


PORTADA

<b>Nombre Alumno / DNI</b>	Marcos Marín / 53802325N
<b>Título del Programa</b>	Data Science & AI
<b>Nº Unidad y Título</b>	UNIT 1 PROGRAMMING & CODING
<b>Año académico</b>	2023-2024
<b>Profesor de la unidad</b>	GABRIELA GARCÍA
<b>Título del Assignment</b>	AB FINAL
<b>Día de emisión</b>	13/10/2023
<b>Día de entrega</b>	23/01/2024
<b>Nombre IV y fecha</b>	
<b>Declaración del estudiante</b>	<p>Certifico que la presentación del <u>assignment</u> es completamente mi propio trabajo y entiendo completamente las consecuencias del plagio. Entiendo que hacer una declaración falsa es una forma de mala práctica.</p> <p>Fecha: 23/01/2024</p> <p>Firma del alumno: </p>

**Plagio**

El plagio es una forma particular de hacer trampa. El plagio debe evitarse a toda costa y los alumnos que aseguranse de comprender las prácticas de referencia correctas. Como alumno de nivel universitario, se espera que utilice las referencias adecuadas en todo momento y mantenga notas cuidadosamente detalladas de todas sus fuentes de materiales para el material que ha utilizado en su trabajo, incluido cualquier material descargado de Internet. Consulte al profesor de la unidad correspondiente o al tutor del curso si necesita más consejos

## INFORME DE LENGUAJES, PARADIGMAS Y ESTÁNDARES DE PROGRAMACIÓN.

-Introducción: los lenguajes, paradigmas y los estándares son partes fundamentales en el mundo de la programación, los cuales desarrollan papeles clave en el desarrollo de software y la creación de sistemas informáticos.

-Tipos de lenguajes de programación: dentro de la programación existen distintos tipos de lenguaje. Los dividimos por niveles según su complejidad, por lo que tenemos lenguajes de bajo, medio y alto nivel.

- *Lenguajes de bajo nivel*: son lenguajes que dependen totalmente de la máquina, o lo que es lo mismo, el programa que realizamos con este tipo de lenguajes no se puede ni utilizar ni migrar en otras máquinas. Como por ejemplo ensamblador que es un ejemplo típico de este lenguaje, cuyos usos más comunes son programación de sistemas embebidos, controladores de hardware...
- *Lenguajes de medio nivel*: este lenguaje es una mezcla entre bajo nivel y alto nivel, donde podríamos situar "C" ya que podemos trabajar con direcciones de memoria, registros del sistema... las cuales son unas características de lenguajes de bajo nivel, pero a la vez pudiendo realizar operaciones de alto nivel. Como por ejemplo C y C++ los cuales los solemos usar para el desarrollo de sistemas integrados, controladores de dispositivos...
- *Lenguajes de alto nivel*: diferenciamos un lenguaje de alto nivel cuando se encuentra más cercano al lenguaje natural que al de la máquina. Por lo que en la mayoría de los casos un lenguaje de alto nivel lo puedes migrar de una máquina a otra sin problema. Como por ejemplo Python, Java, Java Script... Las cuales solemos usar para el desarrollo de aplicaciones web, ciencia de datos, inteligencia artificial, entre otras cosas.

- Paradigmas de programación: los principales paradigmas que encontramos son los siguientes: imperativo y declarativo, los cuales tienen diferentes características:

- *Imperativo*: según este paradigma, un programa consiste en una secuencia claramente definida de instrucciones para un ordenador. El código fuente de los lenguajes imperativos encadena instrucciones una detrás de otra que determinan lo que debe hacer el ordenador en cada momento para lograr el resultado deseado.

Para gestionar las instrucciones, se integran estructuras de control como bucles o estructuras anidadas en el código.

Los lenguajes de programación imperativa más conocidos son: Java, C, C++, Ensambladores, Python, Ruby...

- *Declarativo*: la base de este paradigma es que siempre se describe el resultado final deseado, en lugar de mostrar todos los pasos de trabajo. En esta programación se determina automáticamente la vía de solución. Esto funciona siempre y cuando las especificaciones del estado final se definan claramente y exista un procedimiento de ejecución adecuado. En caso de que se den las dos condiciones, la programación declarativa es muy eficiente.

Los lenguajes de programación declarativa más conocidos son: Prolog, Lisp, Haskell, SQL...

- Estándares de programación: son conjuntos de reglas y pautas que los desarrolladores siguen al escribir el código. Los objetivos de estos estándares es mejorar la legibilidad, la consistencia y la calidad del código, lo cual facilita la colaboración de desarrolladores y el mantenimiento a largo plazo del software. Descripción de algunos de los estándares de programación y sus características:

- *Python*: define reglas para la legibilidad del código, el uso de sangrías, la longitud de las líneas y la convención de nombres.
- *JavaScript*: ESLint es una herramienta que ayuda a aplicar estándares de codificación en JavaScript, y el estilo de Airbnb proporciona pautas detalladas para estructurar y escribir código JavaScript limpio.

- *Java*: Estándares para el desarrollo de portlets en aplicaciones Java, proporcionando una arquitectura estándar para la construcción de componentes reutilizables en portales web.

Beneficios de adherirse a estándares y consecuencias de no hacerlo:

- Legibilidad y mantenibilidad.
- Consistencia.
- Facilita la colaboración.
- Calidad del código.

Consecuencias de no adherirse a estándares:

- Dificultades de mantenimiento.
- Inconsistencia.
- Dificultades en la integración.
- Mayor propensión a errores.

- *Conclusión*: es muy importante comprender los diversos lenguajes de programación, paradigmas y estándares es fundamental para el desarrollo de software efectivo. Cada uno de los lenguajes, programas... tiene una función y un campo de trabajo, por lo que es muy importante conocer todos al máximo para así poder utilizar lo que mejor se adapte a nuestros intereses.

<https://catedrauno.com/programacion-lenguajes-niveles-436/>

<https://chat.openai.com/c/3a6dd0b3-3c0b-4310-8cdc-bf6c7d2b9d35>

<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/paradigmas-de-programacion/>

<https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/programacion-imperativa/>

<https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/programacion-declarativa/>

<https://web.rramirez.com/estandares-basicos-de-programacion/>

<https://www.uoc.edu/es/estudios/formacion-continua/asignatura-fundamentos-programacion-c>

## INFORME DEL TESTING Y PRUEBAS DE CÓDIGO

### INTRODUCCIÓN:

- El testing y las pruebas de código desempeñan un papel crucial en el ciclo de vida del desarrollo de software, asegurando la calidad y confiabilidad del producto final. Estas prácticas son esenciales para identificar y corregir errores, garantizar el cumplimiento de requisitos y mejorar la experiencia del usuario.

### CONCEPTOS BÁSICOS:

- DEFINICIÓN Y DIFERENCIA ENTRE TESTING Y PRUEBAS DE CÓDIGO:
  - El testing es un proceso de evaluación de un sistema para identificar los defectos, sin embargo las pruebas de código se centran en verificar el correcto funcionamiento de unidades individuales de código.
  - El objetivo de las pruebas de testing es la corrección de posibles errores, entrega de un software más fiable, además de mejorar la calidad del código.  
Los beneficios más notables podrían ser la identificación de problemas hasta la mejora del mantenimiento.

### - TIPOS DE PRUEBAS:

- Pruebas unitarias: son de muy bajo nivel y se realizan cerca de la fuente de la aplicación. Consisten en probar métodos y funciones individuales de las clases, componentes o módulos que usa tu software. En general, las pruebas unitarias son bastante baratas de automatizar y se pueden ejecutar rápidamente mediante un servidor de integración continua.
- Pruebas de integración: las pruebas de integración verifican que los distintos módulos o servicios utilizados por tu aplicación funcionan bien en conjunto. Por ejemplo, se puede probar la interacción con la base de datos o asegurarse de que los microservicios funcionan bien en conjunto y según lo esperado. Estos tipos de pruebas son más costosos de ejecutar, ya que requieren que varias partes de la aplicación estén en marcha.
- Pruebas de sistema: Se centran en evaluar el sistema completo. Selenium es comúnmente utilizado para pruebas de interfaz de usuario.

- Pruebas de aceptación: son pruebas formales que verifican si un sistema satisface los requisitos empresariales. Requieren que se esté ejecutando toda la aplicación durante las pruebas y se centran en replicar las conductas de los usuarios. Sin embargo, también pueden ir más allá y medir el rendimiento del sistema y rechazar cambios si no se han cumplido determinados objetivos.
- Pruebas de carga y estrés: Evalúan la capacidad del sistema para manejar carga normal y situaciones de estrés. Apache JMeter es ampliamente utilizado para pruebas de carga.

#### - TÉCNICAS DE TESTING:

- TDD (Test Driven Development): Implica escribir pruebas antes del código de producción.

Algún beneficio del TDD podría ser la mejora de la comunicación. Puesto que los problemas llegan en cortos periodos de tiempo.

- BDD (Behaviour Driven Development): Se basa en el comportamiento del sistema desde el punto de vista del usuario.

Algún beneficio de BDD puede ser la automatización de pruebas debido a la claridad y comprensión compartida de los escenarios.

#### - CASOS DE USO Y EJEMPLOS:

- Pruebas de Integración con Postman: en una situación de desarrollo de API, lo utilizamos para realizar pruebas de integración entre los distintos servicios.
- TDD con Python y pytest: seguimos la metodología TDD al escribir pruebas antes de desarrollar funcionalidades en un proyecto Python utilizando pytest.
- Pruebas unitarias con JUnit: en un proyecto de Java, implementamos pruebas unitarias con JUnit para verificar funciones individuales.

#### - CONCLUSIÓN:

- En conclusión, las pruebas de testing son un proceso muy importante, para localizar posibles errores en nuestra estructura y así poder mejorarla en todo lo que se pueda. A parte de solucionarnos los posibles problemas que nos surjan, también contribuye a la confiabilidad y satisfacción de los clientes.

## **INFORME ACERCA DE LOS FRAMEWORKS Y LIBRERÍAS**

### **-Definición de ambas:**

- Framework: estructura que proporciona un entorno para el desarrollo de software. Es un conjunto tanto de reglas como de herramientas para facilitar la creación y mantenimiento de las aplicaciones.
- Librerías: conjunto de rutinas y funciones que pueden ser utilizadas para desarrollar software. Las librerías no imponen una estructura específica y se utilizan para tareas específicas dentro de una aplicación.

### **-Ventajas de ambas:**

- Los frameworks tienen una estructura definida lo cual facilita el desarrollo coherente y ordenado.
- Los frameworks son más productivos, por lo que pueden acelerar el proceso de desarrollo.
- Los frameworks suelen estar diseñados para facilitar la escalabilidad de las aplicaciones.
- Las librerías se pueden utilizar y seleccionar según sea necesario sin comprometer la estructura de la aplicación, por lo que es más flexible que un framework.
- Al utilizar librerías, se reduce el acoplamiento, permitiendo una mayor modularidad en el desarrollo.

### **¿Por qué no he implementado frameworks en mi página?**

- No he utilizado frameworks ya que no lo he visto necesario, el diseño que he hecho a mano me ha gustado sin necesidad de implementar ninguno, además de que la implementación de un framework me obligaba a cambiar la mayor parte de la estructura del código que tenía ya hecho, por lo que por estos motivos he decidido no incorporar un framework a mi página.