.
    return contraseña #Guarda la contraseña

# Pregunta las caracteristicas de la contraseña
longitud = int(input("Ingrese la longitud de la contraseña: "))
numeros = input("¿Su contraseña tiene numeros? (si/no): ").strip().lower() == 'si'
mayusculas = input("¿Su contraseña incluye mayusculas? (si/no): ").strip().lower() == 'si'
caracteres = input("¿Su contraseña incluye caracteres? (si/no): ").strip().lower() == 'si'

print(f"\n Contraseña generada: {contraseña(longitud, numeros, mayusculas, caracteres)}") #Muestra contraseña aleatoria.
```

```
Ingrese la longitud de la contraseña: 8
¿Su contraseña tiene numeros? (si/no): Si
¿Su contraseña incluye mayusculas? (si/no): Si
¿Su contraseña incluye caracteres? (si/no): Si

 Contraseña generada: SY"0ED/J
```

DESAFIOS Y CONSIDERACIONES:

Al ser un ejemplo pequeño, no se encuentran muchos desafíos sin embargo se sabe que la función contraseña esta generada para un flujo especifico por lo que será difícil su reutilización, además de ser poco útil ya en un ambiente mayor como la clave de un correo electrónico.

Al utilizarse una única lista de caracteres es eficiente y rápido, es recomendable modificarlo de acuerdo a las condiciones de entornos mayores, es decir, un gestor más complejo de contraseñas.

COMPARACION DE OTROS PARADIGMAS:

Si se utilizara POO al usar clases y estructuras tendría un enfoque más robusto y funcional, además de la posibilidad de reutilizarlo al separar la lógica o si se utilizara programación funcional, sería más fácil de depurar y probar debido al uso de más funciones además de evitar cambios en los valores a guardar.

CONCLUSIONES

RESUMEN DE LOS PUNTOS CLAVES:

La programación Imperativa, a pesar de ser el paradigma mas antiguo, sigue siendo vigente hasta la actualidad, gracias a su versatilidad y control, es crucial para aplicaciones que requieren un manejo detallado de procesos y memoria y en comparación con otros paradigmas este ofrece un mayor control sobre el programa.

Gracias a su uso en lenguajes como C o Python, y en sistemas operativos, es muy probable que siempre este vigente y adaptándose a las necesidades de la programación, ejemplo de ello es su uso en la inteligencia artificial y seguridad informática.

REFLEXION SOBRE LA IMPORTANCIA DEL PARADIGMA EN EL DESARROLLO DE SOFTWARE:

Al ser el primer paradigma de programación es pionero en la industria, no solo para la comprensión de otros paradigmas, sino que también para la creación de software moderno, además de que está en constante evolución para el impulso del desarrollo tecnológico. Al ser funcional, seguro, de máximo rendimiento y optimización de recursos es ideal para el control de software.

OPINIONE SOBRE APLICABILIDAD EN LA INDUSTRIA ACTUAL DE ACUERDO A LA ESTADISTICA:

Como ya se ha mencionado, se usa en ciencia de datos, desarrollo web e inteligencia artificial. Tiene gran dominancia en la industria debido a su aplicabilidad y que lenguajes como Python y Java lo utiliza para aplicaciones empresariales. Esta demás decir que su desarrollo es imprescindible en muchos sectores claves de desarrollo y que es un paradigma casi imposible que desaparezca.

REPOSITORIO

https://github.com/Jenino1223/PARCIAL-3.git

BIBLIOGRAFIA

- *Imperative programming: Overview of the oldest programming paradigm*.

  (s/f). IONOS Digital Guide. Recuperado el 7 de marzo de 2025, de

  https://www.ionos.com/digitalguide/websites/web-development/imperative-programming/

- madhurihammad Follow Improve. (2021a, marzo 9). *Difference between imperative and declarative programming*. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-imperative-and-declarative-programming/
- madhurihammad Follow Improve. (2021b, marzo 11). *Difference between functional and imperative programming*. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-functional-and-imperative-programming/
- Murias, A. (2024, junio 10). *La Importancia de JavaScript en la Programación Imperativa*. Digital House. https://www.digitalhouse.com/blog/la-importancia-de-javascript-en-la-programacion-imperativa/
- Pérez, D. Q. (2024, enero 30). *Programación imperativa: Un enfoque Paso a Paso para resolver problemas de software*. Linkedin.com. https://es.linkedin.com/pulse/programaci%C3%B3n-imperativa-un-enfoque-paso-para-resolver-david-gdrte
- Wikipedia contributors. (s/f). *Programação imperativa*. Wikipedia, The Free Encyclopedia. https://pt.wikipedia.org/w/index.php?title=Programa%C3%A7%C3%A3o_imperativa&oldid=67343917