Dart Starter Guide

Written by Thapanapong Rukkanchanunt

Variables and String Intepolation

- Convert variable into a string with ${var_name}
  - ‘Hello ${val}’ puts the value of val into a string literal
- You can even do arithmatic inside!
  - ‘${3+2}’ will output 5 as a string
- You can use expression.
  - ‘${"word".toUpperCase()}’ will output ‘WORD’
- For Object type, the output will be toString() function.
  - ‘$myObject’ is the same as myObject.toString()
- You can omit the curly bracket if it’s just a variable. This is preferred.

NULLable Variables

Typically, value of a variable can’t be null

- int a = null; // INVALID in null-safe Dart.
- int a; // still ok but need assignment before use

With the help of ?, variable can be null

- int? a = null; // Valid in null-safe Dart.
- int? a; // The initial value of a is null.

Null-aware Operators

Assign if null

- b ??= val;
- a = value ?? 0; // if a is null, a = value. Otherwise, a = 0

Old conditional assignment still applies

- a == null ? null : a.b // equivalent to if a is null, return null. Otherwise return a.b

Conditional property access (will not execute if previous is null)

- myObject?.someProperty // someProperty will not be called if myObject is null
- myObject?.someProperty?.someMethod() // someMethod() will not be called if myObject.some Property is null.

Collections

Create collection with initialization (final = no reassign)

- final aListOfStrings = ['one', 'two', 'three’];
- final aSetOfStrings = {'one', 'two', 'three’};
- final aMapOfStringsToInts = { 'one': 1, 'two': 2, 'three': 3, };

Can also specify type to the collection

- final aListOfInts = <int>[];
- final aSetOfInts = <int>{};
- final aMapOfIntToDouble = <int, double>{};

Arrow Syntax

- A new way of defining a one-line return function
- Arrow symbol will execute expression on the right and return its value

```
bool hasEmpty = aListOfStrings.any((s) {
  return s.isEmpty;
});
```

```
bool hasEmpty = aListOfStrings.any((s) => s.isEmpty);
```

Cascades

- If you want to perform multiple operations on the same object, consider using cascade.

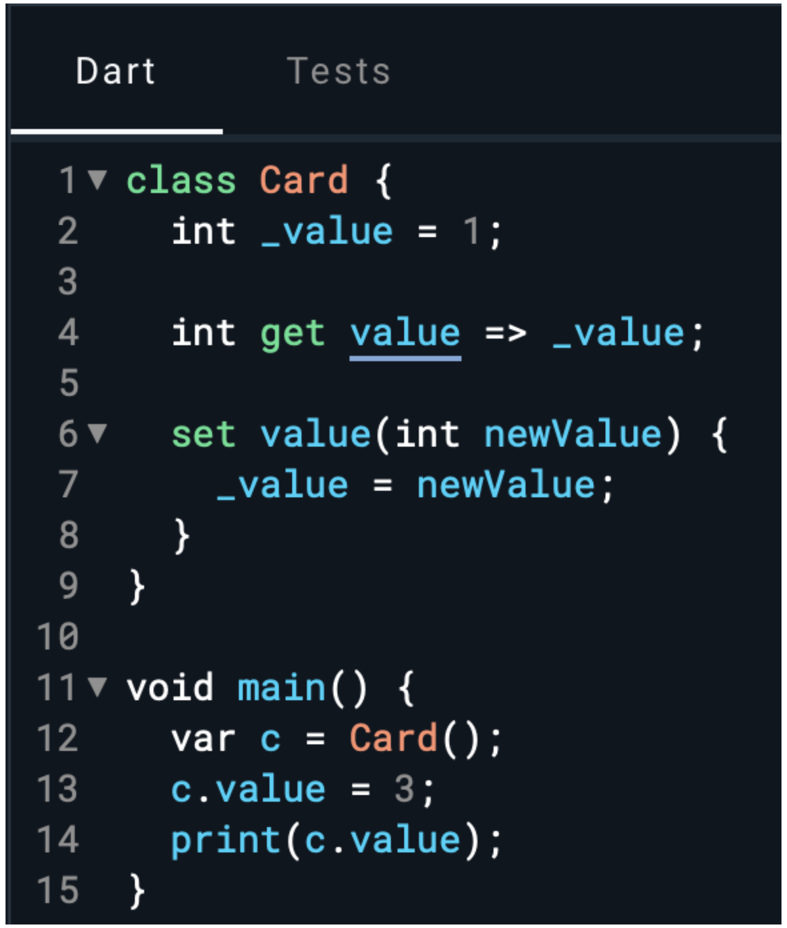
```
var button = querySelector('#confirm');
button?.text = 'Confirm';
button?.classes.add('important');
button?.onClick.listen((e) => window.alert('Confirmed!'));
button?.scrollIntoView();
```

```
querySelector('#confirm')
  ?..text = 'Confirm'
  ..classes.add('important')
  ..onClick.listen((e) => window.alert('Confirmed!'))
  ..scrollIntoView();
```

Classes, Setters, Getters

- Use class keyword to define Class
- Use get keyword to define getter variable
- Use set keyword to define setter variable
    - Getter and setter are essential in document generation. Getter should come before Setter.
    - More on format/convention later in the course.

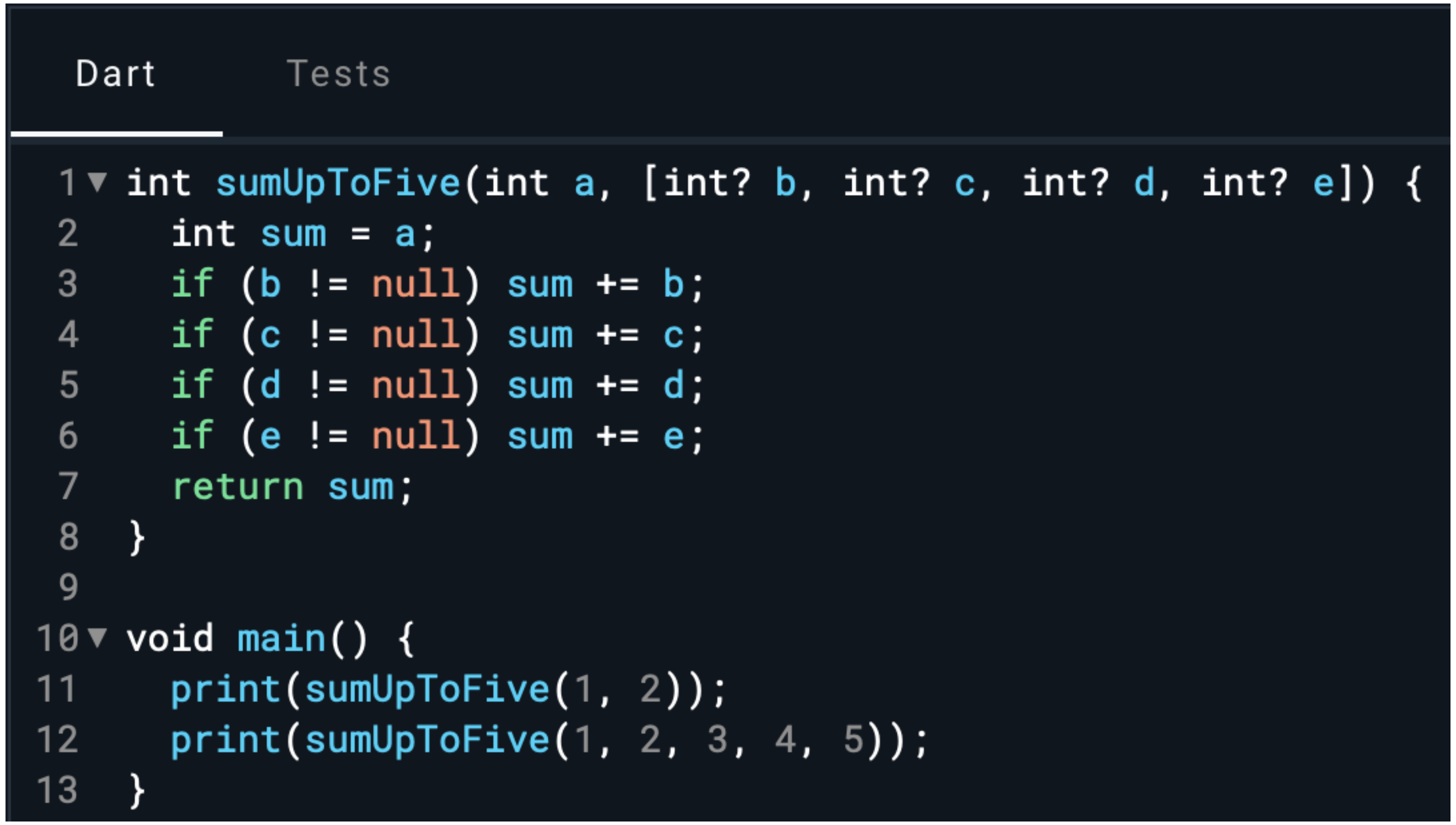
- Naming convention is camelCase



```dart
class Card {
  int _value = 1;

  int get value => _value;

  set value(int newValue) {
    _value = newValue;
  }
}

void main() {
  var c = Card();
  c.value = 3;
  print(c.value);
}
```

Optional Positional Parameters

- Use squared bracket for optional parameters
- Optional parameters must be nullable.



```dart
int sumUpToFive(int a, [int? b, int? c, int? d, int? e]) {
  int sum = a;
  if (b != null) sum += b;
  if (c != null) sum += c;
  if (d != null) sum += d;
  if (e != null) sum += e;
  return sum;
}

void main() {
  print(sumUpToFive(1, 2));
  print(sumUpToFive(1, 2, 3, 4, 5));
}
```

Named Parameters

- You can name parameters using curly brace.
- These parameters are optional without required keyword.
- If named parameter is non-nullable, you must provide default value or make it required.

```
void printName(String firstName, String lastName, {String? middleName}) {
  print('$firstName ${middleName ?? ''} $lastName');
}
```

```
void printName(String firstName, String lastName, {String middleName = ''}) {
  print('$firstName $middleName $lastName');
}
```

```
printName('John', 'Smith', middleName: 'Who');
```

Exceptions

- Dart throws exception with throw keyword and catch exception using try, on, catch keywords.

```
throw Exception('Something bad happened.');
throw 'Waaaaaaah!';
```

```
try {
  breedMoreLlamas();
} on OutOfLlamasException {      // A specific exception
  buyMoreLlamas();
} on Exception catch (e) {       // Anything else that is an exception
  print('Unknown exception: $e’);
} catch (e) {                    // No specified type, handles all
  print('Something really unknown: $e’);
}
```

this Constructor

- Short constructor can assign property with this keyword.

```
class MyColor {
  int red;
  int green;
  int blue;

  MyColor(this.red, this.green, this.blue);
}

final color = MyColor(80, 80, 128);
```

```
MyColor({required this.red, required this.green, required this.blue});
```

```
MyColor([this.red = 0, this.green = 0, this.blue = 0]);
```

```
MyColor({this.red = 0, this.green = 0, this.blue = 0});
```

Initializer List

- Sometimes we want to do something before constructor body begins execution.
- The example below defines fromJson constructor that immediately reads x, y values from the map.

```
Point.fromJson(Map<String, double> json)
    : x = json['x']!,
      y = json['y']! {
  print('In Point.fromJson(): ($x, $y)');
}
```

Named and Factory Constructors

- Named constructors are other ways of creating an instance.
- Factory constructors can return subtypes or null.

```
class Point {
  double x, y;

  Point(this.x, this.y);

  Point.origin()
      : x = 0,
        y = 0;
}
```

```
class Square extends Shape {}
class Circle extends Shape {}

class Shape {
  Shape();

  factory Shape.fromTypeName(String typeName) {
    if (typeName == 'square') return Square();
    if (typeName == 'circle') return Circle();
    throw ArgumentError('Unrecognized $typeName');
  }
}
```

Redirecting Constructors

- Sometimes a constructor’s only purpose is to redirect to another constructor in the same class. A redirecting constructor’s body is empty, with the constructor call appearing after a colon (:).

```
class Automobile {
  String make;
  String model;
  int mpg;
  // The main constructor for this class.
  Automobile(this.make, this.model, this.mpg);
  // Delegates to the main constructor.
  Automobile.hybrid(String make, String model) : this(make, model, 60);
  // Delegates to a named constructor
  Automobile.fancyHybrid() : this.hybrid('Futurecar', 'Mark 2’);
}
```

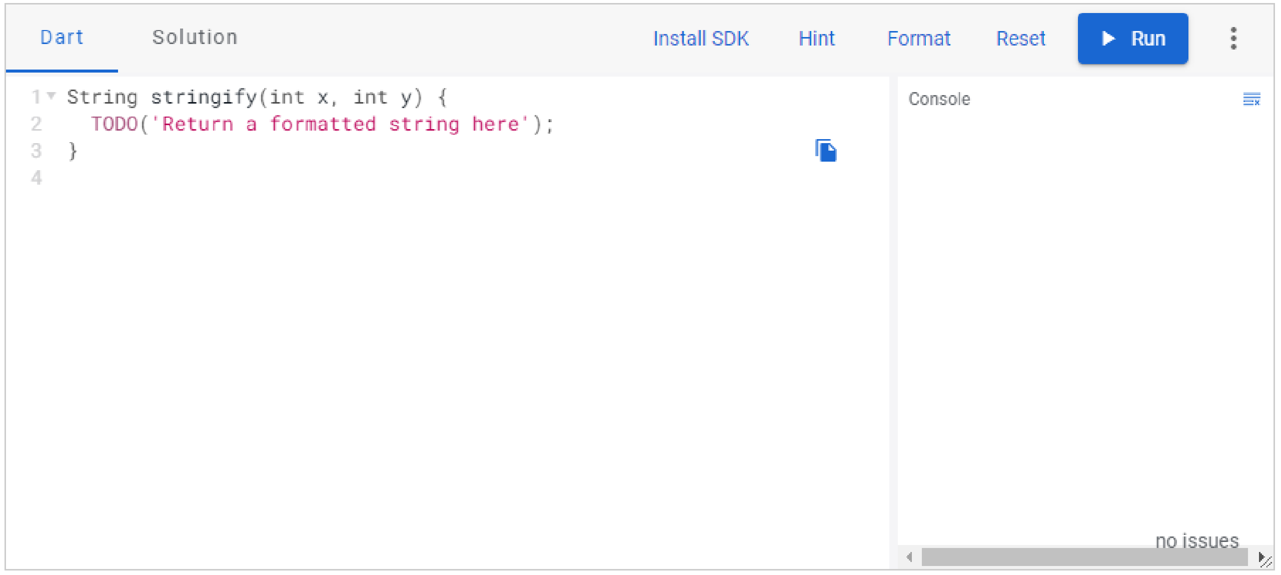
const Constructors

- Just like constant variable, all instance variables of objects created from const constructors are immutable.

```
class ImmutablePoint {
  static const ImmutablePoint origin = ImmutablePoint(0, 0);

  final int x;
  final int y;

  const ImmutablePoint(this.x, this.y);
}
```

Dart cheatsheet codelab

- We will complete tasks in the cheatsheet codelab.
  - https://dart.dev/codelabs/dart-cheatsheet
  - You pass each task when you see "All tests passed!" after clicking "Run."
- You can look at the solutions, but you will learn nothing from it.
- Complete tasks from "String interpolation" to "Const constructors"
- Inform staff if you have finished.



```
String stringify(int x, int y) {
  TODO('Return a formatted string here');
}
```

Resources

- Dart syntax for basic programming
  - https://dart.dev/guides/language/language-tour
- Online interactive lesson on Dart programming
  - https://dart.dev/codelabs/dart-cheatsheet