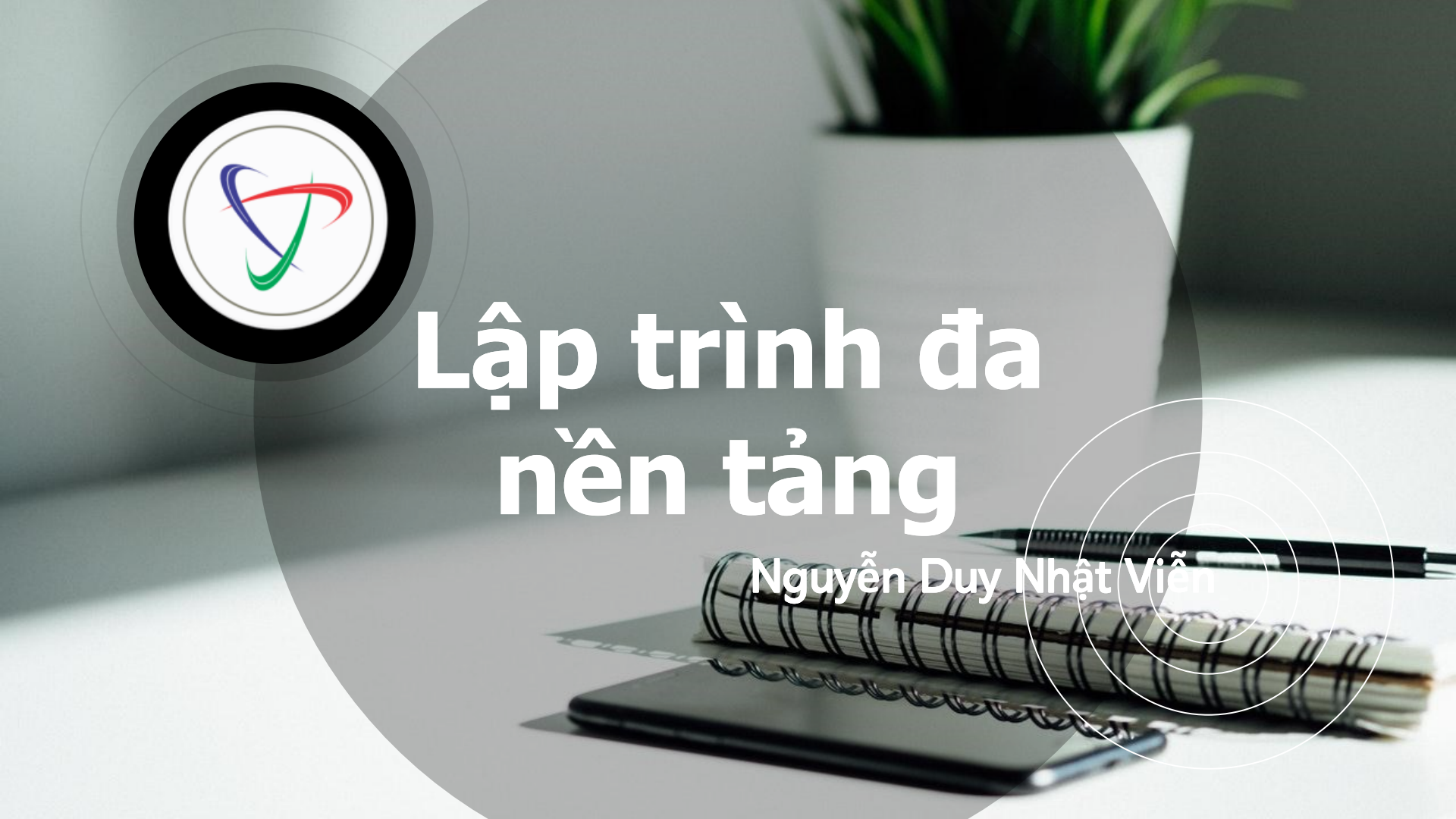
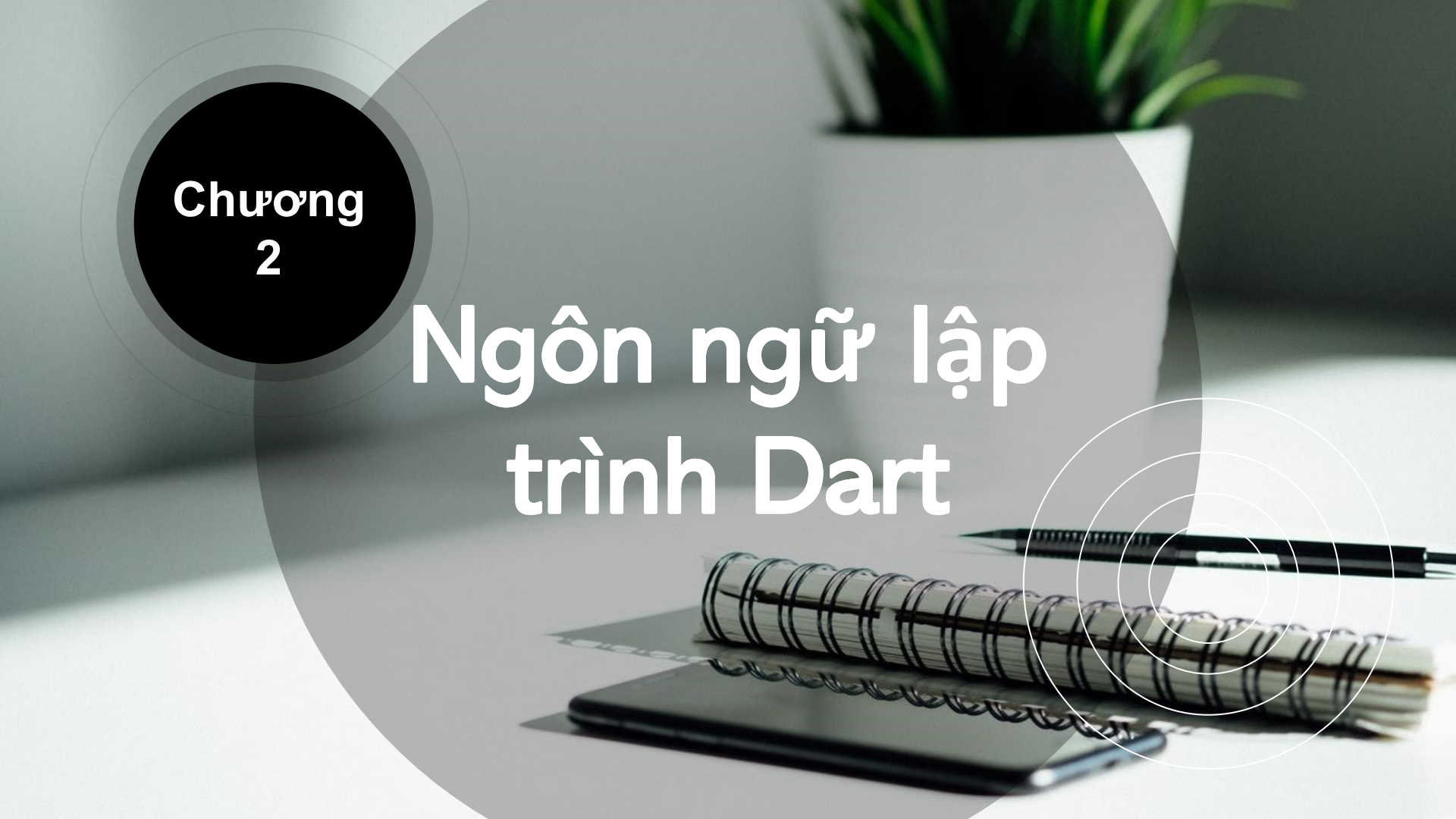


# Lập trình đa nền tảng

Nguyễn Duy Nhật Viễn

Chương 2

# Ngôn ngữ lập trình Dart



# Biến, hằng và các kiểu dữ liệu

# Biến

- Trong Dart mọi thứ đều là object. Một object luôn là thể hiện của một class nào đó.
- Giá trị mặc định của object là null. Như vậy, mọi biến số trong Dart đều là kiểu tham chiếu.
- Dart có một loại biến *dynamic* chấp nhận mọi kiểu dữ liệu.
- Tạo biến với từ khoá var

```
// Khai báo biến a, khởi tạo nó lưu một chuỗi
// (do vậy a có kiểu String, nó chỉ lưu chuỗi)
var a = "Learn Dart";
a = "Learn Dart 2"; // Gán chuỗi khác
a = 100; // Lỗi vì gán số vào a
```

```
// Khai báo và không khởi tạo
// biến b sẽ có giá trị null
// kiểu của b tùy thuộc vào giá trị gán vào nó
var b;
b = 100; // Gán số vào b
b = "aaa"; // Gán chuỗi vào b, không lỗi
```

# Biến

- Ta có thể khai báo và chỉ định kiểu dữ liệu cụ thể cho biến
- Các biến được chỉ định kiểu dữ liệu phải được gán giá trị cùng kiểu
  - String s = 'Chuỗi ký tự'; // Khai báo biến chuỗi
  - double d = 1.1234; // khai báo biến số thực
  - int i = 1; // biến số nguyên
  - bool found = true; // biến logic
- Trường hợp muốn sử dụng biến không xét đến kiểu (chấp nhận gán vào nó nhiều loại kiểu) thì dùng từ khóa dynamic
  - dynamic dyn = 123; // Khởi tạo là số int
  - dyn = "Dynamic"; // Gán chuỗi
  - dyn = 1.12345; // Gán số double

# Hằng

- Hằng lưu giá trị mà không thể thay đổi, sử dụng từ khóa const hoặc final
- *Tạo hằng số const* giá trị của hằng phải cụ thể ngay khi viết code
  - const ten_hang_s0 = biểu_thức_giá_trị;
  - //Ví dụ
  - const dow_0 = 'Sunday';
  - const minutes = 24 * 60;
- *Tạo hằng số final* biến final chỉ được gán một lần duy nhất
  - final name_1 = biểu_thức_giá_trị;
  - final String name_2 = biểu_thức_giá_trị;
  - //Tạo hằng số từ biến
  - var so_ngau_nhien = Random(1000).nextInt(500);
  - final a = so_ngau_nhien * 2;

# Các kiểu dữ liệu

- **Numbers**: được dùng để biểu diễn dạng số.
  - Kiểu *int*: biểu diễn số nguyên tối đa 64-bit phụ thuộc vào phần cứng
    - int numint = 100;
    - //đổi chữ thành số nguyên:
    - int numint = int.parse("120");
  - Kiểu *double* biểu diễn số thực dấu chấm động 64bit.
    - double d1 = 0.1234;
    - var d2 = 12.123;
    - //đổi chữ thành số thực
    - var abc = double.parse("123.123");

# Các kiểu dữ liệu

**Numbers**:

- Cả số nguyên và số thực có các phép toán để tạo thành biểu thức là cộng +,trừ - ,nhân * ,chia /, chia lấy phần nguyên ~/ , chia lấy phần dư %.
  - double c = (a + b) / (a-b);
- Trên các đối tượng số này có các phương thức để chuyển đổi kiểu toString(), toInt(), toDouble()
  - double a = 10.10;
  - int b = a.toInt();
  - String c = (a + b).toString();

# Các kiểu dữ liệu

- **Numbers:** Các phương thức hữu dụng cho kiểu int và double:
  - parse(string) //chuyển kiểu từ string
  - abs()  //trị tuyệt đối
  - round(), ceil(), floor() //làm tròn, làm tròn lên, làm tròn xuống
  - toString(), toInt, toDouble //chuyển kiểu sang String, Int hoặc Double
- Nên sử dụng int hoặc double
  - var a = 1; // int
    var b = 1.0; // double
  - int x = 8;
  - double y = b + 6;
  - num z = 10 - y + x;
  - var a = 1.35e2  //1.35 * $10^2$
  - var a = 0xF1A //F1A in base 16 (3866 in base 10)

# Các kiểu dữ liệu

- Ví dụ:
  - String value = "17";
  - var a = int.parse(value); // String-to-int conversion
  - var b = double.parse("0.98"); // String-to-double conversion
  - var c = int.parse("13", radix: 6); // Converts from 13 base 6
  - String v1 = 100.toString(); // v1 = "100";
  - String v2 = 100.123.toString(); // v2 = "100.123";
  - String v3 = 100.123.toStringAsFixed(2); // v3 = "100.12";

# Các kiểu dữ liệu

Ví dụ:

```
// 1. Nếu chuỗi không chứa/(phải là) 1 số thì val = null
double? val = double.tryParse("12@.3x_"); // null
double? val = double.tryParse("120.343"); // 120.343
// 2. Lệnh gọi lại onError được gọi khi phân tích cú pháp không thành công
var a = int.parse("1_6", onError: (value) => 0); // 0
var a = int.parse("16", onError: (value) => 0); // 16
```

- Hãy nhớ rằng parse () về sau không được dùng nữa, vì vậy, nên chọn tryParse () để có kết quả an toàn

# Các kiểu dữ liệu

- **Strings**: được dùng để biểu diễn chuỗi ký tự Unicode(UTF-16). Chuỗi nằm trong cặp nháy đơn ' ' hoặc nháy kép " ". Nếu có nháy đơn hoặc nháy kép trong 1 chuỗi, dùng \’ hoặc \”.
  - String a = 'Chuỗi ký tự \'\' (nháy đơn)'; // Chuỗi ký tự " (nháy đơn)
  - String b = "Chuỗi ký tự \"\" (nháy kép)"; // Chuỗi ký tự "" (nháy kép)
- Nối chuỗi (dùng toán tử +)
  - String s1 = "S1";
  - String s2 = "S2";
  - String s3 = s2 + '_' + s1;
  - print(s3); //S1_S2

# Các kiểu dữ liệu

- **Strings**:
  - Muốn nhập chuỗi trên nhiều dòng, dùng cú pháp sau (các dòng nằm giữa cặp ''' hoặc """);
    - String s1 = ''' Các dòng chữ
    - trong chuỗi s1 ''';
    - print(s1);
    - //Hoặc
    - String s2 = """ Các dòng chữ
    - trong chuỗi s2 """;
    - print(s2);

# Các kiểu dữ liệu

**Strings**:

- Có thể chèn một biến hoặc một biểu thức vào chuỗi bằng cách ký hiệu $tên_biến, ${biểu thức giá trị}
  - var a = 10;
  - var b = 20;
  - String kq = "Hai số $a, $b có tổng ${a + b}";
  - print(kq); //Hai số 10, 20 có tổng 30

# Các kiểu dữ liệu

**Booleans**: được biểu diễn bởi 2 giá trị *true* và *false* với từ khóa *bool*.

```
bool found = true;
if (found) {
    //Do something
}
```

# Các kiểu dữ liệu

- **List**: Trong Dart, array được dùng theo list. **List<T>** biểu diễn một nhóm các đối tượng được sắp xếp và có thiết kế giống như mảng (array) trong các ngôn ngữ khác.
- Java
  - // 1. Array
  - double[] test = new test[10];
    // 2. Generic list
    List<double> test = new ArrayList<>();
- Dart
  - // 1. Array
    // (no equivalent)
    // 2. Generic list
    List<double> test = new List<double>();

# Các kiểu dữ liệu

**List**: chỉ số của list được đánh từ 0 và truy cập thông qua tên_list[chỉ_số].

Ví dụ

```
//use var or final
final myList = [-3.1, 5, 3.0, 4.4];
final value = myList[1];
```

# Các kiểu dữ liệu

## **List**: THUỘC TÍNH

| THUỘC TÍNH | MÔ TẢ |
|---|---|
| first | Trả vể phần tử đầu của List |
| isEmpty | Trả về true nếu List không chứa phần tử nào cả |
| isNotEmpty | Trả về true nếu List chứa ít nhất 1 phần tử |
| length | Trả về kích thước của List |
| last | Trả vể phần tử cuối của List |
| reversed | Trả về List được sắp xếp theo thứ tự ngược lại |
| single | Kiểm tra, nếu List chỉ chứa 1 phần tử thì trả về phần tử ấy |

# Các kiểu dữ liệu

- **Map**: lưu tập hợp các giá trị kết hợp (còn gọi là mảng kết hợp), thay vì sử dụng chỉ số được đánh từ 0 để tham chiếu đến phần tử, chỉ số của Map được đánh tự do.
- Mỗi phần tử trong Map lưu theo cặp key:value, dùng ký hiệu {} để khởi tạo Map hoặc khởi tạo bằng Map()
- Truy cập đến phần tử Map dùng ký hiệu chấm .key
- Ví dụ
  - var dow = { 'T2' : 'Thứ 2', 'T3' : 'Thứ 3', 'CN' : 'Chủ Nhật' };
  - print(dow.length); //Số phần tử
  - print(dow['T2']); //Truy cập phần tử chỉ số T2
  - dow.putIfAbsent('T4', () => 'Thứ 4'); //Thêm phần tử mới nếu chưa có
  - //Cũng có thể tạo ra Map bằng
  - //var dow = new Map();

# Các kiểu dữ liệu

**Map**:

Ví dụ

```
var details = new Map();
details[‘Username'] = 'admin’; //thêm giá trị cho Map details
details['Password'] = 'admin@123’; //thêm giá trị cho Map details
print(details);
```

# Các kiểu dữ liệu

## **Map**: Thuộc tính

| Thuộc tính | Mô tả |
|---|---|
| keys | Trả về các giá trị tương ứng của key |
| values | Trả về các giá trị tương ứng của value |
| length | Trả về kích thước của Map |
| isEmpty | Trả về true nếu Map là rỗng |
| isNotEmpty | Trả về true nếu Map không rỗng |

# Các kiểu dữ liệu

## **Map**: Phương thức

| PHƯƠNG THỨC | MÔ TẢ |
|---|---|
| addAll() | Thêm tất cả các cặp key-value khác vào Map. |
| clear() | Xoá tất cả các cặp key-value trong Map |
| remove(key) | Xóa key và value liên quan của Map nếu có. |
| forEach() | Áp dung cho từng cặp key-value của Map |

# Các kiểu dữ liệu

```
enum Fruits { Apple, Pear, Grapes, Banana, Orange }

void main() {
    Fruits liked = Fruits.Apple;
    var disliked = Fruits.Banana;

    print(liked.toString()); // prints 'Fruits.Apple'
    print(disliked.toString()); // prints 'Fruits.Banana'
}
```

# Các kiểu dữ liệu

**Enums**: mỗi giá trị trong 1 enum kết hợp với chỉ số (index), index được đánh tự động tăng dần từ 0.

```
enum Fruits { Apple, Pear, Grapes, Banana, Orange }

void main() {
    var a = Fruits.Apple.index;   // 0
    var b = Fruits.Pear.index;   // 1
    var c = Fruits.Grapes.index;   // 2
}
```

# Các kiểu dữ liệu

**Set<T>**: biểu diễn một nhóm các đối tượng duy nhất.

**Runes**: là biểu diễn dạng chuỗi Unicode 32 bit (UTF-32).

**Dynamic Type**: *dynamic* trong Dart có thể hiểu là kiểu dữ liệu tùy chọn.

- Đây là kiểu cơ bản cho mọi kiểu dữ liệu trong Dart.
- Kiểu dữ liệu của một biến nếu không được chỉ định rõ ràng thì sẽ được gán với từ khóa *dynamic*.

# Nullable và Non-Nullable

- Trong Dart, việc truy cập các biến trước khi khởi tạo sẽ gây lỗi
  - int value;
  - print("$value"); // Illegal, doesn't compile
  - int value;
  - value = 0;
  - print("$value"); //Ok
- Kiểm tra:
  - String name = "Alberto";
  - if (name != null) { print(name) }

# Nullable và Non-Nullable

- Nullable và Non-nullable
  - // Non-nullable version - default behavior
    int value = 0;
    print("$value"); // prints '0'
  - // Nullable version - requires the ? at the end of the type
    int? value;
    print("$value"); // prints 'null'
  - String? name = "Alberto";
    String? first = name?[0]; // first = 'A';
  - String? name;
    String? first = name?[0]; // first = 'null';

# Nullable và Non-Nullable

- Khi chắc 1 biến là không null, ta có thể thêm toán tử ! Vào cuối tên biến để chuyển về non-nullable
  - int? nullable = 0;
  - int notNullable = nullable!;
- Chuyển kiểu bằng toán tử as
  - num? value = 5;
  - int otherValue = value as int;

# Nullable và Non-Nullable

- Toán tử ?? chuyển một nullable thành non-nullable
  - int? *nullable* = 10;
  - int nonNullable = *nullable* ?? 0;
  - Nếu bên trái của toán tử ?? (*nullable*) là non-null thì trả về giá trị của nó (*nullable*) , còn không, trả về giá trị bên phải của ?? (0).
- Lưu ý, khi làm việc với các biến nullable, toán tử truy cập thành viên '.' là không được phép sử dung, thay vào đó, ta dùng '?.'.
  - double? pi = 3.14;
  - final round1 = pi.round(); // No
  - final round2 = pi?.round(); // Ok

# Các toán tử

Toán tử số học:

| Symbol | Meaning | Example |
|---|---|---|
| + | Add two values | `2 + 3 //5` |
| - | Subtract two values | `2 - 3 //-1` |
| * | Multiply two values | `6 * 3 //18` |
| / | Divide two values | `9 / 2 //4.5` |
| ~/ | Integer division of two values | `9 ~/ 2 //4` |
| % | Remainder (modulo) of an int division | `5 % 2 //1` |

# Các toán tử

Toán tử quan hệ

| Symbol | Meaning | Example |
|---|---|---|
| == | Equality test | 2 == 6 |
| != | Inquality test | 2 != 6 |
| > | Greather than | 2 > 6 |
| < | Smaller than | 2 < 6 |
| >= | Greater or equal to | 2 >= 6 |
| <= | Smaller or equal to | 2 <= 6 |

# Các toán tử

Toán tử kiểm tra

| Symbol | Meaning | Example |
|---|---|---|
| as | Cast a type to another | obj as String |
| is | True if the object has a certain type | obj is double |
| is! | False if the object has a certain type | obj is! int |

(grapes as Fruit).color = "Green";

if (grapes is Fruit) { grapes.color = "Green"; }

# Các toán tử

- Toán tử logic

| Symbol | Meaning |
| --- | --- |
| `!expr` | Toggles true to false and vice versa |
| `expr1 && expr2` | Logical AND (true if both sides are true) |
| `expr1 \|\| expr2` | Logical OR (true if at least one is true) |

# Các toán tử

Thao tác bit và dịch

| Symbol | Meaning |
| --- | --- |
| `!expr` | Toggles true to false and vice versa |
| `expr1 && expr2` | Logical AND (true if both sides are true) |
| `expr1 \|\| expr2` | Logical OR (true if at least one is true) |

# Điều khiển luồng

# Cấu trúc if

Dạng thức:

if(điều_kiện) { hoạt_động_1 } else {hoạt_động_2}

```
final random = 13;

if (random % 2 == 0)
    print("Got an even number");
else
    print("Got an odd number");
```

# Cấu trúc if

- Biểu thức điều kiện:
  - valueA ?? valueB; //Nếu valueA là non-null thì trả về valueA, còn không tính biểu thức valueB và trả về kết quả.
  - //Tránh nhận giá trị null
  - String? status; // status là null
  - // isAlive là 1 String đã khai báo trước đó
  - if (status != null) isAlive = status;
  - else isAlive = "RIP";
  - //Vận dụng
  - String? status; // This is null
  - String isAlive = status ?? "RIP";

# Cấu trúc if

- Biểu thức điều kiện:
  - condition ? A : B;  //Nếu condition là true thì trả về giá trị A, conf không trả về giá trị B

  - String status;
  - if (correctAns >= 18) status = "Test passed!";
  - else status = "You didn't study enough..."

  - String status = (correctAns >= 18) ? "Test passed!" : "You didn't study enough...";

# Cấu trúc switch

- Khi có các trường hợp cần xét, thay switch
- Chú ý: code sẽ không được dịch nếu lại không có break.

```
enum Status { Ready, Paused, Terminated }

void main() {
    final status = Status.Paused;

    switch (status) {
        case Status.Ready:
            run();
            break;
        case Status.Paused
            pause();
            break;
        case Status.Terminated
            stop();
            break;
        default
            unknown();
    }
}
```

# Cấu trúc switch

Trong thân case có hàm start() nhưng không có breal

```
switch (status) {
    case Status.Ready:
        start();
        //missing "break;" here
    case Status.Paused
        pause();
        break;
    }
}
```

Trong thân case rỗng nên có thể không có break

```
switch (status) {
    case Status.Ready:
    case Status.Paused
        pause();
        break;
    }
}
```

Tương đương:

```
switch (status) {
    case Status.Ready:
        pause();
        break;
    case Status.Paused
        pause();
        break;
    }
}
```

# do/while

- For:
  - for(var i = 0; i <= 10; ++i)
    - print("Number $i");
- While:
  - var i = 0;
  - while (i <= 10) {
    - print("Number $i");
    - ++i;
  - }
- Do/while:
  - var i = 0;
  - do {
    - print("Number $i");
    - ++i;
  - } while (i <= 10)

# Break và continue

Break: ngay lập tức thoát khỏi cấu trúc nếu được gọi

Continue: bỏ vòng lặp hiện hành khi được gọi, tiếp tục vòng lặp kế tiếp

```
for (var i = 0; i <= 3; ++i) {     // 1.
    for(var j = 0; j <= 5; ++j) { // 2.
        if (j == 5)
            break;
    }
}
```

# Cấu trúc lặp for in

- Khi muốn duyệt qua toàn bộ mảng hoặc không quan tâm đến index

```
final List<String> friendsList = ["A", "B", "C", "D", "E"];

for(var i = 0; i < friendsList.length; ++i)
    print(friendsList[i]);
```

- T … rong fr

```
List<String> friendsList = ["A", "B", "C", "D", "E"];

for(final friend in friendsList)
    print(friend);
```

# Assertions

Trong khi viết mã, bạn có thể sử dụng các xác nhận (assert) để đưa ra một ngoại lệ 1 nếu điều kiện đã cho là sai. Ví dụ

```
// the method returns a json-encoded string
final json = getJSON();

// if length > 0 is false --> runtime exception
assert(json.length > 0, "String cannot be empty");

// other actions
doParse(json);
```

assert có 2 tham số, tham số thứ nhất: điều kiện, tham số thứ 2 thông báo khi điều kiện sai
assert không hoạt động ở mode release, chỉ hoạt động ở mode debug

# Hàm

# Khái niệm

- Hàm là một khối lệnh thực hiện một tác vụ gì đó, khối lệnh này được dùng nhiều lần nên gom chúng tại thành một hàm.
- Trong Dart mọi thứ đều là đối tượng nên hàm cũng là một đối tượng (kế thừa Function).
- Khai báo hàm
  - double tinhtong(var a, double b, double c) {
  - return a + b + c;
  - }
- Goij hàm
  - var x = tinhtong(1,2,3);
  - print(x); //6.0

# Khái niệm

- Khai báo hàm chỉ ra kiểu giá trị trả về của hàm (ví dụ double), giá trị trả về bằng biểu thức của lệnh return.
- Trong ngôn ngữ Dart, việc thiếu khai báo kiểu giá trị trả về thì hàm vẫn hoạt động. Tuy nhiên, khi có chỉ rõ kiểu trả về thì giá trị trong biểu thức return phải trùng với kiểu khai báo hàm.
- Tham số được liệt kê sau tên hàm trong cặp ngoặc đơn () như trên có ba tham số a,b,c
- Hàm kết thúc khi chạy hết khối lệnh hoặc gặp lệnh return, giá trị hàm là biểu thức sau return, nếu thiếu giá trị hàm sẽ bằng null
- Khi gọi hàm thì viết tên hàm và truyền đúng tham số theo thứ tự yêu khai báo

# Tham số mặc định

- Tham số mặc định được định nghĩa với dấu "=" hoặc dấu ":". Truyền tham số theo cú pháp: tên_tham_số:giá_trị
  - double tinhtong(var a, {double b:1, double c:2}) {
  - return a + b + c;
  - }
  - var v1 = tinhtong(1);
  - print(v1); //4.0
  - var v2 = tinhtong(1, c:10);
  - print(v2); //12.0
  - var v3 = tinhtong(1, c:2, b:10);
  - print(v3); //13.0

# Tham số mặc định, tên tham số

Tên tham số được chỉ định bằng tên_tham_số:giá_trị, có thể truyền tham số không theo thứ tự

**Declaration**

```
void test({int? a, int? b}) {
    print("$a");
    print("$b");
}
```

**Calling**

```
void main() {
    // Prints '2' and '-6'
    test(a: 2, b: -6);
}
```

Kết quả vẫn không thay đổi nếu gọi:

```
test(b: -6, a: 2);
```

# Tham số mặc định, tên tham số

Ví dụ

**Declaration**

```
void test({int? a, int? b}) {
    print("$a");
    print("$b");
}
```

**Calling**

```
void main() {
    // Prins '2' and 'null'
    test(a: 2);
}
```

**Declaration**

```
void test({int? a, int b = 0}) {
    print("$a");
    print("$b");
}
```

**Calling**

```
void main() {
    // Prints '2' and '0'
    test(a: 2);
}
```

# Tham số bắt buộc

```
void test({int a = 0, required int b}) {
    print("$a");
    print("$b");

        }

        void main() {
            test(a: 5, b: 3); // Ok
            test(a: 5);       // Compilation error, 'b' is required
        }
```

# Tham số tuỳ chọn

- Các tham số tùy chọn của hàm cho phép khi gọi hàm có thể sử dụng hoặc không sử dụng các tham số này.
- Các tham số tùy chọn gom lại trong [] hoặc {}, khi gọi hàm nêu không có tham số này thì nó nhận giá trị null.

```
double tinhtong(var a, [double b, double c]) {
    var tong = a;
    if (b != null) tong += a;
    tong += (c!=null) ? c: 0;
}
print(tinhtong(1)); //1.0
print(tinhtong(1,2)); //3.0;
print(tinhtong(1,2,3)); //6.0;
```

# Tham số tuỳ chọn

Tham số tuỳ chọn phải được đặt cuối danh sách các tham số

```dart
// it compiles
void test(int a, {int? b}) { }

// it doesn't compile
void test({int? a}, int b) { }
```

# Hàm mũi tên =>

Với những hàm chỉ có một biểu thức trả về thì có thể viết ngắn gọn bằng ký hiệu mũi tên

```
bool checkEven(int value) {
    return value % 2 == 0
}

// Arrow syntax
bool checkEven(int value) => value % 2 == 0;

// Arrow syntax with method calls
bool checkEven(int value) => someOtherFunction(value);

// Does NOT work
bool checkEven(int value) => if (value % 2 == 0) ... ;
```

# Hàm mũi tên

```dart
double tinhtong(var a, var b) {
  return a + b;
}

// Có thể viết lại thành

double tinhtong(var a, var b) => a + b;
```

Hàm không có giá trị trả về vẫn có thể dùng hàm mũi tên, trình biên dịch sẽ tự sinh mã return dynamic cuối thân hàm

```dart
void test() => print("Alberto");
```

# Kiểu dữ liệu Function

Dart là ngôn ngữ lập trình hướng đối tượng, vì vậy, hàm cũng là 1 đối tượng kiểu Function

Ví dụ:

- Kiểu dữ liệu: bool Function(int)
- Tên biến: checker
- Giá trị: checkEven

```dart
// Khai báo hàm
bool checkEven(int value) => value % 2 == 0;

Run | Debug
void main() {
  // Gán một hàm cho một biến
  bool Function(int) checker = checkEven;
  // Sử dụng biến
  print(checker(8)); // true
}
```

# Kiểu dữ liệu Function

- Viết tắt
- Ví dụ:
  - Cả var và final đều có kiểu hàm bool Function(int)

```dart
bool checkEven(int value) => value % 2 == 0;

Run | Debug
void main() {
  final checker1 = checkEven;
  var checker2 = checkEven;
  print(checker1(8)); // true
  print(checker2(8)); // true
}
```

- Kiểu Fur

# Hàm ẩn - lambda - closure

Hàm ẩn: Anonymous function

Cú pháp: Kiểu_dữ_liệu tên_biến = (tham_số) =>{//...};

```dart
void main() {
  bool Function(int) isEven = (int value) => value % 2 == 0;
  print(isEven(19)); //false
}
```

```dart
//Single line
final anon_sl = () => 5.8 + 12;

//Multiple lines
final anon_ml = (String nickname) {
  var myName = 'My name is ';
  myName += nickname;
  return myName;
};

print(anon_ml('Beto'));
print(anon_sl());
```

# Hàm ẩn - lambda - closure

Trong nhiều ngữ cảnh khai báo hàm không dùng đến tên hàm, hàm đó gọi là hàm ẩn hoặc lambda hoặc closure

Tạo ra hàm ẩn thực hiện như hàm có tên bình thường, chỉ có phần kiểu trả về và tên bị thiếu

```
(var a, var b) {
  return a + b;
};

//Có thể dùng ký hiệu mũi tên  () => {}
(var a, var b) => {a + b};

//Nếu chỉ có 1 biểu thức trả về như trên có thể viết gọn hơn
(var a, var b) =>  a + b;
```

# Hàm ẩn - lambda - closure

- Hàm ẩn thường vừa khai báo vừa gọi hàm:

```
var x = (var a, var b) {
    return a + b;
}(5,6);
print(x); //11
```

- Hoặc gán hàm ẩn danh vào biến rồi gọi

```
var ham = (var a, var b) {
      return a + b;
};
print(ham(10,11)); //21
```

# Hàm ẩn - lambda - closure

Hàm ẩn danh rất tiện dụng để làm tham số (callback) trong các hàm khác, thậm chí khai báo luôn một ẩn danh ở tham số hàm

```
f1(var a, var b, var printmessage) {
    var c = a + b;
    printmessage(c);
}
f1(1, 2, (x) { print('Tổng là: $x');}); //Tổng là: 3
f1(1, 2, (z) => print('SUM = $z')); //SUM = 3
```

# Hàm callback

Tham số action gọi là callback vì nó thực hiện một hoạt động từ bên ngoài

```dart
// Hàm test không có giá trị trả về, có tham số action
// Tham số action kiểu Function
// - là hàm không có giá trị trả về có tham số kiểu int
void test(void Function(int) action) {
  final list = [1, 2, 3, 4, 5];
  // vòng lặp duyệt toàn bộ list, mỗi item, gọi hàm
  for (final item in list) action(item);
}

Run | Debug
void main() {
  // gọi hàm test
  test(
      // hàm ẩn, không có giá trị trả về, tham số kiểu int
      (int value) {
    print("Number $value");
  });
  test((int value) => print("$value"));
  test((int value) => print("${value + 2}"));
}
```

# Hàm callback

Trong các list và map, ta thường dùng forEach(). forEach() gọi callback thực thi trong khi duyệt các phần tử

```
void main() {
    // Declare the list
    final list = [1, 2, 3, 4, 5];
    // Iterate
    list.forEach((int x) => print("Number $x"));
}
```

# functions

Dart cho phép khai báo các hàm bên trong các hàm khác

Các hàm lồng nhau chỉ có thể được gọi bên trong hàm chứa chúng

Outer function

Inner function

```dart
void testInner(int value) {
    // Nested function
    int randomValue() => Random().nextInt(10);

    // Using the nested funcion
    final number = value + randomValue();
    print("$number");
}
```

# Good and Bad

```dart
// Good
void test([int a = 0]) {}

// Bad
void test([int a : 0]) {}
```

```dart
// Good
void test({int? a}) {}

// Bad
void test({int? a = null}) {}
```

# Good and Bad

```
void showNumber(int value) {
    print("$value");
}

void main() {
    // List of values
    final numbers = [2, 4, 6, 8, 10];

    // Good
    numbers.forEach(showNumber);

    // Bad
    numbers.forEach((int val) { showNumber(val); });
}
```