

PROGRAMING


CURSO 2023-2024



TIENDA ONLINE

IBRAHIM SALEH

ibrahimsm@msmk.university

Nombre Alumno / DNI	ibrahim saleh	60143759R
Título del programa	PROGRAMING	
Año académico	2023-2024	
Profesor de la unidad	GABRIELA GARCIA	
Título del Assignment	RECUPERACIÓN AB final	
Día de emisión	18/09/2023	
Día de entrega	22/02/2024	
Nombre IV y fecha		
Declaración del estudiante	<p>Certifico que la presentación del assignment 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: 20-12-2023</p> <p>Firma del alumno:</p> 	

INFORME PARADIGMAS

¿QUÉ SON LOS PARADIGMAS

Se denominan paradigmas de programación a las formas de clasificar los lenguajes de programación en función de sus características. Los idiomas se pueden clasificar en múltiples paradigmas.

- **Lenguajes:** Los lenguajes de programación son herramientas que permiten a los desarrolladores comunicarse con los ordenadores y dar instrucciones para realizar determinadas tareas.
- **Paradigmas:** Los paradigmas de programación son enfoques o estilos de programación que te explican cómo se deben estructurar y organizar las instrucciones en un programa.
- **Estándares:** Los estándares son pautas y normas aceptadas por la comunidad de desarrollo para garantizar la coherencia y la interoperabilidad entre diferentes sistemas y aplicaciones.

Tipos de lenguajes de programación:

- **Alto nivel:** El lenguaje de alto nivel es el lenguaje que más se aproxima al lenguaje del ser humano, su función principal se basa en que a partir de su desarrollo hay una posibilidad de que se pueda utilizar el mismo programa en distintas máquinas, es decir

que es independiente de un hardware específico.

- Medio nivel: Lenguaje de medio nivel es un lenguaje de programación como el lenguaje C, es decir que se encuentran entre los lenguajes de alto nivel y entre los lenguajes de bajo nivel, suelen ser clasificados muchas veces de alto nivel, pero permiten ciertos manejos de bajo nivel. Son precisos para aplicaciones específicas como la creación de sistemas operativos.

- Bajo nivel: Los lenguajes de bajo nivel, también llamados lenguajes ensambladores, permiten al programador escribir instrucciones de un programa usando abreviaturas del inglés, también llamadas palabras nemotécnicas, tales como: ADD, DIV, SUB, etc.

- Compilados: Estos lenguajes se traducen completamente a código máquina antes de ejecutarse. Los programas escritos en lenguajes compilados deben pasar por un proceso de compilación antes de poder ser ejecutados. En los lenguajes compilados se traducen completamente en código máquina antes de su ejecución

- Interpretados: Un lenguaje de programación interpretado es un tipo de lenguaje de programación que se basa en otra pieza de software llamada intérprete para ejecutarse. Los lenguajes interpretados tienen varias limitaciones en términos de rendimiento, pero la programación con ellos es más fácil, por lo que son ideales para ciertos tipos de aplicaciones.

- Ejemplos:

- Alto nivel: Son independientes del hardware, necesitan ser interpretados o compilados, menor rendimiento, gran comunidad detrás y sintaxis flexible y fácil de leer, se usan en el desarrollo de programas y aplicaciones, Desarrollo de inteligencia artificial y en el desarrollo de bases de datos.

- Medio nivel: Abstracción de hardware, portabilidad, eficiencia y rendimiento, facilidad de programación, manipulación de bajo nivel, Acceso a Funcionalidades del Sistema Operativo y compilación o interpretación. Se usan para la creación de sistemas operativos ya que son precisos.

- Bajo nivel: gestión de memoria directa, dependen del hardware, ejecución más rápida, difícil de leer y escribir, poco apoyo y difíciles de aprender, se una para gestión de sistemas operativos control de todo tipo de máquinas o aplicaciones que usan sistemas en tiempo real.

LENGUAJES SEGÚN SU TIPADO DE DATOS:

Los tipos básicos de datos en la programación son tres: números, textos y booleanos.

Los números no requieren mayor explicación: tu año de nacimiento, tu número de teléfono, la duración del video. Los textos tampoco tienen misterio: "Hola QL team" es un texto, el título del vídeo es un texto.

Por último están los booleanos que son datos que solo tienen 2

valores: verdadero o falso y sirven para que el programa tome decisiones. Por ejemplo, si quieres iniciar sesión en apple ingresas tu usuario y contraseña, el sistema los evalúa y devuelve verdadero si son correctos y te deja pasar. O devuelve falso si son incorrectos y no te deja iniciar sesión.

Estos tres tipos de datos (número, texto y boolean) se conocen como primitivos porque son los básicos de cada lenguaje. Aunque algunos lenguajes tienen varios tipos de número y varios tipos de textos.

¿Qué es exactamente los lenguajes tipados y no tipados?

Los lenguajes tipados son los que exigen que se declare el tipo de dato en las variables, estructura de datos o funciones. Si no lo haces, error.

La diferencia fundamental entre ellos radica en cómo manejan los tipos de datos. En particular, cómo se declaran las variables y expresiones que vamos a utilizar.

Esta pequeña diferencia supone un gran cambio en el uso de los programas, y frecuentemente origina enormes debates sobre qué aproximación es mejor de usar.

En realidad es un debate eterno y sin demasiado sentido. Ambas formas de trabajar tienen ventajas y desventajas. De hecho, ambos tipos de lenguajes evolucionan para incorporar características del otro tipo.

Lo importante es entender las características y diferencias entre ambos y saber usar cada uno de ellos cuando resulte más oportuno.

Lenguajes tipados.

En un lenguaje tipado, también llamados de tipo estricto, se caracterizan por que requieren que definamos el tipo de dato específico de las variables y expresiones que vamos a utilizar.

Ejemplos de lenguajes tipados son C++, C# y Java, entre otros muchos. Veamos algunos ejemplos.

Por ejemplo, así se declaran variables en C# o Java.

```
int numero = 5;  
int resultado = número + 10;  
string mitexto = "hola";
```

Como vemos, en un lenguaje tipado tenemos que indicar el tipo de variable al declararla, por ejemplo un número entero into una cadena de texto string.

Una vez que una variable tenga un tipo está solo puede contener valores compatibles con este tipo. No podríamos asignar a una variable int un texto.

```
int a;  
a = "hola"; // esto daría lugar a un error
```

Por otro lado, los lenguajes no tipados, también llamados de tipado dinámico, no necesitan que indiquemos el tipo de las variables y expresiones al declarar la variable.

Ejemplos de lenguajes no tipados son Python y JavaScript. Veamos algunos ejemplos.

Por ejemplo, en JavaScript no es necesario indicar el tipo de una variable al declararla. Únicamente tenemos que usar la palabra

reservada `let` para indicar que queremos crear una variable.

```
let número = 5;
```

```
let resultado = número + 10;
```

```
let mitexto = "hola";
```

En la gran mayoría de lenguajes no tipados es posible incluso cambiar el tipo de variable una vez creada.

```
let a = 5; // aqui 'a' contiene el número 5
```

```
a = "hola"; // ahora 'a' contiene el texto "hola"
```

En realidad, internamente los lenguajes no tipados si tienen tipos. Tu programa lo necesita para saber cómo manipular los valores de las variables. Pero el lenguaje de programación hace que su uso sea transparente para el usuario la mayoría de las veces.

Paradigmas de programación:

- Imperativo: Los programas consisten en una sucesión de instrucciones o conjunto de sentencias, como si el programador estuviese dando órdenes concretas. El programador describe el código paso a paso de que va a hacer su programa, Programación estructurada: La programación estructurada es un tipo de programación imperativa donde el flujo de control se define mediante bucles anidados, condicionales y subrutinas, en lugar de a través de `GOTO`. Programación procedimental: Este paradigma de programación es en el que consiste en basarse en un bajo número de expresiones repetidas, englobarlas todas en un procedimiento o función y llamarlo cada vez que tenga que ejecutarse. Programación modular: el proceso consiste en dividir un

programa en módulos o subprogramas con el fin de hacerlo más manejable y legible. Se trata de una evolución de la programación estructurada para resolver

problemas de programación más complejos.

- Declarativo: Este paradigma no necesita definir algoritmos porque describe el problema en vez de encontrar una solución del mismo. Este paradigma utiliza el principio del razonamiento lógico para responder a las preguntas o cuestiones consultadas. Programación Lógica: prolog, Programación funcional: lisp, scala, java y kotlin

- Orientado a objetos: En este modelo de paradigma se construyen modelos de objetos que representan elementos (objetos) del problema a resolver, que tienen características y funciones. Permite separar los diferentes componentes de un programa, simplificando así su creación, depuración y posteriores mejoras. La programación orientada a objetos disminuye los errores y promueve la reutilización del código. La programación orientada a objetos se muestra de diferentes conceptos como: Abstracción de datos, Encapsulación, Eventos, Modularidad, Herencia y Polimorfismo.

- Funcional: Entendemos por programación funcional el lenguaje de programación declarativo donde el programador especifica lo que quiere hacer exactamente, en lugar de lidiar con el estado de los objetos. Es decir, las funciones estarían en un primer lugar y nos centraremos en expresiones que pueden ser asignadas a cualquier variable.

- Lógico: El paradigma de programación lógico se basa en la lógica matemática y tiene como objetivo principal la resolución de

problemas mediante la expresión de hechos y reglas lógicas. Uno de los lenguajes más representativos de este paradigma es Prolog. En lugar de enfocarse en cómo se deben realizar las operaciones (como en los paradigmas imperativos), el paradigma lógico se centra en declarar qué hechos y reglas son verdaderos.

Lenguajes representativos:

- Imperativo:

Usado en :c, fortran, cobol

Características Distintivas: Enfoque en describir cómo se deben realizar las operaciones.

- Orientado a objetos:

Usado en: c++,Java,python,c#

Características Distintivas:Organiza el código en torno a objetos

- Funcional:

Usado en: Skala, Haskell, lisp

Características Distintivas:

- Lógico:

Usado en: prolog

Características Distintiva

Estándares de programación:

- Los estándares son una serie de reglas establecidas para un lenguaje de programación, o bien un estilo de programación

determinado, el estilo garantiza que todos los ingenieros que contribuyen a un proyecto tengan una forma propia de diseñar su código, lo que da como resultado una base de código coherente, asegurando una fácil lectura y mantenimiento. El uso de estándares es muy importante en la calidad de software, sin embargo mantener todos los proyectos cumpliendo a la perfección con esto no es una tarea fácil, requiere un gran esfuerzo y constancia por parte del equipo de desarrollo. Mientras más y más compañías han adoptado estándares, todavía hay aquellas que realizan el desarrollo de sus proyectos sin ellos.

- **Descripción de estándares:**

PEP 8 Python Enhancement Proposal 8:

Descripción: Es una guía de estilo para el código Python, propuesta por la comunidad Python.

Características principales: Define convenciones de estilo para la escritura de código en Python, incluye pautas sobre la indentación, el uso de espacios en blanco, nombres de variables, entre otros.

JSON JavaScript Object Notation:

Descripción: No es un estándar formal, pero es ampliamente adoptado como formato de datos ligero y fácil de leer.

Características principales: Representa datos como pares clave-valor. Es independiente del lenguaje y fácil de entender.

Utilizado comúnmente en la comunicación entre sistemas web.

- **Beneficios de adherirse a estándares:**

Consistencia y Calidad, eficiencia y productividad, seguridad, interoperabilidad, innovación sostenible, cumplimiento normativo y

reputación y confianza

- consecuencias de no hacerlo:

problemas de interoperabilidad, riesgos de seguridad, problemas de calidad, código inconsistente y difícil de mantener, falta de cumplimiento normativo, dificultades en la colaboración e innovación limitada.

INFORME TESTING CÓDIGOS

¿QUÉ ES EL TESTING?

El testing o pruebas de software es la realización de pruebas sobre el código con el fin de detectar errores y mejorar su calidad. Según quien realiza las pruebas tenemos 2 tipos: manuales y automáticas.

MANUALES: Las pruebas manuales son aquellas en las que se prueba la navegación por la aplicación interactuando con la misma, por ejemplo accediendo a la aplicación y pulsando los botones para comprobar si funcionan o no.

AUTOMÁTICAS: las pruebas automáticas en las que se utiliza una herramienta para realizar los test. Por ejemplo, en una prueba automatizada podemos grabar una navegación y luego ejecutarla de forma automática desde la herramienta.

¿QUÉ TIPOS DE TEST HAY?

● Unit tests

- (Las pruebas unitarias son a nivel bajo)
- Este tipo de testing consiste en probar de forma individual las funciones o métodos, debido a lo específicas que son,
- Generalmente son las pruebas automatizadas de menor coste, y pueden ejecutarse rápidamente por un servidor de *continuous integration* (integración continua).
- Idealmente, cuando planeamos y escribimos pruebas unitarias, debemos aislar la funcionalidad hasta un punto en el que no se pueda separar más, y entonces hacer pruebas a partir de ello. Justamente, el nombre de este tipo de testing hace referencia a una "unidad de código", que es independiente del resto.
- Estas pruebas verifican que el nombre de la función o método sea adecuado, que los nombres y tipos de los parámetros sean correctos, y así mismo el tipo y valor de lo que se devuelve como resultado.
- Dado que las pruebas unitarias no deben tener ningún tipo de dependencia, se suele reemplazar los llamados a APIs y servicios externos por funcionalidad que los imite (para que no exista interacción que vaya más allá de la unidad que está siendo probada).
- En muchos casos incluso se suele reemplazar las consultas a bases de datos, de modo que la prueba este enfocado en operar a partir de los valores de entrada, sin depender de ninguna fuente externa.

- Si no es factible aislar el uso de bases de datos de nuestras pruebas unitarias, será importante tener en cuenta el rendimiento y buscar optimizar nuestras consultas.
- Esto es importante, porque si nuestras pruebas unitarias son de larga duración, resultará incómodo ejecutarlas y ralentizará significativamente los tiempos de desarrollo.

● Integration tests

- Las pruebas de integración verifican que los diferentes módulos y/o servicios usados por nuestra aplicación funcione en armonía cuando trabajan en conjunto.
- Las pruebas de integración son típicamente el paso siguiente a las pruebas unitarias.
- Y son generalmente más costosas de ejecutar, ya que requieren que más partes de nuestra aplicación se configuren y se encuentren en funcionamiento.
- Functional tests
- Las pruebas funcionales se centran en los *requerimientos de negocio* de una aplicación.
- Estas pruebas verifican la salida (resultado) de una acción, sin prestar atención a los estados intermedios del sistema mientras se lleva a cabo la ejecución.
- la diferencia entre los 2 es:
- una prueba de integración puede simplemente verificar que las consultas a una base de datos se ejecuten correctamente,

- mientras que una prueba funcional esperaría mostrar un valor específico a un usuario, en concordancia a lo definido por los requerimientos del producto.

- **End-to-end tests**

- Las pruebas de lado a lado el comportamiento de los usuarios con el software, en un entorno de aplicación completo.
- Estas pruebas verifican que los flujos que sigue un usuario trabajen como se espera, y pueden ser tan simples como: cargar una página e iniciar sesión.

- **Regression testing**

- Las pruebas de regresión verifican que un conjunto de escenarios que funcionaron correctamente en el pasado, para asegurar que sigan así:
- No debemos agregar nuevas características a nuestro regression test suite hasta que las pruebas de regresión actuales pasen.
- Una falla en una prueba de regresión significa que una nueva funcionalidad ha afectado otra funcionalidad que era correcta en el pasado, causando una "regresión".

- **Smoke testing**

- Son pruebas rápidas de ejecutar,
- Su objetivo final es asegurar que las características más importantes del sistema funcionen como se desea.

- **Acceptance testing**

- Estas pruebas requieren que el software se encuentre en funcionamiento, y se centran en imitar el comportamiento de los usuarios, con el objetivo de rechazar cambios si no se cumplen los objetivos. Estos objetivos pueden ir más allá de obtener una respuesta específica, y medir el rendimiento del sistema.
- **Performance testing**
- Las pruebas de rendimiento verifican cómo responde el sistema cuando se encuentra bajo una alta carga.

Estos tests son no-funcionales(no funcionan), y pueden tener muchas formas para entenderlo

- la fiabilidad.
- estabilidad.
- disponibilidad de la plataforma.

FRAMEWORKS

Un framework es un conjunto de reglas y convenciones que se usan para desarrollar un software de una manera mucho más rápida y eficiente. Estos marcos en el entorno de trabajo se emplean para ahorrar cierto tiempo y esfuerzo en el desarrollo de aplicaciones, ya que proporcionan una estructura básica que se puede utilizar como base para poder empezar.

¿QUÉ FRAMEWORKS HAY?

- 1-Laravel
- 2-Codeigniter
- 3-Symfony
- 4-Zend
- 5-Phalcon
- 6-Cakephp
- 7-Yii
- 8-FuelPHP
- 9-Java

¿QUÉ FRAMEWORKS SE APLICARIAN A UNA TIENDA ONLINE?

Estos son los frameworks que se aplicarían a una tienda online:

Symfony: la capacidad de controlar todos los accesos a la información además del control “Predeterminado” de ataques CSRF o XSS, permite la creación de apps en diferentes idiomas, tiene un código abierto de muy buena calidad, con una arquitectura fácil de usar y diseños fáciles de comprender y fáciles de utilizar para el desarrollador web.

Laravel: permite agregar información de utilidad mediante su directorio Packalyst, sistema de enrutamiento muy eficaz, lo que permite relacionar elementos de una determinada aplicación con las rutas que el usuario introduce en la web.

Phalcon: La principal ventaja del Phalcon, es la utilización eficiente de la memoria, Esto te permite eliminar la necesidad de mantener cualquier tipo de información fuera de la memoria.

Drupal: te permite almacenar y gestionar grandes cantidades de datos con la que puedes trabajar en base a un diseño escalando a un desarrollo a medida, esta hecho para sitios con mucho tráfico, la mayoría de las organizaciones gubernamentales y alguna compañía grande del mundillo internacional cuentan con sites desarrollados bajo este framework.

DISEÑO RESPONSIVE

¿QUÉ ES UN DISEÑO RESPONSIVE?

El diseño responsive es una técnica de diseño web que permite que un sitio web se ajuste automáticamente al tamaño y la resolución de la pantalla del dispositivo desde el cual se está accediendo, sin importar si un usuario está navegando desde un ordenador de sobremesa, ipad, o un teléfono, la página web se mostrará de manera clara y concisa, legible y fácil de navegar.

