



**Universidad Nacional Autónoma  
de México**  
**Facultad de Ingeniería**



**Asignatura: Estructura de Datos y Algoritmos 1**

**Actividad 1: Acordeón**

**Alumna: Hernández Vázquez Daniela**

**Profesor: M.I. Marco Antonio Martínez Quintana**

**Fecha: 26/02/2021**

**2021-2**



## Actividades:

- Realizar un acordeón del Lenguaje C con lo visto en las prácticas de Laboratorio de Fundamentos de Programación.
- Investigar un lenguaje de programación cuya inicial sea la misma que la de su nombre Investigar un lenguaje de programación cuya inicial sea la misma que la de su nombre, en este caso **Daniela**, usaré **Dart**.

Un acordeón, como ya sabemos, es un trozo de papel en el que se sintetizan aspectos claves que pueden ser olvidados o son difíciles de aprender antes de una evaluación. Son básicamente notas de aspectos clave del curso.

## Lenguaje C

<b>Control de versiones:</b> Registro de cambios Local, Centralizado, Distribuido	<b>Repositorio:</b> Directorio de trabajo Local, Remoto, Nube	<b>Sistema operativo:</b> Conjunto de programas y datos que administra recursos de hardware y software de un equipo de computo y/o comunicación
<b>Linux:</b> SO tipo Unix de libre distribución Núcleo (kernel), programas y bibliotecas.	<b>Comandos básicos (Linux):</b> <b>Terminal</b> <ul style="list-style-type: none"><li>- <b>ls</b> lista elementos</li><li>- <b>touch</b> crea archivos de texto <code>touch nombre_archivo[.ext]</code></li><li>- <b>mkdir</b> crea una carpeta <code>mkdir nombre_carpeta</code></li><li>- <b>cd</b> cambio de directorio</li><li>- <b>pwd</b> conocer ruta</li><li>- <b>find</b> <code>find . -name cadena_buscar</code></li><li>- <b>clear</b></li><li>- <b>cp</b> copy <code>cp archivo_origen archivo_destino</code></li><li>- <b>mv</b> <code>mv ubicación_origen/archivo ubicación_destino</code></li><li>- <b>rm</b> eliminar <code>rm nombre_archivo</code> Ó <code>rm nombre_carpeta</code></li></ul>	
<b>Algoritmo:</b> Serie de pasos ordenados para dar solución a un problema.	<b>Diagramas de flujo:</b> Representación grafica de un algoritmo	<b>Ciclo de vida del software:</b> <ul style="list-style-type: none"><li>- <b>Definición de necesidades.</b></li><li>- <b>Análisis</b> [datos de entrada y salida]</li><li>- <b>Diseño</b> de estructura de datos, arquitectura del programa y el procedimiento algorítmico.</li><li>- <b>Codificación.</b></li><li>- <b>Pruebas, Mantenimiento y evolución.</b></li></ul>
<b>Pseudocódigo:</b> Representación escrita del Algoritmo.		
<b>Estructuras de control de flujo</b> {secuencial (orden) {condicional (if-else) {repetitivas o iterativas(cíclicas, do-do while)		

Funciones: Subprocesos

Identificador (nombre) y un valor de retorno.

Editores (de texto), compilación y ejecución. Lenguaje C, de alto nivel.

Símbolo del sistema:

gcc calculadora.c -o calculadofra.exe  
Calculadora.exe

- Comentarios {// ó /\* →\*/}
- Declaración de variables y tipos de datos (Signed [-], unsigned[+];{char, short, int, long, enum}): [modificadores] tipoDeDato identificador [= valor];
  - Caracteres (char): codificación definida or la maquina (%c, %d, %i, %o, %x)
  - Cadena de caracteres (%s)
  - Enteros (): números sin punto decimal (%d, %i, %ld, %o, %x)
  - Flotantes(float): números reales de precisión normal (%f, %lf, %e, %g)
  - Dobles (double): números reales de doble precisión
- Almacenar e imprimir variables
  - printf("El valor de la variable real es: %lf", varReal);
  - scanf ("%i", &varEntera);

\a (alarma), \b (retroceso), \f (avance de hoja), \n (salto de línea), \r (regreso de carro), \t tabulador horizontal, \v (tabulador vertical), '\0' carácter nulo

- Modificadores de alcance
  - Const y static
- Operadores {+,-,\*,/,%, >>,<<,&,|,~,!,&&,| |}
- Expresiones lógicas {==,!=,>,<,>=,<=}
- Estructuras de selección (if, if-else, switch-case), repetición (while, do, do-while, for), (define, break, continue)

Función main

```
valorRetorno nombre (parámetros){  
    // bloque de código de la función  
}
```

### Depuración de programas:

**Error.** Acción humana.

**Defecto (Fault).** Error en el software.  
Por una Falla (failure).

**Falla (failure).** Es una desviación del servicio o resultado esperado.

### Apuntadores (while, for):

variable que contiene la dirección de una variable.

Trabajan directamente con la memoria, a través de ellos se accede con rapidez a un dato.

TipoDeDato \*apuntador, variable;

apuntador = &variable;

Inicia con el carácter \*.

### Arreglos:

Conjunto de datos contiguos del mismo tipo.

A cada elemento (dato) del arreglo se le asocia una posición particular, el cual se requiere indicar para acceder a un elemento en específico.

### Depuración de C con GCC y GDB:

gcc -g -o calculadora calculadora.c  
gdb ./calculadora

- *list* o *l*: listar Ej: *list 4,6*
- *b*: punto de ruptura Ej: *b 5*
- *d* o *delete*: Elimina un break point. Ej: *d 5*
- *clear*: Elimina todos break point. Ej: *clear*
- *info line*: Ej: *info line 8*
- *run* o *r*: Ejecuta el programa
- *c*: Continúa con la ejecución del programa después de un punto de ruptura.
- *s*: continua con la siguiente instrucción después de un break point.
- *n*: salta a la siguiente línea de código después de un break point.
- *p* o *print*: Muestra el valor de una variable, Ej: *p suma\_acumulada*
- *ignore*: Ignora un break point. Ej: *ignore 5*
- *q* o *quit*: Termina la ejecución de GDB.

### Arreglos unidimensionales(while, for)

tipoDeDato nombre[tamaño]

### Arreglos multidimensionales(while, for)

tipoDeDato nombre[tamaño] ...[tamaño]

### Lectura y escritura de datos: FILE \*F

\*FILE fopen(char \*nombre\_archivo, char \*modo);

- r: Abre un archivo de texto para lectura.
- w: Crea un archivo de texto para escritura.
- a: Abre un archivo de texto para añadir.
- r+: Abre un archivo de texto para lectura / escritura.
- w+: Crea un archivo de texto para lectura / escritura.
- a+: Añade o crea un archivo de texto para lectura / escritura.
- rb: Abre un archivo en modo lectura y binario.
- wb: Crea un archivo en modo escritura y binario.

int fclose(FILE \*apArch);

char \*fgets(char \*buffer, int tamaño, FILE \*apArch);

char \*fputs(char \*buffer, FILE \*apArch);

int fprintf(FILE \*apArch, char \*formato, ...);

int fscanf(FILE \*apArch, char \*formato, ...);

int fread(void \*ap, size\_t tam, size\_t nelem, FILE \*archivo)

int fwrite(void \*ap, size\_t tam, size\_t nelem, FILE \*archivo)

## Lenguaje Dart

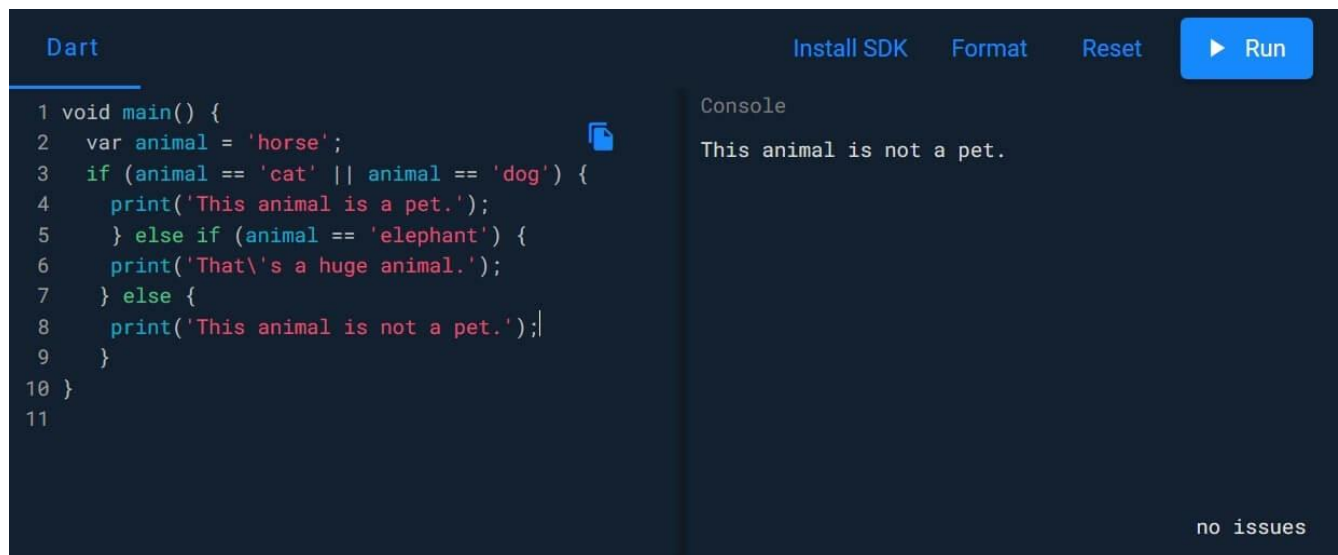
### Datos del lenguaje Dart:

- Es relativamente nuevo si se compara con los diferentes lenguajes de programación de internet.
- Dedicado a la creación de aplicaciones móviles para smartphones o tabletas.
- Desarrollado por Google en 2010.
- Es una alternativa interesante a JavaScript en los navegadores web actuales.
- Como los navegadores no pueden trabajar con este lenguaje de forma natural, y JavaScript puede ejecutarse en todos los navegadores actuales, existe el compilador Dart2js, es decir, "Dart para JavaScript".
- El lenguaje Dart se asemeja a los ya establecidos lenguajes de programación orientados a objetos, entre los que se encuentran Swift, C# o Java, que se subordinan a determinados paradigmas de programación.
- Las reglas para combinar caracteres definidos, es decir la sintaxis, son similares al lenguaje C.
- El lenguaje Dart dispone de variables, operadores, enunciados condicionales, bucles, funciones, clases, objetos y listas.
- Es posible probarlo en internet a través de la página de DartPad
- Dart es un lenguaje de código abierto, de acceso gratuito para cualquier persona.

Cualquier programa escrito en Dart comienza con la función "main".

```
void main() {  
}
```

Ejemplo de definición de una variable y ejecución de un enunciado condicional:



The screenshot shows the DartPad web interface. On the left, the Dart code editor contains the following code:

```
1 void main() {  
2   var animal = 'horse';  
3   if (animal == 'cat' || animal == 'dog') {  
4     print('This animal is a pet.');5   } else if (animal == 'elephant') {  
6     print('That\'s a huge animal.');7   } else {  
8     print('This animal is not a pet.');9   }  
10 }  
11
```

On the right, the console output shows the result of the program execution:

```
Console  
This animal is not a pet.
```

At the top right of the interface, there are buttons for "Install SDK", "Format", "Reset", and a blue "Run" button. At the bottom right, it says "no issues".

Un ejemplo sencillo para demostrar la programación con el lenguaje Dart

En la variable "animal" (`var animal`), sustituye "horse" por "cat", "dog" o "elephant" y observa los cambios en el resultado en la consola a la derecha.

El código siguiente utiliza muchas de las características más básicas de Dart:

```
// Define a function.
void printInteger(int aNumber) {
    print('The number is $aNumber.');// Print to console.
}

// This is where the app starts executing.
void main() {
    var number = 42; // Declare and initialize a variable.
    printInteger(number); // Call a function.
}
```

Esto es lo que este programa utiliza que se aplica a todas (o casi todas) aplicaciones dart:

**// This is a comment.** Un comentario de una sola línea.

**Void** Un tipo especial que indica un valor que nunca se usa.

**Int** Otro tipo, que indica un entero.

**42** Un literal numérico especie de constante en tiempo de compilación.

**print()** Una forma práctica de mostrar la salida.

**'...'** (o **"..."**) Un literal de cadena.

**\$variableName** (o **\${expression}**) Interpolación de cadenas

**main()** La función especial, *necesaria*, de nivel superior donde se inicia la ejecución de la aplicación.

**Var** Una forma de declarar una variable sin especificar su tipo.

### **Variables: Ejemplo de creación de una variable e inicialización, sin tipo: `var name = 'Bob';`**

Si un objeto no está restringido a un solo tipo, especifique el o tipo, siguiendo [las directrices de diseño](#). `nameStringObjectdynamic : dynamic`  
`name = 'Bob';`

Otra opción es declarar explícitamente el tipo que se infiere: `String name = 'Bob';`

## Valor predeterminado

Las variables no inicializadas tienen un valor inicial. Incluso las variables con tipos numéricos son inicialmente null, porque los números, como todo lo demás en Dart, son objetos.`null`

```
int lineCount;  
assert(lineCount == null);
```

## Final y const

Si nunca tiene la intención de cambiar una variable, utilice `final` o `const`, ya sea en lugar de `var` o además de un tipo. Una variable final solo se puede establecer una vez; una variable const es una constante en tiempo de compilación. (Las variables Const son implícitamente finales.) Una variable final de nivel superior o clase se inicializa la primera vez que se usa.`finalconstvar`

```
final name = 'Bob'; // Without a type annotation  
final String nickname = 'Bobby';  
No puede cambiar el valor de una variable:final
```

```
const bar = 1000000; // Unit of pressure (dynes/cm2)  
const double atm = 1.01325 * bar; // Standard atmosphere
```

La palabra clave no es solo para declarar variables constantes. También puede usarlo para crear valores constantes, así como para declarar constructores que crean valores constantes. Cualquier variable puede tener un valor constante.`const`

```
var foo = const [];  
final bar = const [];  
const baz = []; // Equivalent to `const []`
```

Puede omitir desde la expresión inicializadora de una declaración, como la anterior.  
`foo = [1, 2, 3];` // Was const [] No puede cambiar el valor de una variable:`const`

```
baz = [42]; // Error: Constant variables can't be assigned a value.  
Puede definir constantes que utilicen comprobaciones de tipos y fundiciones ( y ), colección si, y operadores de pliegue ( y ): isas.....?
```

```
const Object i = 3; // Where i is a const Object with an int value...  
const list = [i as int]; // Use a typecast.  
const map = {if (i is int) i: "int"}; // Use is and collection if.  
const set = {if (list is List<int>) ...list}; // ...and a spread.
```

**Tipos incorporados:** El idioma Dart tiene soporte especial para los siguientes tipos:

- Números
- Cadenas
- Booleanos
- listas (también conocidas como *matrices*)
- Establece
- Mapas
- runas (para expresar caracteres Unicode en una cadena)
- Símbolos

Dado que cada variable de Dart hace referencia a un objeto (una instancia de una *clase*), normalmente puede usar *constructores* para inicializar variables. Algunos de los tipos integrados tienen sus propios constructores. Por ejemplo, puede usar el constructor para crear un mapa.`Map()`

### Números

Estos tipos de caracteres tienen valores similares en bits al lenguaje C en cuanto a `int` and `double`

Los números literales son constantes en tiempo de compilación. Muchas expresiones aritméticas también son constantes en tiempo de compilación, siempre y cuando sus operandos sean constantes en tiempo de compilación que se evalúan como números.

```
const msPerSecond = 1000;
const secondsUntilRetry = 5;
const msUntilRetry = secondsUntilRetry * msPerSecond;
```

### Cadenas

Una cadena Dart es una secuencia de unidades de código UTF-16. Puede utilizar comillas simples o dobles para crear una cadena:

```
var s1 = 'Single quotes work well for string literals.';
var s2 = "Double quotes work just as well.";
var s3 = 'It\'s easy to escape the string delimiter.';
var s4 = "It's even easier to use the other delimiter.";
```

### Booleanos

Para representar valores booleanos, Dart tiene un tipo denominado `bool`. Solo dos objetos tienen `bool` de tipo: los literales booleanos `true` y `false`, que son ambas constantes en tiempo de compilación.`bool true false`

este: `if (nonbooleanValue) assert (nonbooleanValue)`

```
// Check for an empty string.
var fullName = "";
```



```

assert(fullName.isEmpty);

// Check for zero.
var hitPoints = 0;
assert(hitPoints <= 0);

// Check for null.
var unicorn;
assert(unicorn == null);

// Check for NaN.
var iMeantToDoThis = 0 / 0;
assert(iMeantToDoThis.isNaN);

```

## Listas

Tal vez la colección más común en casi todos los lenguajes de programación es la *matriz* o el grupo ordenado de objetos. En Dart, las matrices son objetos [List](#), por lo que la mayoría de las personas simplemente las llaman *listas*.

Los literales de lista de dardos se parecen a los literales de matriz de JavaScript. Aquí hay una lista simple de Dardos:

```

var list = [1, 2, 3];
var list = [
  'Car',
  'Boat',
  'Plane',
];

```

Las listas utilizan la indexación basada en cero, donde 0 es el índice del primer valor y es el índice del último valor. Puede obtener la longitud de una lista y hacer referencia a los valores de lista tal como lo haría en JavaScript: `list.length - 1`

Dart también ofrece **colección if** y **collection for**, que puede utilizar para crear colecciones mediante condicionales () y repetición ().`iffor`

## If

```

var nav = [
  'Home',
  'Furniture',
  'Plants',
  if (promoActive) 'Outlet'
];

```

## For

```

var listOfInts = [1, 2, 3];
var listOfStrings = [
  '#0',

```

```
for (var i in listOfInts) '#$i'
];
assert(listOfStrings[1] == '#1');
```

## Símbolos

Un [objeto Symbol](#) representa un operador o identificador declarado en un programa Dart. Para obtener el símbolo de un identificador, utilice un literal de símbolo, que solo va seguido del identificador: `#`: `#radix` `#bar`

Los literales de símbolo son constantes en tiempo de compilación.

## Funciones

```
bool isNoble(int atomicNumber) {
  return _nobleGases[atomicNumber] != null;
}
```

## Parámetros

Una función puede tener cualquier número de *parámetros posicionales necesarios*. Estos pueden ir seguidos de *parámetros con nombre* `paramName: value` o de *parámetros posicionales opcionales* (pero no ambos).

```
String say(String from, String msg, [String device]) {
  var result = '$from says $msg';
  if (device != null) {
    result = '$result with a $device';
  }
  return result;
}
```

Este es un ejemplo de llamar a esta función sin el parámetro opcional:

```
assert(say('Bob', 'Howdy') == 'Bob says Howdy');
```

Y aquí hay un ejemplo de llamar a esta función con el tercer parámetro:

```
assert(say('Bob', 'Howdy', 'smoke signal') ==
  'Bob says Howdy with a smoke signal');
```

## Valores de parámetro predeterminados

La función se puede utilizar para definir valores predeterminados para parámetros con nombre y posicionales. Los valores predeterminados deben ser constantes en tiempo de compilación. Si no se proporciona ningún valor predeterminado, el valor predeterminado es `.=null`

La función `main()`

Cada aplicación debe tener una función de nivel superior, que sirve como punto de entrada a la aplicación. La función devuelve y tiene un parámetro opcional para los argumentos. `main()` `main()` `voidList<String>`

Este es un ejemplo de la función de una aplicación web: `main()`

```
void main() {  
  querySelector('#sample_text_id')  
    ..text = 'Click me!'  
    ..onClick.listen(reverseText);  
}
```

## Alcance léxico

Dart es un lenguaje de ámbito léxico, lo que significa que el ámbito de las variables se determina estáticamente, simplemente por el diseño del código. Puede "seguir las llaves hacia afuera" para ver si una variable está en el ámbito.

A continuación se muestra un ejemplo de funciones anidadas con variables en cada nivel de ámbito:

```
bool topLevel = true;  
  
void main() {  
  var insideMain = true;  
  
  void myFunction() {  
    var insideFunction = true;  
  
    void nestedFunction() {  
      var insideNestedFunction = true;  
  
      assert(topLevel);  
      assert(insideMain);  
      assert(insideFunction);  
      assert(insideNestedFunction);  
    }  
  }  
}
```

Observe cómo puede utilizar variables desde todos los niveles, hasta el nivel superior. `nestedFunction()`

## Cierres léxicos

Un *cierre* es un objeto de función que tiene acceso a variables en su ámbito léxico, incluso cuando la función se utiliza fuera de su ámbito original.

```
/// Returns a function that adds [addBy] to the
/// function's argument.
Function makeAdder(int addBy) {
  return (int i) => addBy + i;
}
```

```
void main() {
  // Create a function that adds 2.
  var add2 = makeAdder(2);

  // Create a function that adds 4.
  var add4 = makeAdder(4);

  assert(add2(3) == 5);
  assert(add4(3) == 7);
}
```

Todas las funciones devuelven un valor. Si no se especifica ningún valor devuelto, la instrucción se anexa implícitamente al cuerpo de la función. `return null;`

```
foo() {}
```

```
assert(foo() == null);
```

## Operadores

Dart es compatible con los operadores que se muestran en la tabla siguiente. Puede implementar muchos de estos operadores como miembros de clase.

Descripción	Operador
postfijo unary	<code>expr++</code> <code>expr--</code> <code>()</code> <code>[]</code> <code>.</code> <code>?.</code>
prefijo unario	<code>-expr</code> <code>!expr</code> <code>~expr</code> <code>++expr</code> <code>--expr</code> <code>await expr</code>
Multiplicativo	<code>*</code> <code>/</code> <code>%</code> <code>~/</code>
Aditivo	<code>+</code> <code>-</code>
Cambio	<code>&lt;&lt;</code> <code>&gt;&gt;</code> <code>&gt;&gt;&gt;</code>
bit a bit Y	<code>&amp;</code>
XOR bit a bit	<code>^</code>
bit a bit O	<code> </code>

Descripción	Operador
prueba relacional y de tipo	<code>&gt;=</code> <code>&gt;</code> <code>&lt;=</code> <code>&lt;</code> <code>as</code> <code>is</code> <code>is!</code>
Igualdad	<code>==</code> <code>!=</code>
lógico Y	<code>&amp;&amp;</code>
lógico O	<code>  </code>
si es nulo	<code>??</code>
Condicional	<code>expr1 ? expr2 : expr3</code>
Cascada	<code>..</code>
Asignación	<code>=</code> <code>*=</code> <code>/=</code> <code>+=</code> <code>-=</code> <code>&amp;=</code> <code>^=</code> <i>etcetera.</i>

Cuando se utilizan operadores, se crean expresiones. Estos son algunos ejemplos de expresiones del operador:

```

a++
a + b
a = b
a == b
c ? a : b
a is T

```

### Operadores aritméticos

Dart es compatible con los operadores aritméticos habituales, como se muestra en la tabla siguiente.

Operador	Significado
<code>+</code>	Añadir
<code>-</code>	Restar
<code>-expr</code>	Unary menos, también conocido como negación (invertir el signo de la expresión)
<code>*</code>	Multiplicar
<code>/</code>	Dividir
<code>~/</code>	Dividir, devolver un resultado entero
<code>%</code>	Obtener el resto de una división entera (modulo)

Ejemplo:

```
assert(2 + 3 == 5);
assert(2 - 3 == -1);
assert(2 * 3 == 6);
assert(5 / 2 == 2.5); // Result is a double
assert(5 ~/ 2 == 2); // Result is an int
assert(5 % 2 == 1); // Remainder

assert('5/2 = ${5 ~/ 2} r ${5 % 2}' == '5/2 = 2 r 1');
```

Dart también admite operadores de incremento y decremento de prefijo y postfijo.

Operador	Significado
<code>++var</code>	<code>var = var + 1</code> (el valor de expresión es <code>var + 1</code> )
<code>var++</code>	<code>var = var + 1</code> (el valor de expresión es <code>var</code> )
<code>--var</code>	<code>var = var - 1</code> (expression value is <code>var - 1</code> )
<code>var--</code>	<code>var = var - 1</code> (expression value is <code>var</code> )

Example:

Operador	Significado
<code>==</code>	Igual; ver debate a continuación
<code>!=</code>	No es igual
<code>&gt;</code>	Mayor que
<code>&lt;</code>	Menos de
<code>&gt;=</code>	Mayor o igual que
<code>&lt;=</code>	Menor o igual que

## Declaraciones de flujo de control

Puede controlar el flujo de su código Dart utilizando cualquiera de las siguientes opciones:

- `if` Y `else`
- `for` Bucles
- `while` y `- buclesdowhile`
- `break` Y `continue`
- `switch` Y `case`
- `assert`

Finalmente

Para asegurarse de que se ejecuta algún código, se produce o no una excepción, utilice una cláusula. Si ninguna cláusula coincide con la excepción, la excepción se propaga después de que se ejecute la cláusula: `finallycatchfinally`

```
try {  
  breedMoreLlamas();  
} finally {  
  // Always clean up, even if an exception is thrown.  
  cleanLlamaStalls();  
}
```

La cláusula se ejecuta después de cualquier cláusula coincidente: `finallycatch`

```
try {  
  breedMoreLlamas();  
} catch (e) {  
  print('Error: $e'); // Handle the exception first.  
} finally {  
  cleanLlamaStalls(); // Then clean up.  
}
```

Obtenga más información leyendo la sección [Excepciones](#) del recorrido por la biblioteca.

## Clases

Dart es un lenguaje orientado a objetos con clases y herencia basada en mixin. Cada objeto es una instancia de una clase y todas las clases descienden de [Object](#). *Herencia basada en Mixin* significa que aunque cada clase (excepto Object) tiene exactamente una superclase, un cuerpo de clase se puede reutilizar en varias jerarquías de clases. [Los métodos de extensión](#) son una forma de agregar funcionalidad a una clase sin cambiar la clase ni crear una subclase.

### Uso de miembros de clase

Los objetos tienen *miembros* que constan de funciones y datos (*métodos* y *variables de instancia*, respectivamente). Al llamar a un método, se *invoca* en un objeto: el método tiene acceso a las funciones y datos de ese objeto.

### Uso de constructores

Puede crear un objeto mediante un *constructor*. Los nombres de constructor pueden ser uno o . Por ejemplo, el código siguiente crea objetos mediante el y constructores: `ClassNameClassName.identifierPointPoint()Point.fromJson()`

## Constructores

Declare un constructor creando una función con el mismo nombre que su clase (más, opcionalmente, un identificador adicional como se describe en [Named constructores](#)). La forma más común de constructor, el constructor generativo, crea una nueva instancia de una clase:

Métodos: Los métodos son funciones que proporcionan comportamiento para un objeto.

Operadores: Los operadores son métodos de instancia con nombres especiales.

Captadores y establecedores: Los captadores y establecedores son métodos especiales que proporcionan acceso de lectura y escritura a las propiedades de un objeto.

Comentarios: básicamente mismo código de C.

## Conclusiones

Realizar el acordeón del lenguaje C fue mucho más sencillo que el del lenguaje Dart, yo creo que esto se ve influenciado porque ya he estudiado el lenguaje C y me explicaron sus características mas importantes. Mientras que el lenguaje Dart solo recopile información, es un lenguaje orientado a objetos y aparecen funciones con las cuales no estoy familiarizada. Quizá si estudiara un poco más acerca de este tipo de lenguaje un acordeón sería más fácil de realizar debido a que sería solo un repaso ya que mucha de la información se encontraría en mi cabeza.

La diferencia en mis conocimientos de un lenguaje a otro se ven muy reflejado en la organización del acordeón. De todos modos, en la tercera referencia es un directorio completo del lenguaje y ahí podría sacar mucha información para mi estudio.

## Referencias:

- 1&1. (15.10.20). Dart de Google: Una introducción al lenguaje Dart. 27.02.21, de IONOS  
Sitio web: <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/lenguaje-de-programacion-dart-de-google/>
- Victor Diví . (2010). ¿Qué es el lenguaje de programación Dart?. 27.02.21, de InLab FIB  
Sitio web: <https://inlab.fib.upc.edu/es/blog/que-es-el-lenguaje-de-programacion-dart>
- Flutter. (2015). Dart Tour . 27.02.21, de Dart Sitio web:  
<https://dart.dev/guides/language/language-tour>