RELACIÓN DE EJERCICIOS DE DART (III) EJERCICIOS CON FUNCIONES - 2ª PARTE

Para la resolución de estos ejercicios, seguiremos la documentación oficial

1- Vamos a resolver por fin, aunque de forma no satisfactoria estéticamente, el problema de introducir valores por entrada estándar y validarlos antes de que propaguen un error en tiempo de ejecución. Con valores numéricos, será fácil, con los **String**, algo más complejo.

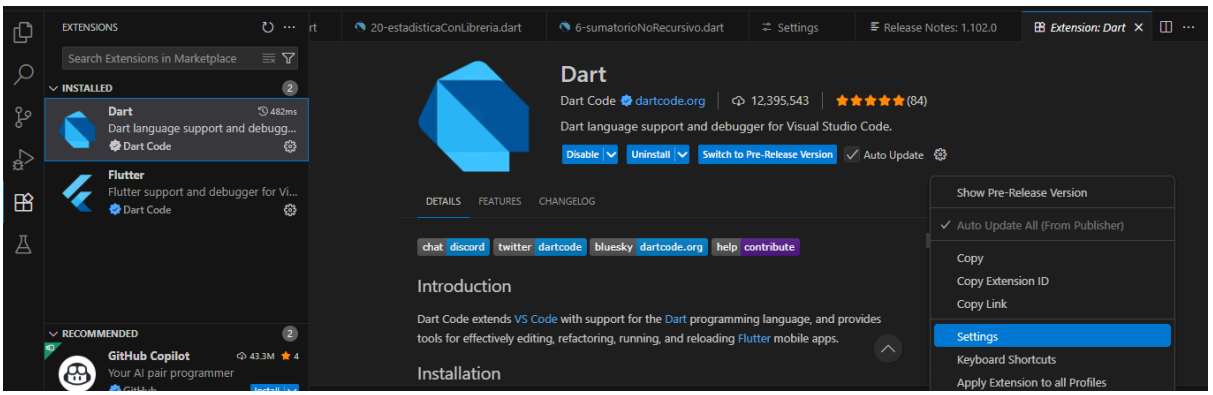
Volveremos a versionar el ejercicio 6 de la relación (II) que calculaba la suma de los n primeros números naturales, pero esta vez, el valor de n será solicitado por consola.

Para ello deberás importar la librería **dart:io** e invocar a las funciones:

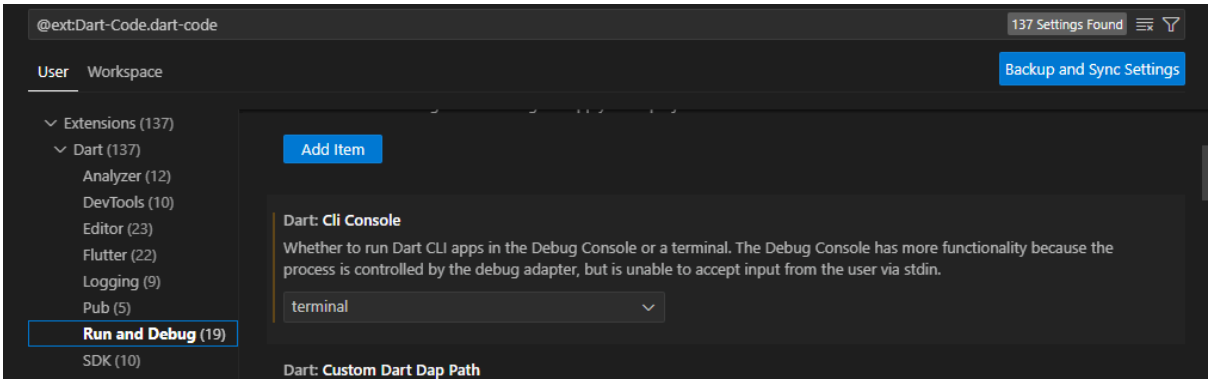
- stdout.write("Introduce un número ") → devuelve void
- stdin.readLineSync() → devuelve un string captado por la entrada estándar
- int.parse() → convierte un String en número entero (se puede utilizar también con double y bool)

También deberás ajustar el entorno de ejecución dentro de VSCode de la siguiente manera: Hasta ahora mismo, la consola que muestra los resultados es la de Depuración (Debug) que no se adapta exactamente al comportamiento de stdin.

Puedes cambiar esto yendo Extensiones -> Dart. pulsa la ruedecita de settings que aparece bajo el título, y después pulsar Settings:



Si te desplazas hacia abajo, en el apartado Run & Debug encontrarás "Dart: Cli Console". Debes ponerla a “terminal”. Finalmente, reinicia VSCode



Todo funciona correctamente si se introduce un número tal y como se espera, pero ¿qué ocurre si no es así? ¿qué pasa si se introduce algo no convertible a entero?¿o un intro? Soluciona este problema con **try-catch**.

Ahora puedes retomar los ejercicios de las relaciones anteriores y experimentar con distintos tipos de datos de entrada. Propongo rehacer el ejercicio de cálculo de factorial (en alguna de sus versiones), y/o el de cálculo de las raíces de una ecuación de 2º grado a partir de los coeficientes a, b y c (de tipo double)

2- Haz un programa Dart “adivina un número entre 1 y 100”. El programa deberá extraer un número al azar, ir pidiendo al usuario exterior que lo adivine, y darle pistas (“te pasaste”, “te quedaste corto”), hasta que lo acierte. En ese momento, mostrará una felicitación, así como el número de intentos que te ha costado acertar.

Probar distintas estrategias. La estrategia más exitosa en promedio es la *búsqueda binaria*

Para resolver el resto de ejercicios de esta relación, estudiaremos a fondo la librería **dart:core**. No será necesario importarla, pues está “built-in”, pero se recomienda dar un repaso a la documentación al respecto.

3- Vamos a declarar en un programa Dart un String al que inicializamos con una frase. Examinaremos el contenido del String para saber si está todo en mayúsculas, en minúsculas o ni lo uno ni lo otro. (**toUpperCase()** y **toLowerCase()**). Investigaremos y probaremos también cómo se crea un string multilínea.

4- Haremos un pequeño programa en Dart que defina e inicialice internamente un **String** y, a continuación, una serie de datos de cada uno de los tipos vistos en la relación 1 (**int, double, bool, List, Map, Enum, Set**) y concatenamos el String con cada uno de ellos utilizando el **operador +** y la función **toString()**. Mostraremos el resultado por pantalla para ver cómo se comporta esta función.

5- Para el caso inverso anterior, utilizaremos la función **parse()**: a partir de una serie de **Strings** que representan en formato cadena datos de tipo **int, double**, etc… convertiremos y asignaremos su contenido a variables de cada uno de estos tipos. ¿Qué ocurre con **List, Map, Enum, Set**…? ¿Cómo deben convertirse en **String**?

Por otra parte, a la hora de convertir un double en String, existen opciones que permiten controlar el formato. Investígalo

6- Haz un pequeño programa que Dart que te pida una frase por la entrada estándar y te la muestre con las palabras en orden contrario. Y también la frase en orden inverso carácter a carácter. También que te diga si la frase es un palíndromo (por ejemplo: 'Dabale arroz a la zorra el abad' debería dar positivo)

7- Haz un pequeño programa en Dart al que le introduzcas una frase y una subcadena. Buscará si está contenida en la frase, y en qué posición la encuentra por primera y última vez

8- Vamos por fin a trabajar la validación de strings. Investiga cómo se definen las **expresiones regulares** en Dart y construye una función que devuelva true si el string que se le pase como parámetro hace match con la expresión regular de un color codificado en RGB:

- ejemplos positivos: #4F6, #982A3B
- ejemplos negativos: red, 987s, 882244

9- Crea otra función que haga lo mismo con URL's

10- Lo mismo con correos electrónicos

11- Ídem con DNIs españoles. Tendrá que comprobar que la letra es correcta . Consulta la web del Ministerio de Interior

12- Crea una librería con todas estas funciones llamada **validaciones.dart,** y prueba a importarla desde cada programa anterior. Investiga la documentación oficial para más posibilidades

13- Investiga las funciones que se pueden utilizar en el manejo de una lista: **isEmpty, add, addAll, length, removeAt, clear, indexOf, sort**… y construye un programa que defina una lista inicial de colores:

```
List<String> colores = ['rojo','verde','azul'];
```

A partir de ahí, muestre el siguiente menú de operaciones:

1. incluir color
2. eliminar color
3. buscar color
4. mostrar lista de colores
5. salir

El usuario entrará en un bucle del que se sale pulsando la opción 5. En cualquier otro caso, efectuará la operación elegida.

Prueba a incluir, eliminar colores, y visualiza la evolución de la lista. ¿Qué ocurriría si incluimos dos veces el mismo color?¿Se puede considerar esto un comportamiento erróneo? De ser así, ¿cómo lo solucionarías?

14- Haremos lo mismo con las funciones aplicables a un Map: **isEmpty, isNotEmpty, containsKey, remove, putIfAbsen**t, y los atributos **entries, values, keys, length**. y el siguiente **Map** inicial:

```
Map<String,String> grupo Whatsapp = { 'Pepe': '111111111',
                                      'María':'222222222',
                                      'Ana':'3333333'}
```

1. incluir persona
2. eliminar persona
3. buscar persona
4. mostrar componentes grupo
5. salir

15- La clase URI provee funciones para codificar, decodificar, manipular e interpretar URIs. Después de consultar la documentación oficial, y también este artículo explicativo, haz un programa que declare un string que contenga una URI, haga la decodificación, lo parsee y extraiga sus diferentes partes: **scheme, host, path, fragment, origin, queryParameters** y las muestre por pantalla.

Construye también un objeto de la clase URI definiendo cada una de sus partes.

Prueba diferentes tipos de URIs como se indica en el artículo indicado.

16- Explora la clase **DateTime** y las funciones disponibles para operar con fechas y horas. Experimenta con ellas a partir de los ejemplos que se incluyen en este artículo

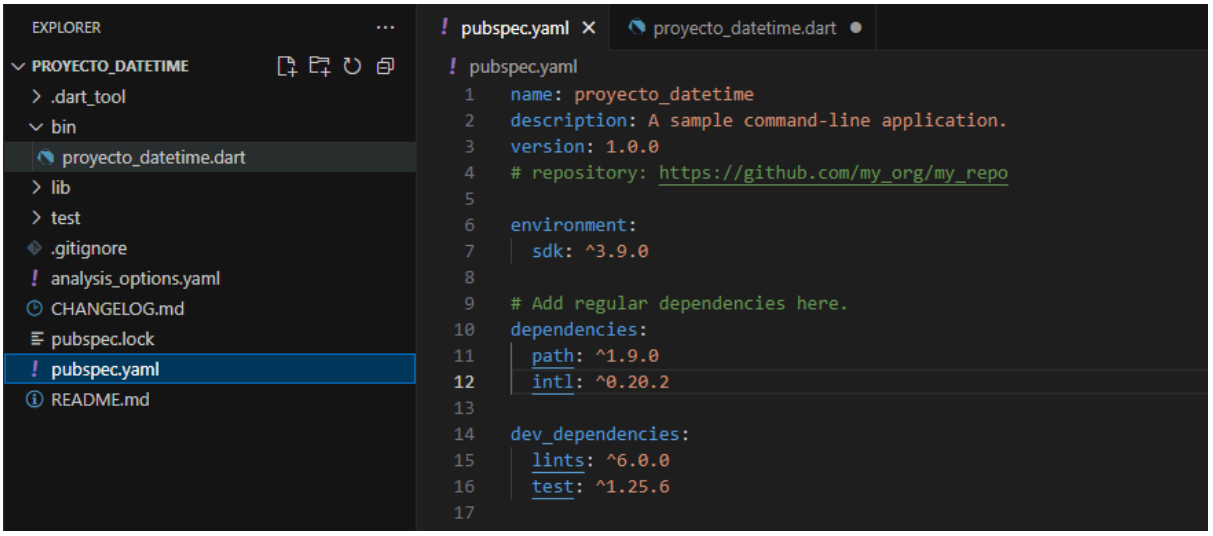
IMPORTANTE: Para poder utilizar algunas de estas tools, vamos a tener que trabajar de una manera distinta (y más profesional) a como lo veníamos haciendo hasta ahora.

Vamos a **crear un proyecto en Dart**

Podemos hacerlo de dos formas:

- manualmente: escribiendo en la consola el comando
  `dart create --template=console-simple mi_app`
- desde VSC:
  view → Command Pallete → Dart: New project → Console application

Después de esto, elijamos la opción que sea, se nos muestra en VSC una estructura de carpetas y archivos de configuración similar a esta:



Nos interesa especialmente fijarnos en el archivo pubspec.yaml. En él aparecen, entre otras cosas, las dependencias de paquetes a los que queramos acceder desde nuestra aplicación. En este caso, el paquete es: **intl** y debemos incluirlo en **dependencies**, bajo **path**

Nuestro programa Dart ejecutable, deberá estar ahora en la subcarpeta bin e incluir la línea de importación:

```
import 'package:intl/intl.dart';
```

A partir de este momento, podremos utilizar todas las tools que se especifican en ese artículo e incluso más.

17- Siguiendo con esta forma de trabajar, vamos a crear un nuevo proyecto llamado **proyecto_testing** y vamos a crear tests unitarios de prueba para las funciones de validación de los ejercicios 9,10 y 11.

- En primer lugar crearemos el proyecto, del mismo modo que lo hicimos en el ejercicio anterior
- Después, pondremos en la subcarpeta **lib** una copia de la librería creada en el ejercicio 12
- A continuación, añadiremos al archivo **pubspec.yaml**, en la sección **dev_dependencies**

  `test: ^1.26.3` (o la última versión)
- Dentro del subdirectorio **test** del proyecto, se habrá creado automáticamente un archivo .dart con el mismo nombre del archivo ejecutable del subdirectorio bin
- Dentro de este archivo de prueba, importa la librería a testear y también el paquete test:

  `import 'package:test/test.dart';`
- Luego, utiliza la función **test** para definir un caso de prueba. Esta función recibe una descripción del caso y una función que contiene la lógica de la prueba.
- Dentro de la función de prueba, utiliza la función **expect** para verificar resultados. expect recibe el valor actual y el valor esperado, y lanza un error si no coinciden

- A continuación, debes ejecutar los tests:
    - manualmente: dart test
    - desde VSC: navega hacia el archivo de pruebas, y pulsa run

Investiga cómo hacerlo, en caso de dudas, en esta documentación y también en este artículo. Debes probar una vez cada función (9-10-11)

18- Vamos a continuar con el ejercicio 8. Probaremos todas las posibilidades: colores sin #, con más dígitos, con menos, con caracteres no contemplados, etc…Para ello crearemos un **group()** con todas las pruebas unitarias relativas a esa función de validación. Ejecutemos el test con todas las posibilidades

19- Finalmente haremos lo propio con todas las funciones. Crearemos una batería de tests para cada una de ellas y los agruparemos utilizando group. Por no multiplicar los proyectos, utilizaremos el mismo

Para más información, se recomienda también leer este otro artículo

20- Investiga qué son las funciones sin nombre y como se declaran y utilizan en Dart. Haz un programa que pruebe cada uno de los siguientes tipos:

1. Función sin nombre asignada a una variable:
2. Función sin nombre dentro de una función de orden superior (como forEach)
3. Función con la sintaxis de flecha (=>) para funciones de una sola línea
4. Función como argumento de otra función
5. Función dentro de un map

BONUS: Para seguir profundizando:

- useful list methods in Dart