

Nội dung chính

1. Dart variables
 - Các kiểu dữ liệu trong dart
 - Function
 - Enum
 - dynamic vs var
2. Null safety
3. Toán tử trong dart
4. Access modifier
5. For loop, switch case
6. Try catch
7. Practise

Open talk

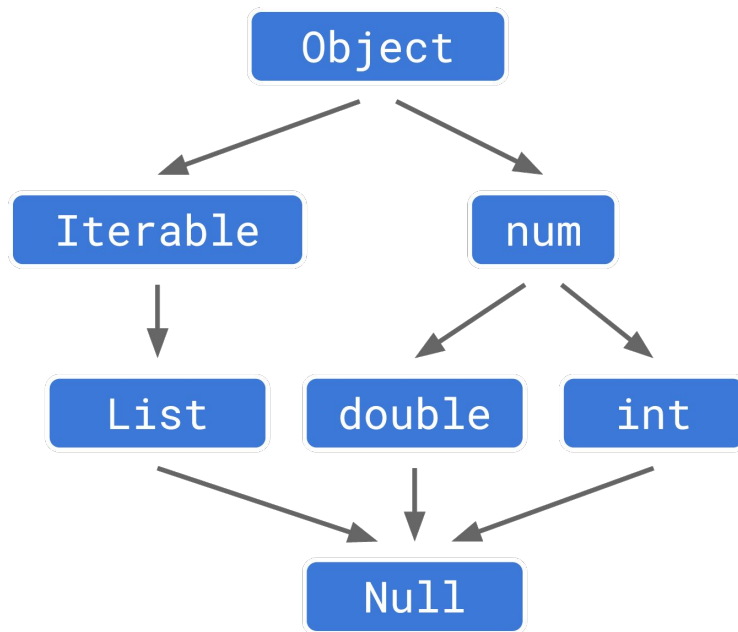
- Các kiểu dữ liệu trong lập trình mà bạn biết?
- Null có phải là 1 kiểu dữ liệu?

Dart Variables

- Các kiểu dữ liệu chính trong dart hầu hết giống với các ngôn ngữ lập trình khác, gồm 2 kiểu chính:

- Non-nullable types
- Nullable types

- Chương trình sẽ bị crash (lỗi) nếu ta truy cập các phương thức của biến với giá trị null.



Các kiểu dữ liệu chính

String

String là gì?

- Kiểu String trong Dart sử dụng để biểu diễn chuỗi ký tự Unicode(UTF-16) (bạn có thể sử dụng các xâu có kí tự tiếng Việt hoặc bất cứ thứ tiếng nào mà sử dụng được với mã Unicode).
- Nhìn chung string trong dart giống các ngôn ngữ khác.

Cách khai báo và sử dụng String

```
void main(List<String> args) {  
    /// Có thể sử dụng nháy đơn ' ' hoặc nháy kép "" để khai báo 1 string  
    String s = 'Hello world';  
    String s1 = "Hello world";  
}
```



1 vài phương thức cần nhớ

- Đa phần các phương thức của string trong dart đều giống so với java/kotlin/python....

- split()
- contains()
- toUpperCase()
- toLowerCase()
- indexOf()
- startsWith()
- replaceAll()
- trim()

split(), contains()

```
void main(List<String> args) {  
    String s = 'Hello world '  
    /// Tách kí tự trong 1 chuỗi  
    List<String> splittedStrs = s.split(" "); splittedStrs -> [Hello,  
world]  
  
    /// kiểm tra 1 string có chứa 1 string khác không  
    bool hasWorldStr = s.contains("world"); // hasWorldStr -> true  
}
```


toUpperCase(), toLowerCase(), indexOf()

```
void main(List<String> args) {  
    String s = 'Hello world';  
  
    /// chuyển đổi tất cả kí tự về kí tự viết hoa, viết thường  
    String sUpperCase = s.toUpperCase(); // sUpperCase -> HELLO WORLD  
    String sLowerCase = s.toLowerCase(); // sLowerCase -> hello world  
  
    /// lấy vị trí đầu tiên của kí tự 'o'  
    int sIndexOf = s.indexOf("o"); // sIndexOf -> 4  
}
```

startsWith(), replaceAll(), trim()

```
void main(List<String> args) {  
    String s = 'Hello world '  
  
    /// kiểm tra xem string bắt đầu với kí tự 'H'  
    bool sStartsWith = s.startsWith("H"); // sStartsWith -> true  
  
    /// thay thế các kí tự trong 1 chuỗi  
    String sReplaced = s.replaceAll("world", "everyone"); // sReplaced -> Hello everyone  
  
    /// loại bỏ khoảng trắng ở đầu/cuối string  
    String sTrimmed = s.trim();  
    print("s -> \"$s\""); // sẽ in ra: s -> "Hello world "  
    print("sTrimmed -> $sTrimmed"); // sẽ in ra: sTrimmed -> Hello world  
}
```

int

int là gì?

- int là kiểu số nguyên. Tùy thuộc vào bộ nhớ máy, nền tảng máy (32 bit hay 64 bit) mà kiểu số nguyên trong Dart có giá trị từ -2^{63} đến $2^{63} - 1$.
- Ngoài kiểu số nguyên int, trong Dart còn có kiểu số nguyên BigInt để làm việc với các số nguyên lớn.
- BigInt khá giống với BigInteger trong java, phù hợp để lưu trữ các số nguyên lớn, như số tiền lạm phát ở Venezuela

Cách khai báo và sử dụng int

```
void main(List<String> args) {  
    int a = 10;  
    int b = -10;  
  
    print("a = $a"); // a = 10  
    print("b = $b"); // b = -10  
}
```

double

double là gì?

- double là kiểu số thực. Các phiên bản cũ của Dart thì kiểu số thực bắt buộc phải có dấu . ví dụ để khai báo biến số thực a có giá trị 10 chúng ta phải viết rõ double a = 10.0 Tuy nhiên các phiên bản hiện nay đã bỏ phần dấu . này đi, bạn có thể khai báo double a = 10
- Trong dart sẽ không có Float như java.

Cách khai báo và sử dụng double

```
void main(List<String> args) {  
    double a = 10;  
    double b = -10.0;  
  
    int c = 10;  
    int d = -10;  
  
    int _resultInt = (a + b + c + d).toInt();  
    double _resultDouble = (a + b + c + d);  
  
    print("_resultInt -> $_resultInt"); // _resultInt -> 0  
    print("_resultDouble -> $_resultDouble"); // _resultInt -> 0.0  
}
```



1 vài phương thức cần nhớ

- Các phương thức cần nhớ để chuyển đổi giữa các kiểu dữ liệu:
 - `toInt()`
 - `toString()`
 - `toDouble()`
 - `parseInt()`
 - `int.tryParse()`
 - `parseDouble()`
 - `double.tryParse()`

Chuyển đổi từ int, double sang string:

```
void main(List<String> args) {  
    int a = 10;  
    double b = -10.0;  
  
    String _aStr = a.toString();  
    String _bStr = b.toString();  
  
    print("int a toString: $_aStr"); // int a toString: 10  
  
    print("double b toString: $_bStr"); // double b toString: -10.0  
}
```


Chuyển từ int qua double:

```
void main(List<String> args) {  
    int a = 10;  
    double _aDouble = a.toDouble();  
  
    print("int a toDouble: $_aDouble"); // int a toDouble: 10.0  
}
```

Chuyển từ double qua int:

```
void main(List<String> args) {  
    double b = -10.0;  
    int _bInt = b.toInt();  
  
    print("double b toInt: $_bInt"); // double b toDouble: -10  
}
```

Chuyển từ String qua double, int:

```
void main(List<String> args) {  
    String c = "20";  
  
    int _cInt = int.parse(c);  
    double _cDouble = double.parse(c);  
  
    print("Parse String c to int: $_cInt"); // Parse String c to int: 20  
    print("Parse String c to double: $_cDouble"); // Parse String c to double:  
    20.0  
}
```

bool

bool là gì?

- Kiểu dữ liệu bool trong Dart có 2 giá trị là true (đúng) và false (sai) được sử dụng để thể hiện kết quả của một mệnh đề logic (các phép toán so sánh, kiểm tra, các hàm...)

Cách khai báo và sử dụng bool

```
void main(List<String> args) {  
    bool a = true;  
    bool b = false;  
  
    if (a) { /// các viết tắt khi so sánh bool == true  
        print("a is $a"); // a is true  
    }  
  
    if (a == true) { /// viết kiểu này rõ nghĩa hơn  
        print("a == true"); // a == true  
    }  
  
    if (!b) { /// cách viết tắt khi so sánh bool == false  
        print("b is $b"); // b is false  
    }  
}
```

Cách khai báo và sử dụng khi so sánh nhiều mệnh đề

```
void main(List<String> args) {  
    bool a = true;  
    bool b = false;  
  
    if (a && !b) { /// so sánh nhiều mệnh đề, viết tắt  
        print("a == true and b == false"); // a == true and b == false  
    }  
  
    if (a || b) { /// so sánh nhiều mệnh đề, viết tắt  
        print("a == true or b == true"); // a == true and b == false  
    }  
}
```

array

array là gì?

- Kiểu dữ liệu array (danh sách - List) trong Dart được sử dụng để biểu diễn cho một tập hợp các đối tượng theo một thứ tự nhất định.
- Kiểu array trong Dart tương đương với kiểu mảng Array trong các ngôn ngữ khác (java, kotlin).
- Dữ liệu trong array được lưu trữ theo vị trí.

Open talk

- Các collection mà bạn biết?
- List
- ArrayList
- Set
- Vector
- LinkedList
- Queue

Cách khai báo và sử dụng array

```
void main(List<String> args) {  
    List strs = ["a", "b", "c"]; /// cách viết tắt  
    List<String> strs_1 = <String>["a", "b", "c"]; /// cách viết tắt rõ  
nghĩa, sử dụng [generic]  
}
```



1 vài phương thức cần nhớ

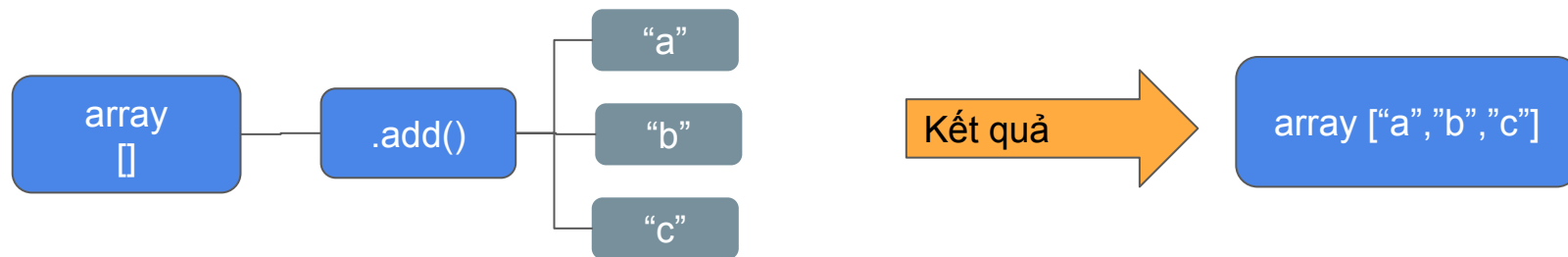
- Các phương thức cần nhớ khi làm việc với array:

- `add()`
- `addAll()`
- `remove()`
- `subList()`

Thêm mới, xoá 1 phần tử vào array

```
void main(List<String> args) {  
    List strs = ["a", "b", "c"]; /// cách viết tắt  
    List<String> strs_1 = <String>["a", "b","c"]; /// cách viết tắt rõ nghĩa, sử dụng  
[generic]  
  
    strs.add("d"); // "d" -> [a, b, c, d]  
  
    strs.remove("d"); // "d" -> [a, b, c]  
  
    /// đoạn code trên, nhưng sử dụng `Cascade notation` giống với [Builder Design Pattern]  
để viết code ngắn gọn hơn  
    strs..add("d")..remove("d"); // [a, b, c]  
}
```

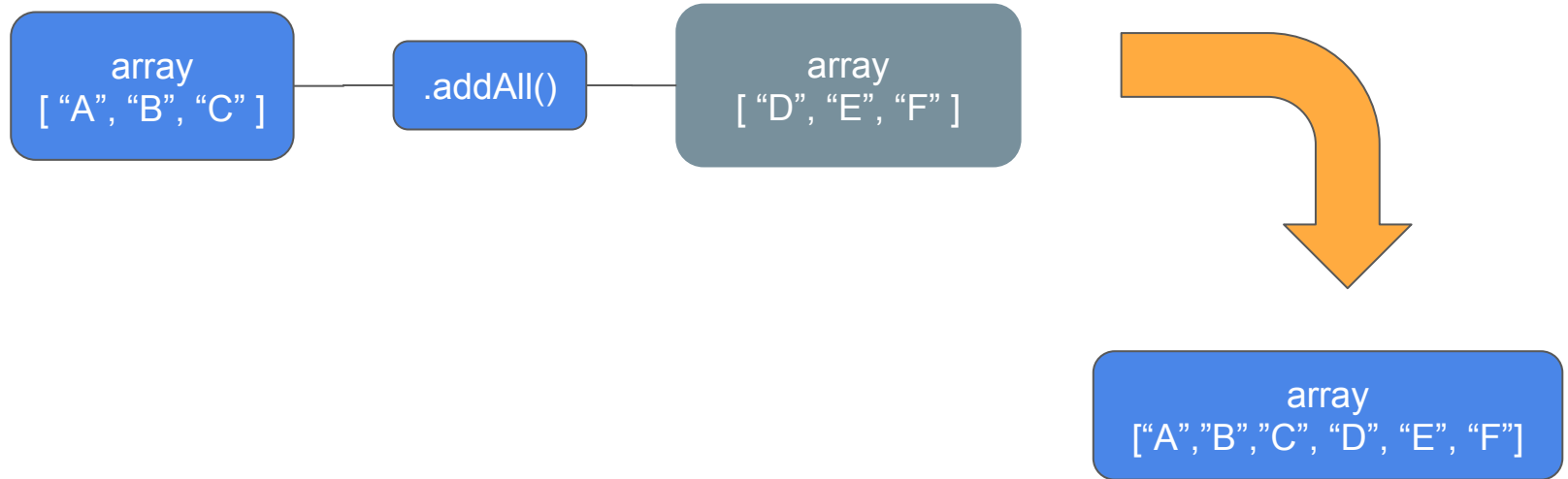
Thêm mới, xoá 1 phần tử vào array



Thêm toàn bộ dữ liệu từ 1 array khác

```
void main(List<String> args) {  
    List<String> strs_1 = <String>["a", "b", "c"]; /// cách viết tắt rõ  
nghĩa, sử dụng [generic]  
  
    List<String> strs_2 = ["d", "e", "f"];  
    strs_1.addAll(strs_2); // strs_1 -> [a, b, c, d, e, f]  
}
```

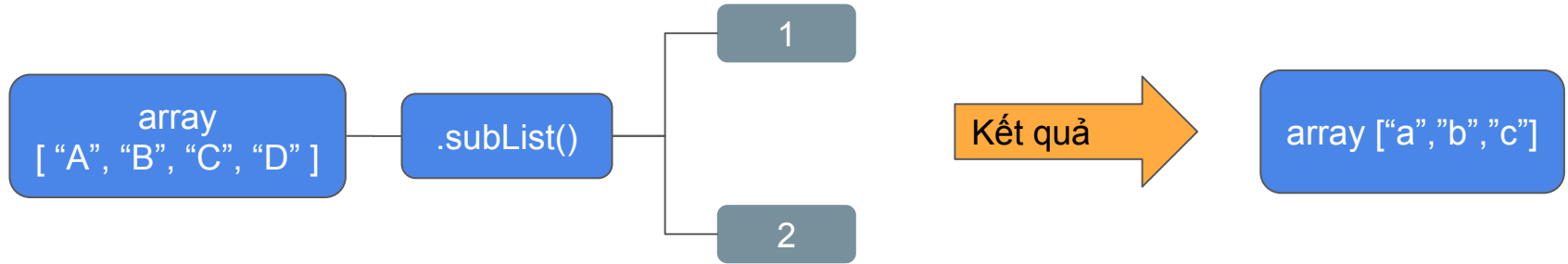
Thêm mới, xoá 1 phần tử vào array



Tách các phần tử thành 1 array từ 1 array

```
void main(List<String> args) {  
    List<String> strs_1 = <String>["a", "b", "c", "d"];  
  
    List<String> strs_1_subList = strs_1.sublist(2, 4);  
    // strs_1_subList -> [c, d]  
}
```

Thêm mới, xoá 1 phần tử vào array



Lấy 1 phần tử từ 1 array

```
void main(List<String> args) {  
    List<String> strs_1 = <String>["a", "b", "c"];  
  
    /// lấy 1 phần tử tại 1 vị trí index trong array  
    String _strByIndex = strs_1[0]; // _strByIndex -> a  
}
```

Tổng kết array

Dart cũng giống các ngôn ngữ lập trình khác như java, kotlin...đều hỗ trợ các phương thức nâng cao cho array như:

- map, reduce, every, skip, take
- where, firstWhere, indexWhere
- fillRange, insert, insertAll

1 số phương thức cần nhớ:

- add()
- addAll()
- remove()
- subList()

map

map là gì?

- Kiểu dữ liệu map trong dart là một đối tượng lưu trữ dữ liệu dưới dạng một cặp khóa-giá trị. Mỗi giá trị được liên kết với khóa của nó và nó được sử dụng để truy cập giá trị tương ứng của nó. Cả khóa và giá trị đều có thể là bất kỳ loại nào.
- Trong dart cũng có hỗ trợ HashMap như các ngôn ngữ lập trình khác: java, kotlin

Open talk

- Bạn đã từng sử dụng map nào?
- Map
- HashMap
- TreeMap
- LinkedHashMap
- WeekHashMap
- SortedMap

Cách khai báo và sử dụng map

```
void main(List<String> args) {  
    /// cách viết ngắn gọn  
    Map map = {"key": "value"}; // map -> {key: value}  
  
    /// cách viết rõ nghĩa  
    Map<String, String> map1 = Map<String, String>();  
    map1["key"] = "value";  
    // map1 -> {key: value}  
}
```

1 vài phương thức cơ bản cần ghi nhớ khi sử dụng map

```
void main(List<String> args) {  
    Map<String, String> map1 = Map<String, String>();  
    map1["key"] = "value"; // map1 -> {key: value}  
  
    /// 1 vài phương thức cơ bản  
    Map<String, String> map2 = Map.from(map1); // map2 -> {key: value}  
  
    String key = map1.keys.first; // key  
  
    String value = map1.values.first; // value  
  
    /// trong map key là duy nhất, không thể có 2 key cùng giá trị  
    Map map12 = map1.addAll(map2); // -> {key: value}  
}
```

map

- Map lưu trữ theo key-value. Vậy có thể thêm 1 cặp key-value trùng với cặp key-value đã tồn tại trong map không?

```
void main(List<String> args) {  
    /// điều gì sẽ xảy ra nếu add 1 map với key mới?  
    Map mapWithNewValue = {"key": "value"}..addAll({"key": "newValue"});  
  
    print("mapWithNewValue -> $mapWithNewValue");  
}
```



Cần chú ý

- Map lưu trữ dữ liệu theo key-value
- Key trong map không thể trùng. Nếu thêm 1 map khác có cùng key, value sẽ được cập nhật lại.

Các từ khoá khi khai báo biến

Dart cũng như các ngôn ngữ lập trình khác đều hỗ trợ các keyword như:

- `const`
- `final`
- `late`
- `static`

Ví dụ minh họa

```
void main(List<String> args) {  
    const a = 10; final b = "b"; late String c;  
  
    a = 20; // vì a là const, nên không thể thay đổi giá trị  
  
    void initB(){ b = 10; // vì b là final, nên không thể thay đổi giá trị }  
  
    initB();  
  
    void initC(){ c = "c"; }  
  
    initC();  
    print(A.d); // sẽ in ra true  
}  
  
class A { static bool d = false; }
```

const vs final

const và **final** trong dart cũng như các ngôn ngữ lập trình khác:

Giống:

- Luôn được dùng để ngăn giá trị của biến thay đổi sau khi khởi tạo.

Khác:

- **const** được khởi tạo ngay khi compile code, **final** được gán giá trị sau khi compile code, và chỉ được khởi tạo khi nó được truy cập tới.
- Không phải cung cấp giá trị khởi tạo cho các biến final, nhưng với const là bắt buộc.
- Một biến instance có thể là **final** nhưng không thể là **const**, vì chỉ có class nào được khởi tạo khi trước compile cả



Cần chú ý khi sử dụng late, vì có thể phá vỡ Null Safety:

```
void main(List<String> args) {  
    DataKeeper dataKeeper = DataKeeper();  
  
    print("a -> ${dataKeeper.a.toString()}");  
    // sẽ in ra:  
    // Unhandled exception:  
    // LateInitializationError: Field 'a' has not been initialized.  
}  
  
class DataKeeper{  
    late int a;  
}
```

Function

Function

Function là gì?

- Hàm là một khối lệnh thực hiện một tác vụ, khối lệnh này được dùng nhiều lần nên gom chúng lại thành một hàm.
- Trong Dart mọi thứ đều là đối tượng nên hàm cũng là một đối tượng (kế thừa Function).

Open talk

- Bạn đã từng viết hàm trong ngôn ngữ lập trình nào chưa?

Cách khai báo và sử dụng function

```
void main(List<String> args) {  
    void test(String test) {  
        print("Đây là hàm test, params test: $test");  
    }  
  
    /// viết và gọi function (hàm)  
    /// cách gọi hàm phổ biến  
    test("cách gọi hàm phổ biến"); // Đây là hàm test, params test: cách gọi hàm  
    /// phổ biến  
}
```


Cách viết khác: sử dụng .call()

```
void main(List<String> args) {  
  void test(String test) {  
    print("Đây là hàm test, params test: $test");  
  }  
  
  /// vì trong dart, mọi thứ đều là đối tượng,  
  /// và hàm kế thừa Function, nên có thể viết như này  
  test.call(  
    "gọi qua method call()"); // Đây là hàm test, params test: gọi qua  
  method call()  
}
```

Là 1 object nên Function cũng có thể truyền vào như 1 params

```
void main(List<String> args) {
    functionTest (() => test("function call function"));
    // sẽ in ra: Đây là hàm test, params test: function call function

    var _ret = functionTestCallBack (parseStringFromInt);
    print("Kết quả khi gọi functionTestCallBack -> $_ret"); // Kết quả khi gọi functionTestCallBack -> Kết quả thực thi function: -1

    var _ret1 = functionTestCallBack ((intStr) => parseStringFromInt ("10"));
    print("Kết quả khi gọi functionTestCallBack -> $_ret1"); // Kết quả khi gọi functionTestCallBack -> Kết quả thực thi function: 10
}

void test(String test){ print("Đây là hàm test, params test: $ test"); }

int parseStringFromInt (String number) => int.tryParse (number) ?? -1;

void functionTest (Function function){ function.call(); }

/// [functionTestCallBack] sẽ thực thi như sau:
/// - Yêu cầu truyền vào 1 function với parameter là 1 string
/// - Thực thi function [callback] và trả về dữ liệu kiểu int
/// - Trả về kiểu dữ liệu sau khi thực thi function [functionTestCallBack] dạng String
String functionTestCallBack (int Function (String) callback){
    int result = callback.call(""); // "" là default value
    return "Kết quả thực thi function: $ result";
}
```

Các loại params trong function

- Optional Params: hay named params:

- Có thể truyền vào params hoặc không truyền.
- Params truyền vào thông qua tên params: Ví dụ: *name*: “*ToanNM*”

- Positional Params:

- Có thể truyền vào params hoặc không truyền.
- Params truyền vào phải đúng vị trí, và không thể sử dụng nhiều kiểu dữ liệu cho các params.

Ví dụ về các loại params trong function:

```
void main(List<String> args) {  
    /// function trong dart hỗ trợ optional params,  
    /// - nếu không truyền params vào function: params sẽ nhận giá trị mặc định  
    /// - nếu truyền params vào function: params sẽ nhận giá trị được truyền vào từ function  
    functionWithOptionalParam (); // a=a, b=b  
  
    functionWithOptionalParam (b: "không phải giá trị b", a: "không phải giá trị a" );  
    // a=không phải giá trị a, b=không phải giá trị b  
  
    functionWithPositionalParam ("a", "b", "c"); // a=a, b=b, b=c, d=, vì không truyền vào d, nên d=""  
}  
  
void functionWithOptionalParam ({String a = "a", String b = "b"})  
{ print("params a=$a, b=$b"); }  
  
void functionWithPositionalParam (String a, [String b = "", c = "", d = ""])  
{ print("params a=$a, b=$b, b=$c, d=$d");}
```

Inline function: function trong function

```
void main(List<String> args) {  
    /// Inline function (function lồng function trong dart)  
    /// [_inlineFunction2] được định nghĩa bên trong function [_inlineFunction1]  
    /// Lúc này chỉ có thể call function [_inlineFunction2] trong function [_inlineFunction2]  
    void _inlineFunction1(){  
        print("_inlineFunction1");  
  
        void _inlineFunction2(){ print("_inlineFunction2"); }  
  
        _inlineFunction2();  
    }  
    _inlineFunction1();  
    // sẽ in ra: _inlineFunction1  
    // sẽ in ra: _inlineFunction2  
}
```

Mở rộng function từ 1 class

- Mở rộng, viết thêm function cho 1 class có sẵn mà không phải extend hay implement lại các class có sẵn.

```
void main(List<String> args) {  
    /// Dart có hỗ trợ extension như kotlin, swift....:  
    /// function [test] được viết thêm vào lớp String  
    String a = "a";  
    a.test(); // Mở rộng function trên lớp String  
}  
  
extension stringExt on String{  
    void test(){ print("Mở rộng function trên lớp String"); }  
}
```



Cần chú ý

- Đây là 1 trong những lỗi đầu đời của tôi, hy vọng bạn sẽ không mắc phải

```
void main() {  
    int value = 0;  
    void test(int value){  
        value = 10; // chỉ thay đổi biến value, trong scope function test  
        print("trong function test value -> $value");  
    }  
    test(value);  
    print("ngoài function test value -> $value"); // kết quả value = 0  
}
```

Open talk

- Vậy có cách nào để truyền **`primitive params`** vào function mà vẫn giữ được tham chiếu của nó không? *`pass params by reference`*

Đơn giản là dùng 1 class để cập nhật lại giá trị của biến

```
void main(List<String> args) {  
    var data = DataKeeper(10);  
    print("data ban đầu: ${data.value}"); // data ban đầu: 10  
    increase(data);  
    print("data sau khi increase: ${data.value}"); // data sau khi increase: 20  
}  
  
class DataKeeper {  
    int value;  
    DataKeeper(this.value);  
}  
  
void increase(DataKeeper data) {  
    data.value = 20;  
}
```

Tổng kết Function

- Function cũng là 1 object trong dart.
- Khi tạo inline function, cần chú ý tới việc đặt tên hàm và nên tuân thủ clean code (1 function không dài quá 20 line, 1 line không nên quá 80 kí tự)
- Function với optional params linh hoạt trong hầu hết các trường hợp.
- Function với positioned params sẽ khó mở rộng trong tương lai. Ví dụ 1 function với positioned params, sẽ không khả thi khi cần truyền thêm params khác variable type.
- Khi truyền primitive params vào 1 function, cần chú ý rằng giá trị của biến chỉ được thay đổi trong function đó.

Enum

Enum là gì?

- Kiểu enum trong Dart còn gọi là kiểu liệt kê (kiểu liệt kê thứ tự enumerated) được sử dụng để liệt kê các giá trị hằng số. Kiểu liệt kê được khai báo bằng cách sử dụng từ khóa enum.
- Kiểu liệt kê enum cũng là một kiểu iterable, tức là cũng có thể duyệt tuần tự qua từng phần tử của nó.
- Enum trong dart cũng giống enum trong các ngôn ngữ lập trình khác như kotlin, java, swift....

Enum

- Bạn đã từng sử dụng enum chưa? Ứng dụng enum vào mục đích gì?

Ví dụ khai báo enum và thêm function: getName()

```
enum DayOfWeeks { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, }

extension DayOfWeeksExt on DayOfWeeks {
    String getName() {
        switch (this) {
            case DayOfWeeks.Monday: return "Thứ 2";
            case DayOfWeeks.Tuesday: return "Thứ 3";
            case DayOfWeeks.Wednesday: return "Thứ 4";
            case DayOfWeeks.Thursday: return "Thứ 5";
            case DayOfWeeks.Friday: return "Thứ 6";
            case DayOfWeeks.Saturday: return "Thứ 7";
            case DayOfWeeks.Sunday: return "Chủ nhật";
            default: return "Không xác định";
        }
    }
}

void main(List<String> args) {
    List<DayOfWeeks> dayOfWeeks = DayOfWeeks.values;
    dayOfWeeks.forEach((element) { print("day is: ${element.getName()}"); // các ngày từ thứ 2 -> chủ nhật });
}
```

dynamic vs var

Open talk

- var trong các ngôn ngữ lập trình khác?

dynamic là gì?

- Trong Dart mọi thứ đều là object. Đã là object thì luôn phải là instance của một class nào đó.
- Vì tất cả là đều là object nên dù là số, chữ hay bất kể loại dữ liệu nào thì giá trị mặc định của nó đều là **`null`**. Nhờ vậy, mọi biến số trong Dart đều là **`reference type`**.
- Cũng chính vì thế mà Dart có một loại biến dynamic chấp nhận mọi kiểu dữ liệu.

var là gì?

- var cũng thuộc kiểu dữ liệu dynamic, nhưng khác ở chỗ dynamic có thể thay đổi kiểu giá trị, còn var thì không thể thay đổi kiểu giá trị
- Cùng xem ví dụ để hiểu rõ hơn cách hoạt động, ưu nhược điểm của dynamic và var nhé.

Dùng var để khai báo các kiểu dữ liệu

```
void main(List<String> args) {  
    /// Khai báo biến var  
    /// Ưu điểm: nhanh, và không cần quan tâm tới [runtimeType]  
    var intVar = 10; // intVar -> 10, loại dữ liệu (variableType) -> int  
  
    var doubleVar = 10.4; // doubleVar -> 10.4, variableType -> double  
  
    var stringVar = "A"; // stringVar -> A, variableType -> String  
  
    var boolVar = 10; // boolVar -> 10, variableType -> int  
  
    var arrayVar = [1, 2, 3]; // arrayVar -> [1, 2, 3], variableType -> List<int>  
  
    var mapVar = {"key": "value"};  
    // mapVar -> {key: value}, variableType -> _InternalLinkedHashMap<String, String>  
}
```

Dùng dynamic để khai báo các kiểu dữ liệu:

```
void main(List<String> args) {  
    /// Khai báo biến dynamic, giống var, nhưng có thể thay đổi variable type  
    dynamic dynamicA = 10; // dynamicA -> 10, lại dữ liệu (variableType) -> int  
  
    dynamicA = 10.4; // dynamicA -> 10.4, variableType -> double  
  
    dynamicA = "A"; // dynamicA -> A, variableType -> String  
  
    dynamicA = true; // dynamicA -> true, variableType -> bool  
  
    dynamicA = [1, 2, 3]; // dynamicA -> [1, 2, 3], variableType -> List<int>  
  
    dynamicA = {"key": "value"};  
    // dynamicA -> {key: value}, variableType -> _InternalLinkedHashMap<String, String>  
  
    /// error khi chạy, vì dynamicA hiện tại là Map, không có phương thức toInt()  
    var a = dynamicA.toInt();  
    // Class '_InternalLinkedHashMap<String, String>' has no instance method 'toInt'  
}
```

Vậy tác dụng của dynamic, var là gì?

```
void main(List<String> args) {  
    Test test = Test().instance(); /// phải chỉ rõ rằng hàm `instance` trả về  
    đối tượng `Test`  
    var a = Test().instance(); /// không cần quan tâm tới `variable type` của  
    hàm `instance`  
  
    /// sử dụng var/dynamic thích hợp cho việc trung chuyển các `variable`  
    /// tức chuyển từ hàm này qua hàm khác, class này qua class khác  
}  
  
class Test{ Test instance() { return this; } }
```

Vậy tác dụng của dynamic, var là gì?

```
void main(List<String> args) {  
    void test(dynamic params){  
        print(params.toString());  
    }  
  
    test(10);  
    test("abc");  
    test(true);  
}
```

Nhưng nên tránh code như sau

```
void main(List<String> args) {  
    void test(dynamic params){  
        print(params.toDouble());  
    }  
  
    test(10);  
    test("abc"); // sẽ báo lỗi Error: NoSuchMethodError: 'toDouble' Dynamic call  
of null. Receiver: "abc"  
}
```

Tổng kết dynamic, var

- dynamic đúng với cái tên của nó, có thể sử dụng linh hoạt trong rất nhiều trường hợp.
- Nhưng đây vừa là ưu điểm, vừa là nhược điểm:
 - Trong ví dụ trên, nếu không check runtimeType thì khó mà biết được variable type của dynamic, điều này dễ dẫn tới việc sử dụng sai phương thức

Tổng kết Dart Variables

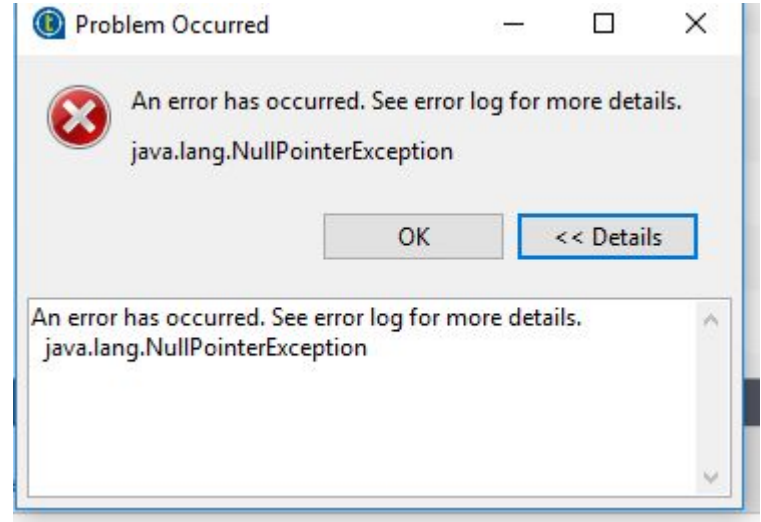
Tổng kết dart variables

- Hầu hết các biến trong dart giống với các ngôn ngữ lập trình khác như: java, kotlin, python.
- Các biến trong dart, mặc định đều là non-nullable, tức không thể có giá trị null.
- Dart Function hỗ trợ cả positional params và optional params.
- Khi sử dụng dynamic cần tránh thay đổi variable types quá nhiều.
- **`late`** có thể phá vỡ Null Safety, cần chú ý khi sử dụng.

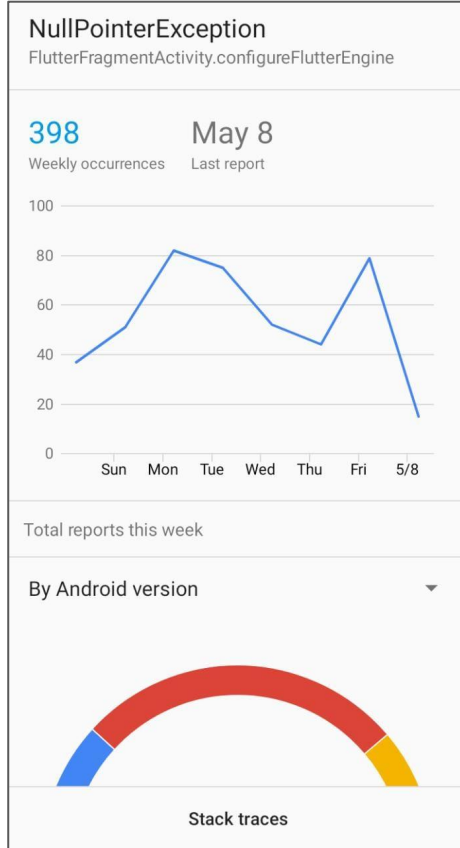
Null Safety

Null Pointer Exception

- Tôi vẫn nhớ những ngày đầu tiên code java, hầu hết lỗi tôi mắc phải là **Null Pointer Exception**, nhìn nhiều tới quen....



Null Pointer Exception report



- 1 thời gian sau, khi đi làm ở 1 vài công ty, khi xem các báo cáo về bugs ở công ty, tôi mới phát hiện ra 80% bugs hiện tại các developer mắc phải đều do Null Pointer Exception

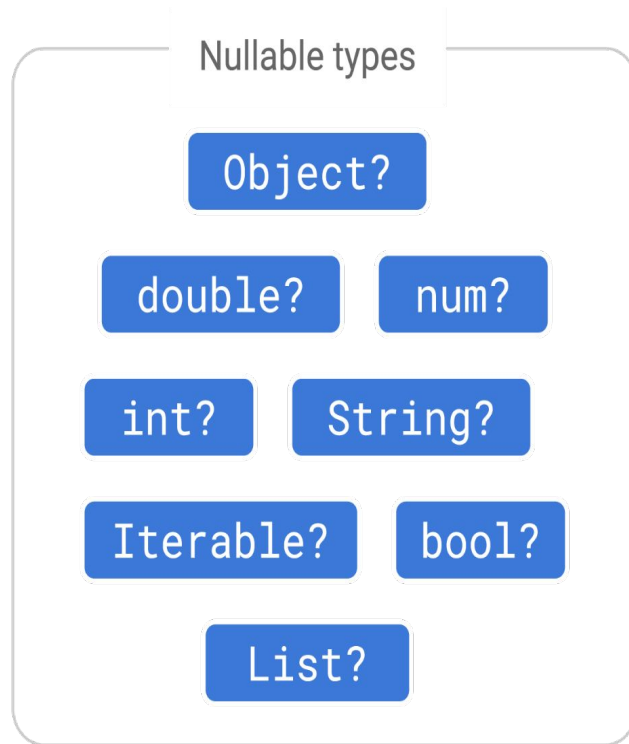
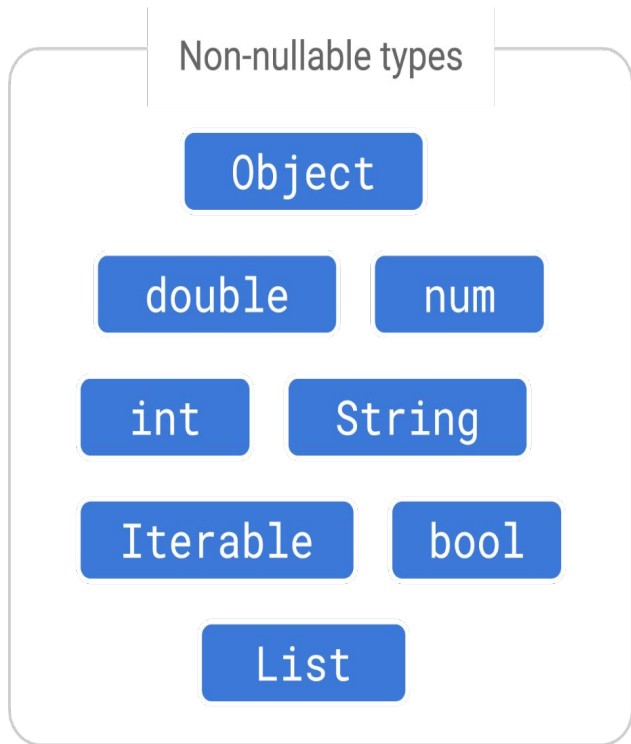
Open talk

- Vậy có cách nào để giải quyết vấn đề trên?

Null Safety

- Từ khi chuyển qua Flutter, tôi dùng version 1.12.13, lúc đó tôi chưa thực sự có niềm tin vào Flutter, vì chưa có Null Safety, nhưng từ version 2.0 trở lên, Flutter hiện tại đã khác xưa.
- Trong dart chia rõ **Non-Nullable Variables**, tức các biến không thể null, và **Nullable Variables**, tức các biến có thể nhận giá trị null

Non-nullable types và nullable types trong dart

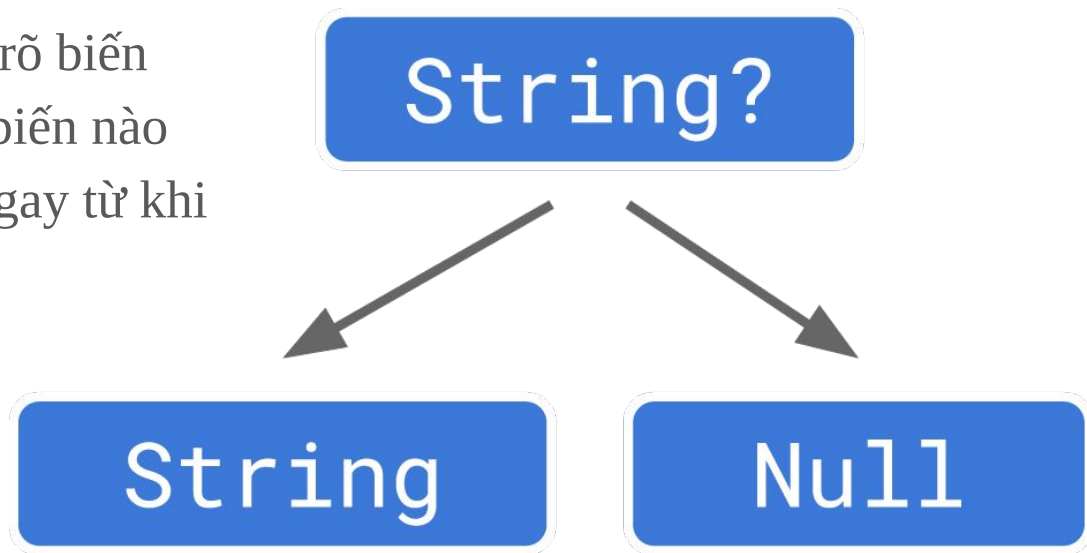


Cùng tìm hiểu về Nullable Variables qua ví dụ sau nhé

```
void main(List<String> args) {  
    /// khi áp dụng `Null safety`, không thể khai báo string s = null  
    /// theo cách thông thường;  
    String s = null; // sẽ báo lỗi, vì String ở đây không thể nhận giá trị  
    null  
  
    String? sCanBeNull = null;  
    print("sCanBeNull -> $sCanBeNull"); // sCanBeNull -> null  
}
```

Nullable Types

- Như vậy, Flutter đã tách biệt rõ biến nào có thể nhận giá trị Null, biến nào không thể nhận giá trị Null ngay từ khi được khởi tạo.



Cùng xem ví dụ tiếp theo khi áp dụng các hàm nhé

```
void main(List<String> args) {  
    String? sCanBeNull = null;  
    List<String> splitteds = sCanBeNull.split("");  
    /// với biến có thể null khi thực thi 1 hàm, sẽ báo lỗi để tránh `Null Pointer Exception`  
    /// The method 'split' can't be unconditionally invoked because the receiver can be 'null'.  
    /// Chúng ta buộc phải kiểm tra `sCanBeNull` != null,  
    List<String> splitteds_1 = sCanBeNull?.split("") ?? [];  
    /// ?? là Null-aware, sẽ giải thích ở phía dưới  
  
    /// Hoặc có 1 cách nữa là chắc chắn rằng giá trị hiện tại của `sCanBeNull` không phải null  
    List<String> splitteds_2 = sCanBeNull!.split("");  
}
```

Open talk

- Bạn đã thấy được lợi ích mà Null Safety mang lại?

Null aware

Null aware

- Trong dart **null aware** được thể hiện bằng dấu '??'.
- Hiểu nôm na là nếu biểu thức bên trái dấu '??' cho ra kết quả null, thì sẽ lấy kết quả ở bên phải dấu '??'

Ví dụ về Null aware

```
void main(List<String> args) {  
    String? sCanBeNull = null;  
    List<String> splitteds_3 = sCanBeNull?.split("") ?? [];  
    /// Ở đây, sẽ thực thi như sau:  
    /// Nếu `sCanBeNull` != null sẽ gọi hàm split("") và trả về giá trị  
    /// Nếu `sCanBeNull` == null sẽ trả về giá trị []  
    /// 1 vài ví dụ về `Null-aware`  
    sCanBeNull ??= "Value"; // nếu sCanBeNull = null, sCanBeNull = "Value"  
    sCanBeNull ??= "Test"; // sCanBeNull sẽ không được cập nhật, vì giá trị hiện tại =  
    "Value"  
    print("sCanBeNull -> $sCanBeNull");  
}
```

Toán tử trong dart

Toán tử số học

- Toán tử là những ký hiệu được dùng để thực hiện một phép toán số hoặc logic nhằm để tùy chỉnh hoặc thao tác dữ liệu.
- Thao tác dữ liệu là một phần thiết yếu của bất kỳ ngôn ngữ lập trình nào và Dart cũng không ngoại lệ.
- Dart cung cấp một tập hợp các toán tử phong phú cho các kiểu cơ bản của nó.

Open talk

- Các phép toán số học mà bạn biết?

Toán tử số học

```
void main(List<String> args) {  
    int a = 10;  
    a *= 2;  
    print("a -> $a");  
    a = a + 1; // tương tự với a++ hoặc ++a  
    a = a - 1; // tương tự với a-- hoặc --a  
    a = a * 2; // tương tự với a*=2  
    a = (a / 2) as int; // tương tự với a/=2, phép chia sẽ trả về double, as int để ép kiểu sang int  
    a = 10 ~/ 6; // chia lấy nguyên, cần thận nhầm với ~/ là toán tử bitwise  
    a = 10 % 3;  
    int b = 10 * (9-8);  
    print("b -> $b");  
}
```

Bảng tổng hợp Toán tử số học

Biểu tượng	Ý nghĩa	Ví dụ
+	Cộng	$2+3 = 5$
-	Trừ	$2-3 = -1$
*	Nhân	$2*3 = 6$
/	Chia	$9/2 = 4.5$
~/	Chia lấy phần nguyên	$9 ~/2 = 4$
%	Chia lấy số dư	$9 \% 2 = 1$

Open talk

- Đố vui: Thể hiện phép toán: $10 + 6 - 2 * 4 / (9 - 3)$ trong dart

Toán tử số học

- Trong dart hỗ trợ nhiều kiểu viết như:

```
int a = 10;  
a = a + 1; // tương tự với a++ hoặc ++a
```

Open talk: toán tử so sánh

- Đưa ra các ví dụ về so sánh

Toán tử so sánh

- 1 toán tử so sánh trả về dữ liệu kiểu bool: **đúng** hoặc **sai**

Toán tử so sánh

```
void main(List<String> args) {  
    String str = "Hello World";  
    int a = 10;  
  
    print(str == a); // so sánh =, equal; sẽ in ra: false  
    print(10 == a); // so sánh =, equal; true  
  
    print(str != a); //so sánh khác, not equal; sẽ in ra: true  
    print(10 > a); // so sánh lớn hơn, sẽ in ra: false  
    print(10 >= a); // so sánh lớn hơn hoặc bằng, sẽ in ra: true  
    print(10 < a); // so sánh nhỏ hơn, sẽ in ra: false  
    print(10 <= a); // so sánh nhỏ hơn hoặc bằng, sẽ in ra: true  
}
```

Bảng tổng hợp Toán tử so sánh

Biểu tượng	Ý nghĩa	Ví dụ
$==$	Kiểm tra bằng	$2 == 6$
$!=$	Kiểm tra khác	$2 != 6$
$>$	Lớn hơn	$2 > 6$
$<$	Nhỏ hơn	$2 < 6$
$>=$	Lớn hơn hoặc bằng	$2 >= 6$
$<=$	Nhỏ hơn hoặc bằng	$2 <= 6$

Toán tử logic

- Gồm 3 phép toán:

Toán tử	Ký hiệu	Ý nghĩa
hoặc		mệnh đề a đúng hoặc mệnh đề b đúng
và	&&	mệnh đề a đúng và mệnh đề b đúng
phủ định biểu thức	!biểu thức	phủ định của mệnh đề a. Ví dụ mệnh đề a đúng, phủ định của mệnh đề a là sai.

Toán tử logic

```
void main(List<String> args) {  
    String str = "Hello World";  
    int a = 10;  
  
    print(str == a || a == 10); // `||` là hoặc, 1 trong 2 điều kiện đúng -> true; sẽ in ra: false  
    print(str == a && a == 10); // `&&` là hoặc, cả 2 điều kiện đúng -> true; sẽ in ra: false  
  
    /// 3 cách viết này cho ra kết quả giống nhau:  
    if((str == a) == false) {  
        print("true");  
    }  
    if(!(str == a)) {  
        print("true");  
    }  
    if(str != a) {  
        print("true");  
    }  
}
```

Mẹo viết tắt với toán tử logic

```
void main(List<String> args) {  
    int a = 10;  
  
    /// cách viết phổ biến  
    if(a == 10) {  
        a = 20;  
    } else {  
        a = 10;  
    }  
  
    /// cách viết tắt  
    int b = a == 10 ? 20 : 10;  
  
    print(a == b); // sẽ in ra: false  
}
```

```
void main(List<String> args) {  
    int a = 10;  
  
    /// cách viết phổ biến  
    if(a == 10) {  
        print("a == 10");  
    } else {  
        print("a != 10");  
    }  
  
    /// cách viết tắt:  
    /// sau `?` là biểu thức thực hiện khi điều kiện  
    đúng,  
    /// sau `:` là biểu thức thực hiện khi điều kiện  
    sai  
    print("${a == 10 ? 'a == 10' : 'a != 10'}");  
}
```

Ép kiểu, kiểm tra kiểu

```
void main(List<String> args) {  
    int a = 10;  
  
    double b = a as double; // ép kiểu int -> double mà không dùng toDouble  
    print("b -> $b");  
  
    bool isInt = a is int;  
    print("a isInt -> $isInt"); // sẽ in ra: true  
}
```

Bảng tổng hợp ép kiểu kiểm tra kiểu

Biểu tượng	Ý nghĩa	Ví dụ
as	Ép kiểu này tới kiểu khác	obj as String
is	True nếu đối tượng chứa kiểu ấy	obj is String
is!	False nếu đối tượng chứa kiểu ấy	obj is! String



Cần chú ý

- Khi ép kiểu có thể gây ra lỗi, nên kiểm tra bằng ``is`` trước khi ép kiểu

```
void main(List<String> args) {  
    dynamic a = 10;  
    a = "Hello world";  
    if(a is int) {  
        double b = a.toDouble();  
        print("b -> $b");  
    } else {  
        print("can not parse a");  
    }  
}
```


Access Modifier

Open talk

- Các access modifier trong các ngôn ngữ lập trình khác?
 - Public
 - Protected
 - No modifier
 - Private
 - Internal

Java modifier

Access Levels				
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Dart modifier

- Access modifier giúp xác định phạm vi có thể truy cập của biến (variable), phương thức (method/function), hàm khởi tạo (constructor) hoặc lớp (class).
- Trong dart, chỉ tồn tại duy nhất 2 levels: **public** và **private**
- Từ flutter 3.0 trở lên, flutter quy định **private** chỉ nên dùng cho biến và function trong class. Nếu sử dụng **private var** trong function sẽ báo warning:
“Avoid leading underscores for local identifiers.dartno_leading_underscores_for_local_identifiers”

Nên tránh

- Không nên sử dụng biến private trong function.

```
void main(List<String> args) {  
    var _localVar = 10;  
    debugPrint("_localVar -> $_localVar");  
}
```

Ví dụ về dart modifier

```
class Test {  
  String _a = "";  
}
```

```
class _Internal{  
  String _a = "";  
}
```

Ví dụ về dart modifier

```
import 'test.dart';

void main(List<String> args) {
  /// để khai báo biến private, sử dụng underscore `_`
  /// _a có thể truy cập từ hàm main
  InMainClass inMainClass = InMainClass();

  inMainClass._a; // _a là biến private trong class A, nhưng nằm cùng file với

  /// ở 1 trường hợp khác, _a không thể truy cập, vì nằm ở file khác, chú ý phần import 'test.dart';
  Test test = Test();

  test._a; // ở 1 trường hợp khác, _a không thể truy cập, vì nằm ở file khác, chú ý phần import 'test.dart';

  /// khi sử dụng inline function, cần chú ý tới access modifiers
  /// không thể truy cập các biến nằm trong inline function
  void function(){ String inFunction = inMainClass._a; // có thể truy cập }

  String outFunction = inFunction; // không thể truy cập
}

class InMainClass { String _a = ""; }
```

Tổng kết Dart access modifier

- Trong dart chỉ có duy nhất 2 access modifiers levels: **public** và **private**, mặc định là public
- - Khi sử dụng inline function, cần chú ý tới access modifier, khi ở ngoài inline function sẽ không thể truy cập các biến nằm trong inline function.

For loop

Mệnh đề if else trong dart

- Mệnh đề if trong dart được sử dụng để kiểm tra giá trị dạng boolean của điều kiện.
- Mệnh đề này trả về giá trị **True** hoặc **False**.
- Có các kiểu của mệnh đề if-else trong dart như sau:
 - Mệnh đề if
 - Mệnh đề if-else
 - Mệnh đề if-else-if

Mệnh đề if

- Mệnh đề if được sử dụng để kiểm tra giá trị dạng boolean của điều kiện.
- Khối lệnh sau if được thực thi nếu giá trị của điều kiện là **True**.

```
int a = 10;  
if (a == 10) {  
    print("a == 10");  
}
```

Mệnh đề if else

- Mệnh đề if-else cũng kiểm tra giá trị dạng boolean của điều kiện.
- Nếu giá trị điều kiện là **True** thì chỉ có khối lệnh sau if sẽ được thực hiện, nếu là **False** thì chỉ có khối lệnh sau else được thực hiện.

```
int a = 8;
if (a == 10) {
    print("a == 10");
} else {
    print("a != 10");
}
```

Mệnh đề if else if

- Mệnh đề if-else-if cũng kiểm tra giá trị dạng boolean của điều kiện.
- Nếu giá trị điều kiện if là **True** thì chỉ có khối lệnh sau if sẽ được thực hiện.
- Nếu giá trị điều kiện if else nào là **True** thì chỉ có khối lệnh sau else if đó sẽ được thực hiện...
- Nếu tất cả điều kiện của if và else if là **False** thì chỉ có khối lệnh sau else sẽ được thực hiện.

```
int a = 8;
if (a == 1) {
    print("a == 1");
} else if (a == 2) {
    print("a == 2");
} else {
    print("else");
}
```

Open talk

- Các vòng lặp for mà bạn biết?

For loop

- Trong các ngôn ngữ lập trình, vòng lặp cho phép một khối mã được thực thi lặp đi lặp lại nhiều lần.
- Dart cũng như các ngôn ngữ lập trình khác, đều hỗ trợ các vòng lặp
 - for
 - for in
 - while
 - do while

For: Áp dụng theo các điều kiện

```
void main() {  
    /// vòng for cơ bản, theo index  
    for (int i = 0; i < 10; i = i + 1) {  
        print('i= $i'); // sẽ in ra giá trị của i từ 0 -> 9  
    }  
}
```


For: Áp dụng theo các điều kiện

- Ưu điểm là dễ dàng thay đổi điều kiện lặp
- Nhưng tránh code vòng for vô tận như này:

```
int count = 10;
for (var i = 0; i < count; i++) {
    if(count == 10) {
        count = 0;
    }
    print("i -> $i");
}
```

For in: Áp dụng lặp từng phần tử trong mảng

```
void main(List<String> args) {  
    List ints = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];  
    /// cũng là vòng lặp nhưng lặp qua từng giá trị trong 1 mảng  
    for (int i in ints) {  
        print("i = $i"); // sẽ in ra giá trị của i 0 -> 9  
    }  
}
```

Áp dụng break trong vòng lặp

- Trong vòng lặp, có thể dùng từ khoá **break** để thoát khỏi vòng lặp.
- Toàn bộ phương thức phía sau sẽ không được thực thi.

```
void main(List<String> args) {  
    int count = 10;  
    for (var i = 0; i < count; i++) {  
        if (i == 4) {  
            break;  
        }  
        print("i -> $i"); // sẽ chỉ in ra 0 -> 3  
    }  
}
```

- Ở ví dụ này sẽ chỉ in ra kết quả từ 0 -> 3, vì khi break khỏi vòng for khi index = 4.

Áp dụng continue trong vòng lặp

- Trong vòng lặp, có thể dùng từ khoá `continue` để bỏ qua phần tử hiện tại

```
void main(List<String> args) {  
    int count = 10;  
    for (var i = 0; i < count; i++) {  
        if (i == 4 || i == 5 || i == 6){  
            continue;  
        }  
        print("i -> $i"); // 0 -> 3, 7 -> 9  
    }  
}
```

- Ở ví dụ này sẽ chỉ in ra kết quả từ 0
-> 3, 7 -> 9, vì bỏ qua phần tử 3, 4, 5

While: áp dụng cho 1 điều kiện chưa thể xác định, chỉ có thể xác định khi thực thi 1 hoặc nhiều khối lệnh

```
void main(List<String> args) {  
    int x = 0;  
    // trong khi 1 điều kiện đúng, thực thi 1 khối lệnh  
    while (x < 10) {  
        x++;  
        print("x -> $x");  
        // sẽ in ra các giá trị từ 1 -> 10,  
        // khi x = 10, while sẽ dừng, vì điều kiện để chạy while là x < 10  
    }  
}
```

Do while: thực thi khối lệnh trước khi kiểm tra điều kiện trong while

```
void main(List<String> args) {  
    int x = 0;  
    /// thực thi khối lệnh trước khi kiểm tra điều kiện trong while  
    do {  
        x++;  
        print("x -> $x"); // sẽ in ra giá trị của x từ 1 -> 10  
    } while (x > 0 && x < 10);  
}
```

Tổng kết vòng lặp trong dart

- Vòng lặp trong dart giống các ngôn ngữ lập trình phổ biến khác như: java, kotlin, swift, python....



Cần chú ý

- Khi sử dụng vòng lặp, cần chú ý các điều kiện lặp để hạn chế **`infinity loop`** gây treo, lag ứng dụng.

switch case

switch case

- switch case là cấu trúc rẽ nhánh giúp tối ưu các cases hơn khi dùng if-else-if

```
import 'dart:math';

void main(List<String> args) {
  int randomInt = Random().nextInt(100);
  switch (randomInt % 2) {
    case 0:
      print("$randomInt là số chẵn");
      break;
    case 1:
      print("$randomInt là số lẻ");
      break;
    default:
      print("Default case");
      break;
  }
}
```

1 ví dụ khác: switch case cho enum

```
import 'dart:math';

void main(List<String> args) {
  WorkStatus workStatus = WorkStatus.values[Random().nextInt(WorkStatus.values.length)];
  switch (workStatus) {
    case WorkStatus.todo:
      print("todo");
      break;
    case WorkStatus.inProgress:
      print("inProgress");
      break;
    case WorkStatus.done:
      print("done");
      break;
    default:
  }
}

enum WorkStatus {todo, inProgress, done}
```

Tổng kết switch case

- Sử dụng switch case có thể giúp tránh khai báo thiếu các case.

```
import WorkStatus workStatus
```

Missing case clause for 'done'.

Run Try adding a case clause for the missing constant, or adding a default clause. [dart\(missing_enum_constant_in_switch\)](#)

[View Problem](#) [Quick Fix... \(⌘.\)](#)

```
switch (workStatus) {  
  case WorkStatus.todo:  
    print("todo");  
    break;  
  case WorkStatus.inProgress:  
    print("inProgress");  
    break;  
}
```

```
enum WorkStatus {todo, inProgress, done}
```

try, catch, finally, throw

try catch

- try catch trong dart cũng giống các ngôn ngữ khác, mục đích để tránh những lỗi ngoại lệ (exceptions)

```
void main(List<String> args) {  
  int a = 10;  
  try {  
    a = int.parse("Hello world");  
    /// khi parse int sẽ gây ra lỗi, nhưng được bọc bởi try catch  
    /// nên sẽ không thể print ra biến a  
    print("after parse, a = $a");  
  } catch (ex) {  
    print("exception -> ${ex.toString()}"); // exception -> FormatException: Hello world  
  }  
}
```

try on

- try on để chỉ định catch 1 exception cụ thể.

```
void main(List<String> args) {  
    int a = 10;  
    try {  
        a = int.parse("Hello world");  
    } on FormatException {  
        print("on format exception"); // sẽ in ra: on format exception  
    } on SocketException {  
        print("on socket exception");  
    }  
}
```

finally

- Khối lệnh trong finally sẽ được thực hiện bất chấp có xảy ra lỗi trong khối try hay không.

```
void main(List<String> args) {  
    int a = 10;  
    try {  
        a = int.parse("Hello world");  
        /// khi parse int sẽ gây ra lỗi, nhưng được bọc bởi try catch  
        /// nên sẽ không thể print ra biến a  
        print("after parse, a = $a");  
    } catch (ex) {  
        print("exception -> ${ex.toString()}"); // exception -> FormatException: Hello world  
    } finally {  
        print("finally after parse, a = $a"); // sẽ in ra finally after parse, a = 10  
    }  
}
```

throw

- Để chủ động phát sinh lỗi trong Flutter, có thể dùng từ khóa **throw** theo sau là Exception muốn quăng ra.

```
void main(List<String> args) {  
  void checkANumber({int? number}){  
    if (number == null || number <= 0) {  
      throw Exception("number must >= 0");  
    }  
    print("number is $number");  
  }  
  
  checkANumber();  
}
```

PROBLEMS

3

OUTPUT

DEBUG CONSOLE

Restarted application in 111ms.

Error: Exception: number must >= 0

at Object.throw_ [as throw] (http:

assert

- assert dùng để ngăn chương trình tiếp tục thực thi nếu có bất kỳ điều kiện nào bên trong nó là false.
- Lưu ý là assert chỉ khi build ở chế độ debug, production và release sẽ không hoạt động.

```
void main(List<String> args) {  
    void checkANumber({int? number}){  
        assert (number != null || number! >= 0,  
"Number must >= 0");  
        print("number is $number");  
    }  
  
    checkANumber();  
}
```

PROBLEMS

3

OUTPUT

DEBUG CONSOLE

Restarted application in 88ms.

Error: Unexpected null value.

at Object.throw_ [as throw] (http:

Practise

Practise 1: String

Từ 1 String str có sẵn, dùng tất cả các cách mà bạn biết để print ra String sau.
Chú ý các ký tự viết hoa, viết thường, khoảng trắng...

“Đây là kết quả buổi học thứ 2 về Dart: DART BASIC (phần 1)”

```
void main(List<String> args) {  
    String str = " đây là k ết quả của buổi học thứ 2 về dart: dart basics (phần  
1)...";  
    /// hãy viết code bằng tất cả các cách có thể để in ra:  
    /// `Đây là k ết quả buổi học thứ 2 về Dart: DART BASIC (phần 1)`  
}
```

Practise 2: Các variables đã học

Từ 1 array có sẵn (*slide sau*), dùng tất cả các cách mà bạn biết để print ra String sau. Chú ý các ký tự viết hoa, viết thường, khoảng trắng...

“Đây là kết quả buổi học thứ 2 về Dart: DART BASIC (phần 1)”

```
void main(List<String> args) {  
  List arr = [  
    1,  
    2,  
    3,  
    "đây",  
    "kết",  
    "là",  
    true,  
    false,  
    {true: "buổi", 1: "học", 10.2: ":", false: "dart basics"},  
    ['thứ', 'quả', 'về'],  
    "(phần 1)",  
    {"flutter": "dart"},  
  ];  
  
  /// Lấy data từ array trên và tạo ra 1 String với giá trị sau:  
  /// Đây là kết quả buổi học thứ 2 về Dart: DART BASIC (phần 1)  
  
  // ví dụ  
  String s1 =  
    "${arr[3].toString().substring(0, 1).toUpperCase()}${arr[3].toString().substring(1, 3)}";  
  print('$s1'); // sẽ in ra "Đây" các ký tự sau tương tự như vậy.
```

Practise 3: Do and think about what you have learned

Viết 1 chương trình:

1. Khai báo toàn bộ các kiểu dữ liệu: Non-Nullable Types và Nullable Types, dynamic, var.
2. Có sử dụng các từ khoá static, final, late const.
3. Viết 1 hàm tính giai thừa của 6.
4. Viết các hàm chuyển đổi qua lại giữa string, int, double.

```
/// ví dụ về 4. hàm chuyển đổi qua lại giữa string, int, double  
String getIntFromString(int input) => input.toString();
```

Practise 4: Do and think about what you have learned

Viết 1 chương trình:

1. Mở rộng thêm các phương thức từ number:

- Thêm các phương thức add(), subtract, divide, multiple tương ứng với 4 phép tính: cộng, trừ, nhân, chia
- 4 phương thức này đều báo lỗi nếu truyền vào số ≤ 0

```
// kết quả mong muốn
int a = 10;
print("add method: ${a.add(2)}"); // add method: 12
print("subtract method: ${a.subtract(2)}"); // subtract method: 8
print("divide method: ${a.divide(2)}"); // divide method: 5
print("multiple method: ${a.multiple(2)}"); // multiple method: 20
```

Practise 5: Do and think about what you have learned

Viết 1 chương trình:

1. Đầu vào là 1 mảng số tự nhiên từ 0 -> 100
2. Liệt kê ra toàn bộ các số lẻ
3. Liệt kê ra toàn bộ số chẵn
4. Kiểm tra xem 0 là số chẵn hay số lẻ

Practise 6: Do and think about what you have learned

Viết 1 chương trình:

1. In ra màn hình toàn bộ số nguyên tố của 1 số n cho trước.

```
/// ví dụ: nếu  $n = 20$ , kết quả sẽ là  
[0, 1, 2, 3, 5, 7, 11, 13, 17, 19]
```

```
/// ví dụ: nếu  $n = 100$ , kết quả sẽ là  
[0, 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,  
71, 73, 79, 83, 89, 97]
```

Thank you for watching

toannm.jsc@gmail.com