Dart

Giới thiệu cơ bản về ngôn ngữ Dart

Language tour	Notes
A basic Dart program comment void main() print ${expression} **Important concepts**	• Comment theo từng dòng: // • Comment nhiều dòng: /* .... */, /// • Hàm(function) cấp cao nhất, bắt buộc để thực thi ứng dụng. • In ra cửa sổ console print(...) • String interpolation: Bao gồm một biến hoặc chuỗi của biểu thức tương đương bên trong một chuỗi ký tự. • biến một biến thành null bằng cách đặt dấu (?) ở cuối mỗi giá trị. Ex: *var int?* có thể là số nguyên hoặc có thể là null. • Đặt dấu (!) ở cuối mỗi giá trị để khẳng định rằng nó không phải là null (và đưa ra một ngoại lệ nếu có). Ex: *int x = nullableButNotNullInt!*

	• Dart hỗ trợ các loại chung, như List<int> (danh sách các số nguyên) hoặc List<Object> (danh sách các đối tượng thuộc bất kỳ loại nào).
Thêm thư viện https://pub.dev/	`flutter pub add {name}`

Giới thiệu về thư viện Dart

Tên thư viện	Notes
dart:core dart:async dart:math dart:convert dart:html dart:io	• Built-in types, collections, and other core functionality.. Thư viện này được tự động nhập vào mọi chương trình Dart. • Hỗ trợ asynchronous programming, với các class như Future và Stream. • Các hằng số và hàm toán học, cùng với bộ tạo số ngẫu nhiên. • Bộ mã hóa và giải mã để chuyển đổi giữa các cách biểu diễn dữ liệu khác nhau, bao gồm JSON và UTF-8. • DOM và các API khác dành cho ứng dụng dựa trên trình duyệt. • I/O cho các chương trình có thể sử dụng Dart VM, bao gồm ứng dụng Flutter, máy chủ và tập lệnh dòng lệnh.

Lập trình bất đồng bộ: futures, async, await

Key terms	Notes
synchronous operation (hoạt động đồng bộ)	• **synchronous operation** ngăn các hoạt động khác thực hiện cho đến khi nó hoàn thành.

synchronous function

(chức năng đồng bộ)

- **synchronous function** chỉ thực hiện các **synchronous operation**.

asynchronous operation

(hoạt động bất đồng bộ)

- Sau khi được bắt đầu, **asynchronous operation** cho phép các hoạt động khác thực thi trước khi hoàn thành.

asynchronous function

(chức năng bất đồng bộ)

- **asynchronous function** thực hiện ít nhất một **asynchronous operation** và cũng có thể thực hiện các **synchronous operation**.

future là gì?

Notes

Future<T>

Future: the Dart Future class.

future: an instance of the Dart Future class.

- Là future tạo ra giá trị loại T
- Ví dụ: future có loại Future<String> tạo ra giá trị chuỗi.

  Nếu future không tạo ra giá trị có thể sử dụng thì loại của future là Future<void>.
- future có thể ở một trong hai trạng thái: uncompleted or completed.
- Khi bạn gọi một fuction trả về(return) future, fuction đó sẽ xếp hàng công việc cần hoàn thành và trả về(return) uncompleted future.
- Khi hoạt động của future kết thúc, future sẽ hoàn thành với một giá trị(value) hoặc lỗi(error).

```
Future<void> fetchUserOrder() {
  // Imagine that this function is fetching user info from an
  return Future.delayed(const Duration(seconds: 2), () => pri
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

```
Future<void> fetchUserOrder() {
// Imagine that this function is fetching user info but encou
  return Future.delayed(const Duration(seconds: 2),
      () => throw Exception('Logout failed: user ID is invali
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

Làm việc với futures:

async and await

async

await

Notes

- sử dụng **async** trước phần thân của **fuction** để đánh dấu nó là **asynchronous**.

```
Future<void> main() async { ··· }
```

- sử dụng **await** để nhận kết quả hoàn chỉnh của biểu thức **asynchronous**. **await** chỉ hoạt động trong **async function**.

```
Future<void> printOrderMessage() async {
  print('Awaiting user order...');
  var order = await fetchUserOrder();
  print('Your order is: $order');
}

Future<String> fetchUserOrder() {
  // Imagine that this function is more complex and slow.
  return Future.delayed(const Duration(seconds: 4), () => 'La
}

void main() async {
  countSeconds(4);
  await printOrderMessage();
}

// You can ignore this function - it's here to visualize dela
void countSeconds(int s) {
  for (var i = 1; i <= s; i++) {
    Future.delayed(Duration(seconds: i), () => print(i));
  }
}
```

Handling errors

```
Future<void> printOrderMessage() async {
  try {
    print('Awaiting user order...');
    var order = await fetchUserOrder();
    print(order);
  } catch (err) {
    print('Caught error: $err');
  }
}

Future<String> fetchUserOrder() {
  // Imagine that this function is more complex.
```

```
  var str = Future.delayed(
      const Duration(seconds: 4),
      () => throw 'Cannot locate user order');
  return str;
}

void main() async {
  await printOrderMessage();
}
```

Làm việc với futures:

async and await

async

await

Notes

- sử dụng **async** trước phần thân của **fuction** để đánh dấu nó là **asynchronous**.

```
Future<void> main() async { ··· }
```

- sử dụng **await** để nhận kết quả hoàn chỉnh của biểu thức **asynchronous**. **await** chỉ hoạt động trong **async function**.