Fundamentos profesionales del desarrollo de Software

Índice:

Fundamentos Esenciales de la programación

Introducción:

La programación como concepto ----------

- ¿En qué consiste el proceso de programar
- Trabajando con Pseudocódigo

Lenguajes de programación ----------

- Tipos de lenguajes de programación
- Introducción a los lenguajes de estilo y marcado

Aprender a programar con JavaScript ----------

- ¿Qué es y por qué usar JavaScript como primer lenguaje?
- Tipos de datos y datos numéricos
- Caracteres y strings
- Booleanos, arrays y objetos en JavaScript
- Variables en JavaScript
- Ámbito de las variables
- Var, Let y Const en JavaScript
- Tipos y usos de los operadores
- Operaciones con Strings
- Estructura de datos
- Estructuras de control
- Métodos de ordenación
- Otros lenguajes de programación

Conceptos avanzados ----------

- Manejo de eventos
- Trabajar con ficheros
- Buffers y streams
- Funciones en programación
- Manejo de errores y debugging

Control de versiones y testing ----------

- Trabajando con control de versiones
- Testing y prueba de código

Programación orientada a objetos ----------

- Paradigmas de programación
- Clases vs Objetos

Buenas prácticas en programación ----------

- Reglas para ser un as de la programación
- Principios ACID y persistencia de datos

Trabajando con frameworks ----------

- ¿Qué son los frameworks?
- Trabajar con frameworks

Introducción a las metodologías ágiles ----------

- Principios agile
- SCRUM: Organización efectiva de equipos
- Kanban: Coordinando grandes equipos desde lo visual.
- Otras metodologías agilen

Trabajando en equipo de desarrollo ----------

- Roles de equipo en entornos de programación
- Herramientas básicas

La programación como concepto

¿En qué consiste el proceso de programar?

Programar, a grandes rasgos, es decir a una máquina que hacer mediante instrucciones concisas, ordenadas y adaptadas a su nivel de procesamiento.
El arte de programar, además, consiste en descomponer problemas grandes en grupos de tareas más pequeños.

Los algoritmos son grupos de instrucciones que solucionan un problema o resuelven una tarea. Y el lenguaje de programación es el idioma con el que te comunicas con la máquina.

Trabajando con Pseudocódigo

Una de las formas de expresar dichos algoritmos es utilizando Pseudocódigo, que es una manera de expresar los pasos que va a realizar un programa de forma parecida a un lenguaje de programación.

Ventajas:	Desventajas:	Sintaxis:
Facilita la resolución de problemas	Falta de normas y estándares	No tiene sintaxis propia
Es independiente del lenguaje de programación	A veces es difícil traducir las soluciones a código formal	Conviene conocer la existencia de estructuras básicas.
Soluciones fácilmente implementadas con código	No ejecutable directamente	
Fácil de aprender y utilizar		

Lenguajes de programación

Tipos de Lenguajes de programación

Criterios de clasificación:

Lenguajes por nivel de abstracción:	**Lenguajes según sus propios propósitos:**	**Lenguajes según su método de ejecución**
Los lenguajes más próximos al binario son menos abstractos (lenguaje máquinas, lenguaje ensamblador, etc.) Los lenguajes menos próximos a la máquina y más cercanos a la comunicación humana se llaman lenguajes abstractos (Ruby, JavaScript, Python, etc)	Los lenguajes de propósito general (C++, Java) Lenguajes de propósito específico (SQL)	Los lenguajes compilados: necesitan traducir el lenguaje a código máquina. No necesitan el código fuente para ejecutarse. Lenguajes interpretados: leen y ejecutan el código líneo a línea y necesitan acceder al código fuente

Lenguajes según su paradigma de programación:

- Lenguajes imperativos: usan instrucciones o pasos. (Python)
- Lenguajes declarativos: expresan lo que quieren obtener (SQL)
- Lenguajes funcionales: usan funciones predefinidas (Haskell)
- Lenguajes Lógicos: usan la lógica matemática para relacionar elementos (Prolog)
- Lenguajes orientados a objetos: usan clases y objetos para representar al mundo real (JavaScript)

Introducción a los lenguajes de estilo y marcado:

Existen 3 tipos principales de lenguajes de marcas:

1. Marcado de presentación (da formato a un texto)

2. RFT
3. Marcado de procedimientos (está enfocado a la representación del texto)

LaTeX

Marcado descriptivo o semántico (se utiliza etiquetas para describir fragmentos de texto sin especificar cómo deben ser representados o en qué orden)

- XML
- SGML
- Lenguajes de marcado más conocido es HTML

Lenguajes de estilo (sirven para definir y crear la presentación de un documento que se ha escrito con algún lenguaje de marcado)

CSS

Aprender a programar con JavaScript

¿Qué es y por qué usar JavaScript como primer lenguaje?

JavaScript (JS) se utiliza para trabajar y manipular sitios web principalmente. Es un lenguaje de scripting interpretado y Case Sensitive.

Ventajas:

- Muy sencillo de aprender y manejar
- Resultados inmediatos
- Multiplataforma
- Fullstack

Nomenclatura:

- Llaves {}
- Corchetes []
- Paréntesis ()

Alertas - Alert	Alertas - Prompt	Alertas - Console.log
alert(“Hola mundo”); //abre una ventana del navegador e imprime un elemento.	prompt(“Dime tu nombre”); //abre una pequeña ventana en el navegador para el usuario introduzca datos	console.log(“mira aquí”); //Lanza un mensaje en la consola especialmente útil para testing código

Caracteres y strings

Para escribir una cadena de texto usamos “ ” o ‘ ’. Siempre que abrimos unas comillas debemos de cerrarla sino nos lanzará un error.

Booleanos, arrays y objetos en JavaScript

Los valores de un Booleano son True o False.

```
var x = 5;

var y = 6;

var igualdad = (x==y);

console.log(igualdad);

false
```

Los arrays sirven para guardar varios datos dentro de una misma variable.

```
var nombres = [”María”, “Felicia”, “Pepa”],

console.log(nombres [0])

María
```

Objetos que sirven para dar propiedades al valor de una misma variable. Esta variable se empieza a tratar en ese momento como una representación real del mundo.

```
var empleado = {

nombre: “Jessy”,

apellidos: “James”,

edad: 24,

email: “jessy@.com”;

console.log(empleado.email);
```

Variables en JavaScript

Siempre que trabajemos con datos usaremos variables para almacenarlos.

```
var cantidad = 10;
```

```
console.log(cantidad);

10
```

Ámbito de las variables

El ámbito de una variable es el espacio de programa donde esta será visible y accesible.

Var, Let y Const en JavaScript

- Las variables con **Var** pueden ser declaradas global o localmente. Podemos reasignarle valores e incluso podemos Re declararla.
- La variable **Let** está limitada a funcionar dentro de un bloque o porción de código que esté entre llaves. Let puede ser actualizada pero no puede ser Re declarada.
- La variable **Const** mantienen su valor en todo el código.

Tipos y usos de los operadores:

- Operadores de asignación: es el =
- Operadores aritméticos = +, -, *, /
- Operadores de incremento y decremento: Ej a++, a--

Los símbolos de igualdad:

= Asignación

== Igualdad

=== Igualdad estricta

!= diferente

Operadores de comparación

<, >, <=, =>

Operaciones con strings

Las cadenas de caracteres permiten realizar algunas operaciones de análisis, combinación y unión y también tienen algunas propiedades que facilitan enormemente trabajar con ellas.

Entre los métodos y propiedades de los strings:

- String.length: nos enseña la longitud de nuestro string en caracteres
- indexOf: en el que podemos encontrar una palabra dentro de un string

- lastIndexOf: nos devuelve la última ocurrencia de una palabra dentro de nuestro string
- search: que funciona muy parecido a indexOf pero no acepta valores de posición, pero si expresiones regulares como valores de búsqueda.
- Slice: nos permite extraer una parte del string y devuelve lo extraído en una nueva string
- Substring: es igual a silce salvo que no acepta valores negativos.
- substr: parece igual que substring pero el segundo parámetro no es la posición final sino la longitud de caracteres que debemos cortar
- replace: reemplazamos un valor específico por otro
- toUpperCase: sirve para convertir todos los caracteres de un string a mayúsculas
- toLowerCase: nos convierte todo en minúsculas
- concat: nos sirve para unir 2 o más strings
- trim: recorta los espacios en blanco
- split, charAt, charCodeAt

Estructura de datos

Son formas particulares de ordenar la información en programación.

Tipos de estructuras de datos:

Arrays:

- También conocidos como vectores o colecciones
- Una serie de elementos en orden lineal
- Se accede al elemento utilizando un número de índice
- La cuenta de posiciones comienza en cero

```
var objetos = [”sillas”, “llaves”, “vasos”,
“cama”];

var elegido = objetos[0];

console.log(elegido);

//nos aparece por consola “silla”
```

Vectores asociativos:

- Son arrays cuyos índices son cadena de texto (claves)

- Accedemos a cada elemento utilizando la clave

```
var moto ={ color: “negro”, CV: 25, custom: true};

var dato = moto[”color”];

console.log(dato);

//negro
```

Grafos

- Estructura de datos conectada compuesta por nodos
- Cada nodo tiene contenido y referencias a otros nodos

Árboles

- Es un tipo de árbol dirigido
- No se admiten ciclos
- Una colección de árboles es un bosque

Otras estructuras de datos:

- Pilas
- Clases
- Registros
- Variantes
- Conjuntos y multiconjuntos

Estructura de control

Permiten modificar el flujo de ejecución de instrucciones de un programa mediante el uso de iteraciones y condiciones

- Código Condicional

Se ejecuta o no en dependencia a la respuesta a una pregunta:

IF

- Todo debe ser evaluable como true o false
- Si el código es true, se ejecuta
- Si el código es false, no se ejecuta
- Usamos “else” para contemplar más de una condición

```
var balance = 0

if (balance > 0){

console.log(“el balance es positivo”);

else console.log(“el balance es negativo”);}
```

SWITVH

- Permite evitar usar demasiados “IF” en una condición múltiple
- Utilizamos “case” para evaluar cada caso
- “break” para terminar la evaluación de cada caso
- “default” para especificar que hacer fuera de los otros casos

```
switch(combustible){

case “diesel”:

console.log(“1.02$);

break;

case “super97”:

console.log(“1.45$);

break;

case “super 95”:

console.log(“1.67$”);
 break;

default:

console.log(“Lo que has escrito no esta”);}
```

Código iterativo

- Una iteración es la repetición de código dentro de un bucle
- Se repite una tarea un número definido de veces

- Lo que especificamos no es la sentencia a ejecutar, sino cuándo parar la ejecución
- Tener en cuenta el número de repeticiones y los valores inicial y final que tendrá el índice

Código iterativo - Estructura de un bucle

1. Inicialización del índice
2. Comprobación de la condición
3. Incremento del índice

WHILE

```
var cantidad = 0;

// crear indice

var i = 1;

// crear condición

while (i < 10) {

cantidad = cantidad + 100;

// incrementar índice

i++;

}

console.log("El valor final es" + cantidad);
```

- FOR

```
for(let i = 1; i < 10; i++)

{//inicializada, comprueba e incrementa a la vez //haz cosas}
```

- DO… WHILE

Lo que hacemos es mover la condición al final lo que está dentro del bloque se ejecuta al menos una vez, ya que la condición se comprueba después de la ejecución

```
var a = 1; // inicializa

do{ //tu código

a++; //incrementa

} while (a < 10); // comprueba la condición

console.log(a);
```

Metodos de ordenacion

Es un algoritmo que permuta los datos de una estructura en una secuencia determinada.

- Métodos de intercambio: Comparan los elementos entre si e intercambian su posición
- Métodos Quicksort: ordenan grandes cantidades de datos subdividiendo la estructura en otras más pequeñas.
- Métodos de selección: Toman el elemento más pequeño o más grande del vector y lo mueven a la posición 1
- Métodos de Inserción: Insertan elementos no ordenados del vector en vectores más pequeños ya ordenados
- Métodos de búsqueda: Se basan en localizar un elemento concreto en la estructura de datos

Búsqueda secuencial: Recorre el vector comparando elemento por elemento con el dato proporcionado

Búsqueda binaria: Compara el valor con el elemento en el medio del array, si no son iguales, la mitad en la cual el valor no puede estar es eliminada, y la búsqueda continua en mitad restante hasta que el valor se encuentre

Ordenación según permutación datos:

- Ordenamiento interno en la memoria del ordenador
- Ordenamiento externo fuera del ordenador (servidor en la nube)

Por prioridades de ordenación:

- Natural: tarda menos recibiendo una entrada de datos ordenada
- No natural: tarda el mínimo tiempo posible para una entrada inversamente ordenada

Otros lenguajes de programación

Java:

- Es simple y está orientado a objetos como JavaScript
- Gran cantidad de documentación y comunidad amplia y activa
- Multiplataforma, permitiendo crear apps móviles y de escritorio, y trabajar con webs a nivel de servido, entre otras.
- Fuertemente tipado, ideal para aprender

PHP

- Multiplataforma
- Seguridad ante amenazas
- Multitud de frameworks y librerías
- Código abierto

Python

- Data Science, IA, Machine Learning o desarrollo web, entre otros
- Comunidad amplia y activa. Muchas librerías y frameworks
- Simple y fácil de entender. Basado en automatización y scripting
- Permite crear aplicaciones de escritorio o interfaces gráficas
- Extensible y portable

Conceptos avanzados

Manejo de eventos:

Los eventos son acciones que se ejecutan en reacción a otras acciones que suceden en el HTML.

Ej: que se cliquee un botón

HTML te permite utilizar listeners o escuchadores que son manejadores de eventos como atributos a sus elementos.

Los elementos pueden cambiar su propio contenido utilizando this.innerHTML

<button onClick= “this.innerHTML = Date()”>¿Qué hora es)?</button>

- onClick = se produce un cambio cuando se cicla el elemento.
- onChange = produce el cambio cuando se cambia el elemento.
- onMouseOver = produce el cambio cuando se pasar por encima del elemento.
- onMouseOut = crea el cambio cuando sacamos el ratón del rango en HTML del elemento.
- onKeyDown = se produce cuando pulsamos una tecla en concreto
- onLoad = que se produce cuando el navegador ha terminado de cargar la página.

Trabajar con ficheros:

Node.js es una librería y entorno de ejecución para JavaScript que aumenta sus funciones y contiene varios medios para trabajar con archivos y directorios.

Módulo fs:

- fs es la abreviatura de File System
- Los usos comunes para este módulo son crear, leer, escribir, etc.

Leer archivos - fs.readFile

Teniendo un archivo llamado archivo1.html

```
//requerimos filesystem

var fs = require(‘fs’);

//leemos archivo

fs.readFile(‘archivo1.html’);
```

Crear archivo - fs.appendFile

```
//requerimos filesystem

var fs = require(‘fs’);

//añadimos contenido

fs.appendFile(‘archivo1.html’, ‘holita’);
```

Crear archivo - fs.open

w = modo escritura r= modo lectura

```
//requerimos filesystem

var fs = require('fs');

//añadimos contenido

fs.open('archivo1.html', 'w');
```

Crear archivo - fs.writeFile

Este método reemplaza el archivo y el contenido si existe. Si el archivo no existe se crea uno nuevo con el contenido que hemos especificado.

```
//requerimos filesystem

var fs = require('fs');

//añadimos contenido

fs.writeFile('archivo1.html', 'MariCarmen');
```

Actualizar archivos - fs.appendFile

```
//requerimos filesystem

var fs = require('fs');

//añadimos contenido

fs.appendFile('archivo1.html', 'Mas texto al final del documento');
```

Actualizar archivos - fs.writeFile

```
//requerimos filesystem

var fs = require('fs');

//añadimos contenido

fs.writeFile('archivo1.html', 'Mi texto para remplazar');
```

Borrar archivos -fs.unlink

```
//requerimos filesystem

var fs = require('fs');

//añadimos contenido

fs.unlink('archivo1.html');
```

Renombrar archivos - fs.rename

```
//requerimos filesystem

var fs = require('fs');

//añadimos contenido

fs.rename('archivo1.html', 'archivo2.html');
```

Buffers y streams

Un buffer es un espacio en memoria en el que se almacenan datos de manera temporal, normalmente para uso concreto, como evitar que un programa se quede sin datos en una transferencia irregular o lenta.

Módulo fs:

Son datos crudos en hexadecimal optimizados para ser transportados de un punto de memoria a otro.

Las ventajas de trabajar con buffers radican en que nos permiten hacerlo con los datos en crudo

Ejemplo - Crear un buffer de longitud 15

let buffer = Buffer.alloc(1);//Le pedimos que guarde 1 espacio

console.log(buffer);//Buffer 00>00 es un espacio dentro del buffer

// <Buffer 01 02 03 05> - en cada espacio hay un valor

let buffer = Buffer.from([1,2,3,5]); // <Buffer 01 02 03 05>

Un stream podrá definirse como una cinta transportadora que comunica dos puntos para intercambiar datos. Los datos mencionados se descomponen en pequeñas piezas conocidas como chunks (porciones).

Stream:

Su uso se encuentra en sistemas donde la información es tan grande que el canal no lo soporta y, por tanto, la información se descompone en trozos más pequeños (chunks).

Ejemplo Stream:

const fs = require ('fs');

let data = ' ';

let readbleStream = fs.createReadStream(_dirname + '/input.txt');

//chunk → trocito

readableStream.on('data ´, function (chunk) {

console.log(chunk); // <Buffer 73 6f 79 0d 0a 75 6e 20 65 72 ...>

})

Funciones en programación

Es un bloque de código designado para realizar una tarea específica.

```
function lafuncion(a, b){

return a*b;

}

// llamó a la función

lafuncion(4,10);

40
```

- Manejo de errores y debugging

Diferencia entre errores y excepciones:

Un error es lo que se produce en un programa desencadenando un resultado indeseado. Estos no pueden ser manejados en el código.

Una excepción es una situación específica que se puede anticipar y controlar en el código del programa.

Objeto Error

Tiene 3 propiedades:

- error.name // nombre del error (Error:)
- error.message // mensaje del error (Ha habido un error)
- error.stack // muestra la pila de procesos y en que archivo ocurrió el error.

Para trabajar con errores y excepciones también tenemos la sentencia Try…catch, que marca un bloque de instrucciones y específica la respuesta que debe ser lanzada con la excepción que se produzca.

try {

throw new Error (“We made an error”); // instanciamos directamente el objeto error

} catch (error) {// lo obtenemos de arriba

console.log (error.message); // We made an error

}

Debugging:

El debugging o depuración de código sirve para diagnosticar y corregir estos errores.

Métodos de depuración

- console.log(): para revisar momentos puntuales en el código
- breakpoints: para comprobar la ejecución
- debugger: detiene la ejecución y llama a la función de depuración

Control de versiones y testing

Control de versiones

Los sistemas de control de versiones (SCV) son herramientas de software que ayudan a gestionar los cambio en el código fuente.

Características de los SCV

- Seguimiento de modificación en el código
- Control de errores en las versiones
- Control de cambios y compatibilidades

Ejemplos de SCV

- GIT
- Mercurial
- SVN

GitHub, GitLab o Bitbucket son repositorios remotos basado en GIT

Controlar versiones con GIT

- Instalar GIT en tu maquina
- Crear un repositorio nuevo, abrir e inicializar. Añadir repositorio remoto
- add (añadir al área de preparación) & commit (preparar la subida del código al repositorio remoto)
- Subir todo al repositorio remoto

Controlar versiones con GIT - flujo:

git init // inicializa el repositorio local

git remote add URL // añade el repositorio remoto

git add . // añade todos tus archivos al área de preparación

git commit -m “mesagge” // incluye el archivo en el head

git push // envía los cambios a tu repositorio remoto

- Testing y pruebas de software:

Es la realización de pruebas sobre el software, con el fin de obtener información acerca de su calidad y la detección de errores.

Tipos de pruebas:

- Pruebas manuales: son aquellas en las que se prueba la navegación por la aplicación, interactuando con la misma
- Pruebas automáticas: son aquellas donde se emplean herramientas para realizar las pruebas

Pruebas unitarias:

Tomar cada unidad de software por separado y analizar fuera de su contexto para validar que hace lo que tiene que hacer.

- Refactorización del código
- Facilita la integración con otras piezas de software
- Documentación del código

Pruebas unitarias:

- Escribir los test
- Ejecutar los test y mostrar los resultados

Pruebas de integración:

- Complementan a las pruebas unitarias
- Válida la interacción de un sistema completo

- Comprobar la fiabilidad de la comunicación entre componentes.

Pruebas estáticas y dinámicas

- Estáticas: se realizan sin ejecutar el código
- Dinámicas: analizan la performance de la aplicación ejecutando el código.

Pruebas ESRE

ESRE: Son las Pruebas contra Especificaciones de Requerimientos

Las realizan los betatesters

Clasificación de pruebas: funcionales

Se basan en la ejecución, revisión y actualización de las funcionalidades.

Primero: pruebas unitarias, de integración y de sistema.

Después: las pruebas de humo, Alpha, beta y de aceptación

Clasificación de pruebas: no funcionales

- Verifican requisitos establecidos por el cliente
- Primero las de compatibilidad, seguridad, estrés, usabilidad y rendimiento
- Después las de escalabilidad, mantenibilidad, instabilidad y portabilidad

Herramientas para realizar pruebas:

- Selenium
- Soapui
- LoadRunner

Programación orientada a objetos

Paradigmas de programación

Los paradigmas son estilos diferentes de abordar la resolución de problemas de la programación. A si mismo delimita el enfoque que se va adoptar en la forma de escribir código.

Paradigma imperativo:

- Describe cómo debe realizarse el cálculo y no por qué.

- Estos programas se componen de un conjunto de sentencias que van cambiando su estado y se ejecutan según un control de flujo específico que se modifica y modifica al estado el programa.
- Son secuencias de comandos que ordenan acciones en la computadora.
- Las variables son celdas que contienen datos o referencias a esos datos y pueden ser modificadas representando el estado del programa en cada momento.

Paradigma Lógico:

- Se basa en la lógica de predicados de primer orden.
- Los programas se componen de hechos predicados y relaciones.
- Un problema se modela con enunciados de lógica de primer orden. Se realiza una evaluación basada en la resolución SLD, que anua la unificación y el bug tracking y la ejecución consiste en la resolución de un problema de decisión.
- Los resultados se obtienen mediante la instanciación de variables libres.

Paradigmas de las Funciones:

- En que la evaluación se realiza por reducción funcional.
- Se utilizan técnicas como la recursividad, parámetros acumuladores, CPS o mónadas.
- Los programas se componen de funciones, es decir de implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.
- Las funciones son elementos de primer orden en los modelos de cómputo de cálculo lambda y de la lógica combinatoria.

Paradigma orientado a Objetos:

- El comportamiento del programa es llevado a cabo por objetos que son entidades que representan elementos del problema a resolver y tienen atributos y comportamiento.
- Se centran en la estructura y organizan de los programas y son compatibles con las fundamentales
- La programación orientada a objetos expresa un programa como un conjunto de objetos que colaboran entre ellos para realizar tareas. Esto permite que los programas y módulos más fáciles de escribir, mantener y reutilizar también se puedan volver a usar.

Paradigma declarativo:

- Es opuesto al imperativo. En este caso el programa describe los resultados esperados sin listarte los pasos a llevar a cabo.

- Describe también cómo se debe calcular, pero no te explica el cómo.
- No existe un orden de evaluación prefijado. En este caso las variables son nombres asociados a definiciones y una vez distanciadas, son inmutables. No existe una sentencia de asignación y el control de flujo suele ser asociado a la composición funcional, la recursividad o las técnicas de escritura y unificación.

Clases vs. Objetos

Formalmente, en el lenguaje orientado a objetos conocemos que existen las clases que son una definición formal de características y los objetos que son instancias de una clase.

Esto quiere decir que una clase hace las veces de plantilla para poder crear varios objetos a partir de ella.

Buenas prácticas en programación

Buenas prácticas:

- Nombres de variables y funciones autodescriptivos
- Evita usar variables globales que permitan hoisting
- Comenta el código
- Mantén el código simple
- Portabilidad
- Escalabilidad
- Reusabilidad
- Construcción de código
- Testing
- Debugging

Principios ACID

Las bases de datos juegan un papel importante en el desarrollo de software.

Tan primordial como entender como trabajar con datos en otros tipos de estructura, la forma de tratarlos en las estructuras de bases de datos es especialmente útil.

Principios de ACID:

Se aplican en transacciones, es decir, en intercambios de información entre elementos de la base de datos.

ACID: Atomicity, Consistency, Isolation, Durability (Atomicidad, Consistencia, Aislamiento, Durabilidad).

ATOMICIDAD: si una transacción consta de varios pasos, deben ejecutarse todos. Si un paso falla, toda la transacción falla.

CONSISTENCIA: las transacciones llevan de un estado valido a otros estados válidos. Todos los datos deben ser válidos.

AISLAMIENTO: La ejecución de una operación no puede afectar a otras. Define como y cuando los cambios de una operación se muestran al resto.

DURABILIDAD: Una vez se realiza la operación, está persistirá y no se podrá deshacer, aunque falle el sistema.

Trabajando con frameworks

¿Que son los frameworks?

Es un conjunto de archivos y pautas que definen la estructura y metodologías sobre cómo desarrollar un producto de software.

Su objetivo es el desarrollo ágil de aplicaciones mediante la aportación de librerías y/o funcionalidades ya desarrolladas.

Frameworks:

Frameworks de JavaScript: Angular, Ember, Vue, React

“ ” de PHP: Laravel, Codelgnigter, Symfony

“ ” Java: Spring, MVC, JFS, Struts

Una librería es uno o varios archivos escritos en un lenguaje de programación determinado que proporcionan diversas funcionalidades.

Librerías:

- Librerías de JavaScript: Moment.js, Elevator.js, AOS, Chart.js
- Librerías de PHP: pChart, Upload, Underscore, Carbon, Krumo
- Librerías de Java: Hadoop, GWT, Guava
- Librerías de Python: Colorama, Kivy, PyWeather, PyQt

Trabajar con frameworks:

Ventajas	Desventajas
Estructura y orden predeterminados	Tiempo de aprendizaje
Reutilización del código	Versiones inestables
Agilidad en el desarrollo	Menor rendimiento
Menor coste en el desarrollo	Código sin utilizar
Buenas prácticas de desarrollo y uso de patrones	Elección de framework
Minimizar errores y mayor facilidad para solucionarlos	

Facilidad para encontrar librerías	
Colaboración con otros desarrolladores	
Facilita el mantenimiento	

Introducción a las metodologías ágiles

Principios ágiles:

El desarrollo ágil se basa en la generación de software iterativo e incremental, que evoluciona en base a requisitos cambiantes.

Características principales Agile:

- Software iterativo e incrementar
- Planificación, análisis de requisitos, diseño, codificación, pruebas y documentación en cada ciclo
- Siempre entregas de software funcional y sin errores
- Comunicación cara a cara antes que por escrito.

Cuando se comenzó a hablar de “metodologías ágiles” se creó el “Manifiesto ágil”.

VALORES AGILE:

Individuos e interacciones antes que procesos y herramientas.

Software funcionando sobre documentación extensiva

Colaboración con el cliente antes que la negociación contractual

Respuesta ante el cambio sobre el plan.

Principios Agile (I)

- Entrega temprana y continua de software con valor
- Se acepta y abraza el cambio continuo de requisitos.
- Entrega software funcional frecuentes (15 días a 2 meses)
- El equipo de desarrollo trabaja en conjunto con el cliente.
- Los proyectos tienen como centro a las personas

Principios Agile (II)

- El método más eficaz de comunicación es cara a cara
- Medida principal de progreso, entregas de software funcionando.
- Desarrollo sostenido en el tiempo.
- Excelencia técnica y buen diseño.

- Maximizar la cantidad de trabajo no realizado

Principios Agile (III)

- Equipos autoorganizados.
- Perfeccionamiento del trabajo en base a la efectividad

SCRUM: es un marco de trabajo en el que se aplican normas para la organización colaborativa, definiendo roles para desarrollar un producto de calidad con la mayor productividad.

Ventajas:

- Transparencia
- Seguimiento de tareas
- Adaptación
- Comunicación con el cliente

Roles del equipo:

- Product Owner
- SCRUM master
- Equipo de desarrollo

Flujo de trabajo:

- Sprint
- Planificación del sprint
- SCRUM diario
- Revisión del sprint
- Retrospectivita del sprint

Documentos:

- Product Backlog
- Sprint Backlog
- Burn down chart
- Definition of done

Kanban:

Es una metodología ágil para gestionar flujos de trabajo. El objetivo es que vayan definiendo tareas para completar la entrega al cliente.

Principios Kanban

- Empezar con lo que se va hacer ahora
- Búsqueda e implementación de cambios incrementales

- Respetar los procesos, responsabilidades y cargos actuales
- Liderazgo a todos los niveles

Prácticas centrales:

- Visualización de las tareas
- Limitar el trabajo en curso
- Gestión de flujo y mediación del tiempo

Ventajas:

- Eliminación de procesos innecesarios
- Equipo motivado
- Detección de fallos y mayor flexibilidad ante los cambios.
- Aumento de la productividad y mejor gestión del tiempo

Otras metodologías agiles:

Design Sprint:

- Prototipo de prueba en 5 días
- Proceso por días
- Prueba del prototipo con 5 personas del público objetivo

Feature-driven development

- Desarrollo basado en funcionalidades
- Dueños de las clases vs programadores jefe
- 5 pasos de desarrollo

Lean Software Development

- 7 principios de desarrollo
- Piensa en grande
- Actual en pequeño
- Equivócate rápido
- Aprende con rapidez

Extreme Programming (X)

- Adaptabilidad como pilar
- Se abrazan los cambios de requisitos
- Desarrollo iterativo
- Énfasis en la simplicidad general de los proyectos.

Trabajando en equipos de desarrollo

Roles de equipo en entornos de programación

Líder de proyecto:

- Cara visible del proyecto ante al cliente
- Coordina equipos
- Responsabilidad sobre los costes.

Comercial:

- Consigue clientes para la empresa
- Altas dotes sociales
- Primer contacto con el cliente

Architect:

- Encuentra soluciones técnicas optimas
- Documenta los equisetos técnicos
- Toma decisiones respecto a los problemas técnicos

Developer

- Recibe la documentación del architect
- Implementa las soluciones encontradas según los requisitos

Tester

- Asegura los requisitos definidos

- Comprobar la estabilidad del proyecto
- Informa sobre fallos

Cliente

- Dueño final del producto
- Establece las necesidades a nivel empresa

Herramientas básicas

- Editor de código
- Herramientas de versionado (GIT)
- Herramientas de acceso FTP
- Servicio de cloud computing

Introducción a las habilidades profesionales en el desarrollo de software

La vida de un programador ----------

- ¿Qué hacen los programadores?
- Una semana trabajando en programación
- Los mayores retos de los programadores
- Mantenimiento de tus habilidades al día

Desarrollo de software ----------

- ¿Qué es el desarrollo de software?
- El proceso de desarrollo de software
- Exploración de las distintas funciones en el desarrollo de software
- Herramientas utilizadas para el desarrollo de software
- Cómo hacer llegar el software a los clientes

Herramientas de trabajo ----------

- Exploración de los sistemas de control de versiones
- Recorrido por varios servicios de repositorio de código
- Introducción a las librerías y los frameworks
- Estudio de los diferentes tipos de IDE
- Aprende dónde obtener ayuda

Lenguajes de programación ----------

- ¿Qué es un lenguaje de programación?
- Componentes básicos de un lenguaje de programación
- ¿Qué es el código fuente?
- Ejecución del código fuente
- Exploración de variables
- Declaraciones y expresiones básicas

Exposición de Python ----------

- Introducción a Python
- Sintaxis básica de Python
- Guardar datos con Python
- Toma de decisiones con Python
- Funciones en Python

Trabajo con Python ----------

- Introducción a la programación orientada a objetos
- Creación y uso de una clase
- Cómo organizar tu código
- Añadir módulos a los programas
- Como se utiliza Python en el mundo real

Explorando diferentes lenguajes ----------

- Declaraciones y expresiones en los distintos lenguajes
- Exploración de variables en los distintos lenguajes
- Toma de decisiones en el código a través de los lenguajes
- Puestos tecnológicos y sus lenguajes de programación

Tu primer trabajo en programación ----------

- Explora el proceso de contratación
- Creación del portafolio técnico
- Redacción del currículum vitae
- Preparación de las entrevistas técnicas
- Cómo tener éxito al empezar

La vida de un programador

¿Qué hacen los programadores?

Los programadores informáticos diseñan, desarrollan y prueban software.

Una semana trabajando en programación.

- **Lunes:**

Prepara la semana entrante

Reunirse con los equipos para discutir las prioridades

- **Martes:**

Investigar

Escribir código

- **Miércoles:**

Depurar errores

- **Jueves:**

Ejecutar el código

Arreglar errores (Debug)

Escribir pruebas

- **Viernes:**

Mostrar tu trabajo al equipo

Depuración: El proceso de encontrar y eliminar errores o fallos en el código.

Los mayores retos de los programadores

- Asignar nombres: Cuando se escribe código, cada clase, archivo y variable puede tener un nombre.
- Trabajar con código heredado.
- Hacer estimaciones

Factores de estimación:

- ¿Cuántas horas totales de codificación tendrá?
- ¿Qué tipo de investigación es necesaria de antemano?
- ¿Sera fácil o difícil escribir pruebas para esta funcionalidad?
- ¿Tiene que aportar documentación a otra persona?

Mantenimiento de tus habilidades al día.

¿Cómo mantener nuestras competencias al día?

- Suscribirse a un boletín de noticias
- Suscribirse a un podcast
- Trabajar en un proyecto paralelo

Desarrollo de software

¿Qué es el desarrollo de software?

Es el conjunto de instrucciones o programas que indican a un ordenador lo que debe hacer

- Software de sistema
- Software de aplicación
- Software de programación

El proceso de desarrollo de software

Ciclo de vida del desarrollo de software (SDLC)

Fases:

- Recopilación de requisitos
- Análisis
- Diseño
- Codificación
- Pruebas
- Despliegue
- Mantenimiento

Exploración de las distintas funciones en el desarrollo de software

- Equipo de desarrollo: Partes interesadas de la empresa
- Responsable de producto (PM)
- Diseñadores de experiencia de usuario
- Diseñadores de interfaz de usuario
- Programadores: Programadora front-end, back-end y full-stack
- Ingeniera de control de calidad

Herramientas utilizadas para el desarrollo de software

- Pizarras blancas
- Software de colaboración en línea
- Software de gestión de proyecto
- Entornos de desarrollo integrado
- Papel y lápiz

Cómo hacer llegar el software a los clientes

Despliegue de software: Entrega del software a los usuarios previstos

Despliegue de la aplicación web	Despliegue de la aplicación móvil
El código se prueba para detectar errores	El código se empleita en el proyecto existente
El código se empaqueta en el proyecto existente	Se produce el nuevo ejecutable
Se produce un nuevo ejecutable	Se entrega el nuevo ejecutable al QA

El ejecutable se copia en Internet	El QA comprobar manualmente si hay errores
	El ejecutable se copia en la tienda de aplicaciones

Exploración de los sistemas de control de versiones

Sistemas de control de versiones (SCV), son herramientas que ayudan a los equipos de software a gestionar los cambios de código

Ventajas:

- Historial de cambios
- Trabajo simultáneo
- Solución de problemas

GIT: cuenta con un modelo de ramificación. Un modelo de ramificación define las reglas para hacer cambios asiladas en el código y luego incorporarlos al proyecto principal.

Recorrido por varios servicios de repositorio de código

Servicios de alojamiento de repositorios: son aplicaciones web de terceros que envuelven y mejoran un sistema de control de versiones

El servicio de alojamiento más popular para GIT es GitHub

Características:

- Permite almacenar el código en la nube
- Tiene un registro de paquetes de software
- Tiene incorporada la revisión del código
- Los flujos de trabajo de GitHub permiten definir los requisitos que se deben aprobar antes de que se envíe a los clientes una nueva versión del software.

Introducción a las librerías y los frameworks

Librería: código que otra persona ha escrito y verificado

- Ktor: se utiliza con el lenguaje de programación Kotlin.
- NumPy: es una librería de Python.

Frameworks: sirve como modelo de cómo configurarse y desarrollarse el proyecto de software.

- NextJs: es un frameworks para el desarrollo de aplicaciones web

Estudio de los diferentes tipos de IDE

Entorno de desarrollo integrado (IDE):

Aplicación que proporciona herramientas especiales para que los programadores escriban, depuren y compilen código.

Aprende dónde obtener ayuda

- Documentación oficial de tu lenguaje de programación
- Buscar stackoverflow.com donde puedes copiar/pegar el mensaje de error y luego buscar preguntas similares a tu situación
- Consultar en comunidades

Lenguaje de programación

¿Qué es un lenguaje de programación?

Es una forma de proporcionar instrucciones a un ordenador escritas en texto y basadas en un conjunto de reglas

Componentes básicos de un lenguaje de programación

Un lenguaje de programación puede dividirse en su sintaxis (reglas) y en su semántica (significado).

¿Qué es el código fuente?

Son instrucciones informáticas legibles escritas por los programadores.

Ejecución del código fuente

Las instrucciones son el resultado de la ejecución de nuestro código.

Exploración de variables

Una variable es un marcador de posición para un valor desconocido.

Cuando ejecutamos un programa el ordenador nos da un espacio un espacio en su memoria donde podemos poner los datos que queremos usar como referencia más adelante.

Declaraciones y expresiones básicas

Operadores son símbolos que indican al ordenador que realice una acción con alguna entrada.

Expresiones son operaciones que se descomponen en un solo valor.

Palabra clave es una palabra reservada que significa algo especial para el lenguaje de programación en cuestión.

Exposición de Python

Introducción a Python

Python fue diseñado para ser un lenguaje de propósito general.

- Aplicaciones web
- Análisis científicos
- Crear juegos
- etc.

Sintaxis básica de Python

Print: imprime lo que queremos que se vea en pantalla

Python ignora las líneas vacías.

Guardar datos con Python

Declaramos las variables y le asignamos un valor

```
#La puntuación del usuario

puntuación = 750

print(puntuación)
```

Toma de decisiones con Python

Expresión condicional o booleana: Expresión que se descompone en verdadero o falso.

```
if 5 < 6:

print("Yes, 5 is less than 6")
```

Funciones en Python

DRY es un principio básico del desarrollo de software que significa "No te repitas".

Una función es un bloque de código agrupado con un nombre.

```
def saludar (nombre):
```

```
print(f”Hola, {nombre}!)

saludar(“Tiffany”)
```

Trabajo con Python

Introducción a la programación orientada a objetos

Programación orientada a objetos: Utilizar código para representar cosas y situaciones del mundo real.

Una clase es un modelo de cómo se crean otros objetos.

Creación y uso de una clase

Python admite la creación de clases con la palabra class.

```
class Cachorro():
 def __init__(self, nombre, juguete_favorito):

self.nombre = nombre

self.juguete_favorito = juguete_favorito

def jugar(self):
 print(self.nombre + “está jugando con el” + self.juguete_favorito)

marble = Cachorro(“Marble”, “oso de peluche”)

marble.jugar()
```

Cómo organizar tu código

Un módulo es un archivo que consiste en código Python. Pueden definir funciones, clases, variables y mucho más.

Ventajas de dividir el código

- Simplicidad
- Mantenimiento
- Reutilización

Añadir módulos a los programas

Para empezar, utilizaremos la instrucción <import> y luego el nombre del módulo

```
import random
```

```
números = [1,2,3,4,5]

random.shuffle(números)

print(números)
```

Como se utiliza Python en el mundo real

- Django es un frameworks
- Blender (animación)
- scikit-image
- Pandas

Explorando diferentes lenguajes

Declaraciones y expresiones en los distintos lenguajes

Lenguajes de programación más populares:

- Python
- C
- Java
- C++

Java, Python y C++ heredan ideas de C

Declaraciones en C

```
#include <studio.h>

int main() {

printf("¡Hola, mundo!");

return 0;
}
```

Declaraciones en Java

```
public class Hola {

public static void main (String [] args) {

system.out.println("¡Hola mundo!");
 }

}
```

Declaraciones en C++

```
#include <iostream>

using namespace std;
int main () {
```

```
cout <<"¡Hola Mundo!" << endl;

return 0;

}
```

Exploración de variables en los distintos lenguajes

Variables en Java

```
int age = 37;

system.out.println(edad); //37
```

Variables en JavaScript

```
const base = prompt("Introduce la base de un triángulo:");
const height = prompt("Introduce la altura de un triángulo:");
const área = (base * height) / 2;

console.log("El área del triángulo es ${área}");
```

Toma de decisiones en el código a través de los lenguajes

Python	Java	Ruby

if 5 < 6:
print("si")
	if (5 < 6) {
	System.out.println("Si");
	}
		if 5 < 6
		puts "si"
		end

Puestos tecnológicos y sus lenguajes de programación

- Desarrollo de aplicaciones de software
- Desarrollo web
- Administración de sistemas de red
- Ingeniería de control de calidad de software
- Desarrollo móvil

Tu primer trabajo en programación

Explora el proceso de contratación

El embudo de contratación

- Revisión del currículum
- Llamada telefónica con el seleccionador
- Llamada técnica con un experto
- Visita a la empresa

Creación del portafolio técnico

- Trabajar en proyectos de muestra
- Contribuir al código abierto
- Prácticas
- Blog personal

Redacción del curriculum vitae

Promocionarte

- Incluye proyectos relevantes
- Muestra tu historial de trabajo
- Documenta las tecnologías y las librerías con las que está familiarizado
- Mantén la información sucinta y centrada en las métricas

Preparación de las entrevistas técnicas

- Ejercicio de algoritmo
- reto de codificación para llevar acsa
- sesión de codificación en vivo

Investiga las tecnologías que usa el equipo

Cómo tener éxito al empezar

- ten curiosidad
- tomate tu tiempo para practicar
- no lo hagas solo

Fundamentos de la programación: Más allá de lo básico

Introducción ----------

- Ampliado tu conocimiento de los fundamentos de programación
- Conocimientos previos
- Configurando tu entorno de programación

Colecciones ----------

- Entendiendo las colecciones
- Creando colecciones simples
- Creando colecciones complejas
- Trabajando con colecciones
- Colecciones en otros idiomas de programación

Iteración ----------

- Introducción a la iteración
- Iterar a través de las colecciones
- Iterar hasta un punto determinado

Utilizando código externo ----------

- Comparando tipos de código externo
- Trabajando con un módulo
- Entendimiento librerías y frameworks

Trabajando con strings ----------

- Combinando y manipulando strings
- Haciendo búsquedas en cadenas de caracteres
- Creando expresiones regulares

Planificando una aplicación ----------

- Eligiendo un estilo de código
- Escribiendo pseudocódigo

Entrada y salida ----------

- Introducción a entradas y salidas de datos en la programación
- Trabajando con el fichero de entrada y salida

Corrigiendo el código ----------

- Introducción a la corrección de código (debugging)
- Corrigiendo código en un IDE
- Interpretando los mensajes de error
- Corrigiendo código si errores obvios
- Creando una prueba unitaria de código

Código orientado a objetos ----------

- Introducción a la programación orientada a objetos
- Utilización de clases
- Creando clases y objetos personalizados

Tópicos avanzados ----------

- Tópicos avanzados en programación
- Gestión de memoria a través de diferentes lenguajes
- Introducción a los multihilos
- Introducción algoritmos

Introducción

Ampliado tu conocimiento de los fundamentos de programación

Colección: Agrupación de varios elementos con un mismo nombre, llamado variable.

Conocimientos previos

- Variables
- Funciones
- Lógica condicional

Configurando tu entorno de programación

- Descargar Python
- Configurar visual studio code

Colecciones

Entendiendo las colecciones

La agrupación de datos en programación se llama Colección.

- Utiliza la estructura del código para indicar que varios datos están relacionados
- Evita tener que crear un gran número de variables en el código
- ofrece una sintaxis simplificada

Creando colecciones simples

Lista: una colección que agrupa datos en un orden determinado y con un nombre específico

```
#ciudad1 = “San José”

#ciudad2 = “Ciudad de México”
#ciudad3 = “Madrid”

#ciudad4 = “Buenos Aires”

ciudades = [

”San José”,

“Ciudad de México”,

“Madrid”,

“Buenos Aires”

]
```

Creando colecciones complejas

Diccionarios: te permiten guardar información relacionada

```
#Animales

#pez = “Pez Espada”

#anfibio = “Rana”

#reptil = “Tortuga”

#ave = “Avestruz”

#mamífero = “Elefante”

animales (

“pez” : “Pez Espada”,

“anfibio” : “Rana”,

“reptil” : “Tortuga”,

“ave” : “Avestruz”,

“mamífero” = ”Elefante”

)
```

Trabajando con colecciones

La numeración del índice comienza en 0

```
#ciudad1 = “San José”

#ciudad2 = “Ciudad de México”
#ciudad3 = “Madrid”

#ciudad4 = “Buenos Aires”

ciudades = [

”San José”,

“Ciudad de México”,
```

```
“Madrid”,
“Buenos Aires”
]
print(ciudades[3])
```

```
#Animales
#pez = “Pez Espada”
#anfibio = “Rana”
#reptil = “Tortuga”
#ave = “Avestruz”
#mamífero = “Elefante”
animales (
“pez” : “Pez Espada”,
“anfibio” : “Rana”,
“reptil” : “Tortuga”,
“ave” : “Avestruz”,
“mamífero” = ”Elefante”
)
print(animales[”ave”])
```

Colecciones en otros idiomas de programación

En Python se pueden agrupar distintos tipos de datos, que por ejemplo C++ NO.

se pueden modificar y eliminar datos este tipo de colecciones de denomina mutable.

Tuplas es una colección inmutable.

Iteración

Introducción a la iteración

Iteración: Repetición del mismo procedimiento varias veces hasta que se alcanza un punto final específico.

Bucle: código que itera desde el principio al final del proceso y luego vuelve empezar.

Sintaxis:

- Especifica los datos
- Describe que sucederá con los datos en cada iteración
- Indica cuando debe para el bucle

Iterar a través de las colecciones

En Python puedes crear un bucle utilizando la palabra clave for.

```
ingredientes = [
“sal”,
“pimienta”,
“cebollino”,
“chile”,
]
for ingrediente in ingredientes:
print(ingrediente)
print(“¡Así me gustan las tortillas!”)
```

Iterar hasta un punto determinado

```
print(“Contando de 5 en 5 hasta 100: ”)
i = 5
while i <= 100:
print(i)
i += 5
print(“Cuenta terminada”)
```

Utilizando código externo

Comparando tipos de código externo

Módulo: Archivo de Python que contiene código, como variables o funciones.

Paquete o librería: Utilización de varios módulos juntos distribuidos y utilizados como grupo.

Framework: El conjunto de código no se utiliza a la vez, sino de manera específica.

Trabajando con un módulo

```
def mult (x,y) =

print(f”(x) + (y) = (x * y)”)
```

```
import moduloprueba

moduloprueba.mult(10,5)
```

Entendimiento librerías y frameworks

Librerías en Python

- TensorFlow
- Pandas
- NumPy
- SciPy

Frameworks de Python

- Django
- Flask

Trabajando con strings

Combinando y manipulando strings

Concatenación: Varias cadenas se combinan en una única cadena.

```
valor = input(“Ingresa un número: ”)

print(valor + “ Es mi número favorito”)
```

Haciendo búsquedas en cadenas de caracteres

- .capitalize : hace que la primera letra este en mayúsculas
- .find : busco una palabra en una cadena de caracteres
- [9:]Sustraer un valor
- float : convierte una linea de caracteres en numero

Creando expresiones regulares

Expresión regular: Patrón que describe un texto que deseamos buscar dentro de una cadena de caracteres.

- letras
- cifras
- caracteres especiales

- un texto se encierra entre / /
- \d representa un digito
- /w representa un carácter de palabra como una letra
- . el punto representa cualquier carácter
- + el signo de mas representan una o mas ocurrencias del patrón anterior
- el asterisco indica 0 o más ocurrencias
- ? indica entre 0 y 1 ocurrencia

```
import re

codigo_cinco_digitos =”12345"

codigo_nueve_digitos = “12345-6789”

numero_telefonico = “123-456-7890”

regrex_cinco_digitos = f?d{5}*

print(re.search(regrex_cinco_digitos, codigo_cinco_digitos))
```

Planificando una aplicación

Eligiendo un estilo de código

Guías de estilo: Documentación sobre modelos según los que programar

Escribiendo pseudocódigo

Pseudocódigo: Descripción de lo que intentas hacer en un lenguaje sencillo.

Entrada y salida

Introducción a entradas y salidas de datos en la programación

El flujo de datos lo conocemos como Entrada y Salida (E/S)

Trabajando con el fichero de entrada y salida

En Python es posible obtener información del usuario, utilizando el método entrada. Este método detiene la ejecución del programa para darle al usuario la oportunidad de escribir una respuesta.

Puede usar otros métodos de entrada como abrir un archivo desde su computadora

Corrigiendo el código

Introducción a la corrección de código (debugging)

Depuración: Identificación y arreglo de fallos.

- Error de Sintaxis
- Error en tiempo de ejecución
- Error de lógica

Corrigiendo código en un IDE

- Resaltado de sintaxis

- Autocompletado
- Linting

Interpretando los mensajes de error

- Cuando está subrayado es un error.

Creando una prueba unitaria de código

- Casos de prueba: Comandos o scripts diseñados para probar una situación específica.

Código orientado a objetos

Introducción a la programación orientada a objetos

Los paradigmas nos indican cómo estructurar el código de una aplicación, por ejemplo, la Programación orientada a objetos

Cada objeto tiene atributos y comportamientos. Cada atributo son datos o características que tiene el objeto y cada comportamiento es algo que el objeto puede hacer.

Los atributos se denominan “propiedades” y los comportamientos “métodos”.

Se crean objetos utilizando un modelo conocido como clase. Una clase describe los atributos y el comportamiento que debe tener un objeto

Utilización de clases

```
vueltas = [

“cara”,

“cruz”,

“cara”,

“cruz”,

]

print(vueltas.count(“cara”))

print(vueltas.pop())
```

Tópicos avanzados

Gestión de memoria a través de diferentes lenguajes

Almacenamiento:

- Drive o disco duro: Programas, Datos
- Memoria Caché: Ejecutar código, Resultados
- Gestión de memoria: Código que decide lo que se guarda en la memoria y lo que no.

Recolección de basura: Proceso automático de gestión de memoria que hace un seguimiento de los elementos que no se necesitan y los elimina.

Introducción a los multihilos

Cada tarea corriendo en paralelo se conoce como hilo y el nombre que se le da a la técnica de escribir código que ejecuta varios procesos al mismo tiempo se conoce como subprocesos múltiples.

Cada subproceso en un programa requiere capacidad de procesamiento y espacio de memoria adicional en la computadora donde se ejecuta el código

Introducción algoritmos

Algoritmo: Conjunto de instrucciones para describir el mismo resultado.