The Dart Project

Damien Martin-Guillerez

Mardi du dev', Bordeaux, February 2012

JavaScript...

... is flawless[1]:

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0           [] + {}
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0           [] + {}        > [object Object]
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0           [] + {}        > [object Object]
{} + [1, 4]
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0           [] + {}        > [object Object]
{} + [1, 4]   > NaN
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0           [] + {}        > [object Object]
{} + [1, 4]   > NaN         [1, 4] + {}
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0           [] + {}        > [object Object]
{} + [1, 4]   > NaN         [1, 4] + {}    > 1,4[object Object]
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0           [] + {}        > [object Object]
{} + [1, 4]   > NaN         [1, 4] + {}    > 1,4[object Object]
```

- ... and a beautiful syntax

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really clear...

```
"abc"  > abc      1  > 1     "abc" + 1  > abc1
```

- ... with great operator behavior ...

```
{} + 45       > 45          45 + {}        > 45[object Object]
{} + []       > 0           [] + {}        > [object Object]
{} + [1, 4]   > NaN         [1, 4] + {}    > 1,4[object Object]
```

- ... and a beautiful syntax

```
function aClass() {
  this.attribute1;  this.attribute2;
  this.methodA = function() { /* code */ }
}
var aInstance = new aClass();
```

[1] https://www.destroyallsoftware.com/talks/wat

JavaScript...

... is flawless[1]:

- Typing is really

```
"abc"  > abc                                    bc1
```

- ... with great op

```
{} + 45       >                                 45[object Object]
{} + []       >                                 [object Object]
{} + [1, 4]   >                                 1,4[object Object]
```

- ... and a beauti

```
function aClas
  this.attribu
  this.methodA                                  }
}
var aInstance = new aClass();
```

[1] https://www.destroyallsoftware.com/talks/wat

Dart

Project started at Google in September 2011 and they want to:

Dart

Project started at Google in September 2011 and they want to:

- Fix JavaScript...

Dart

Project started at Google in September 2011 and they want to:

- Fix JavaScript...
- ... and the DOM ...

Dart

Project started at Google in September 2011 and they want to:

- Fix JavaScript...
- ... and the DOM ...
- ... and the whole web app development.

1

Introduction to the language

Hello World

```
// Define a function
hello(string subject) {
  print('Hello, $subject!');
}

/* Entry point */
main() {
  hello("World");
  hello('Dart');
}
```

Hello World

```
// Define a function
hello(string subject) {
  print('Hello, $subject!');
}

/* Entry point */
main() {
  hello("World");
  hello('Dart');
}
```

Language concepts

- Standard syntax (from JavaScript and Java)

Language concepts

- Standard syntax (from JavaScript and Java)
- Object-oriented so everything is an object

Language concepts

- Standard syntax (from JavaScript and Java)
- Object-oriented so everything is an object
- Typed but with the default type `Dynamic`

Language concepts

- Standard syntax (from JavaScript and Java)
- Object-oriented so everything is an object
- Typed but with the default type `Dynamic`
- Support of top-level functions, object methods and anonymous lambdas

Language concepts

- Standard syntax (from JavaScript and Java)
- Object-oriented so everything is an object
- Typed but with the default type `Dynamic`
- Support of top-level functions, object methods and anonymous lambdas
- Generics, Typedefs, Exceptions, Libraries, ...

Types

- `String`: `"abcdef"`, `'abcdef'`

Types

- `String`: `"abcdef"`, `'abcdef'`
- `num`: `1` (`int`)

Types

- `String`: `"abcdef"`, `'abcdef'`
- `num`: `1` (`int`), `1.0` (`float`)

Types

- `String`: `"abcdef"`, `'abcdef'`
- `num`: `1` (`int`), `1.0` (`float`)
- `boolean`: `true`, `false`

Types

- `String`: `"abcdef"`, `'abcdef'`
- `num`: `1` (`int`), `1.0` (`float`)
- `boolean`: `true`, `false`
- `List`: `[1, 2, 3]`, `[]`

Types

- String: `"abcdef"`, `'abcdef'`
- num: `1` (`int`), `1.0` (`float`)
- boolean: `true`, `false`
- List: `[1, 2, 3]`, `[]`
- Map: `var map = {'a': 1, 'b': 2}`, `map['a'] = 2`, ...

Types

- String: `"abcdef"`, `'abcdef'`
- num: `1` (`int`), `1.0` (`float`)
- boolean: `true`, `false`
- List: `[1, 2, 3]`, `[]`
- Map: `var map = {'a': 1, 'b': 2}`, `map['a'] = 2`, ...
- Dynamic: `var a = 1; a = "b";`

Types

- String: `"abcdef"`, `'abcdef'`
- num: `1` (`int`), `1.0` (`float`)
- boolean: `true`, `false`
- List: `[1, 2, 3]`, `[]`
- Map: `var map = {'a': 1, 'b': 2}`, `map['a'] = 2`, ...
- Dynamic: `var a = 1; a = "b";`
- Function...

Functions

```
hello(subject) {
  print('Hello, $subject!');
}
// hello("World") prints "Hello, World!"
```

Functions

```
hello(subject) {
  print('Hello, $subject!');
}
// hello("World") prints "Hello, World!"

var h = (subject) { print('Hello, $subject!'); }
// h("Dart") prints "Hello, Dart!"
```

Functions

```
hello(subject) {
  print('Hello, $subject!');
}
// hello("World") prints "Hello, World!"

var h = (subject) { print('Hello, $subject!'); }
// h("Dart") prints "Hello, Dart!"

int fibonacci(int n) {
  if(n == 0) {
    return 0;
  } else if(n == 1) {
    return 1;
  } else {
    return fibonacci(n-2) + fibonacci(n-1);
  }
}
// fibonacci(10) returns 55
```

Object-oriented programming

```
class Point {
  num _x; // protected
  num y;  // public




}
```

Object-oriented programming

```
class Point {
  num _x; // protected
  num y;  // public

  Point() { this._x = 0; this.y = 0; }


}
```

Object-oriented programming

```
class Point {
  num _x; // protected
  num y;  // public

  Point() { this._x = 0; this.y = 0; }


}

Point p = new Point();
p.y = 1; // now p is <0, 1>
// p._x = 1; // not possible
```

Object-oriented programming

```
class Point {
  num _x; // protected
  num y;  // public

  Point() { this._x = 0; this.y = 0; }
  Point.from(this._x, this.y);

}

Point p = new Point();
p.y = 1; // now p is <0, 1>
// p._x = 1; // not possible
```

Object-oriented programming

```
class Point {
  num _x; // protected
  num y;  // public

  Point() { this._x = 0; this.y = 0; }
  Point.from(this._x, this.y);

}

Point p = new Point();
p.y = 1; // now p is <0, 1>
// p._x = 1; // not possible
p = new Point.from(1,2); // now p is <1, 2>
```

Object-oriented programming

```
class Point {
  num _x; // protected
  num y;  // public

  Point() { this._x = 0; this.y = 0; }
  Point.from(this._x, this.y);
  void setX(num x) { this._x = x; }
}

Point p = new Point();
p.y = 1; // now p is <0, 1>
// p._x = 1; // not possible
p = new Point.from(1,2); // now p is <1, 2>
```

Object-oriented programming

```
class Point {
  num _x; // protected
  num y;  // public

  Point() { this._x = 0; this.y = 0; }
  Point.from(this._x, this.y);
  void setX(num x) { this._x = x; }
}

Point p = new Point();
p.y = 1; // now p is <0, 1>
// p._x = 1; // not possible
p = new Point.from(1,2); // now p is <1, 2>
p.setX(3); // now p is <3, 2>
```

Object-oriented programming – extension and abstraction

```
abstract class Point {
    int getX();
    int getY();
}
```

Object-oriented programming – extension and abstraction

```
abstract class Point {
    int getX();
    int getY();
}

class MyPoint extends Point {
    int x; int y;
    MyPoint(this.x, this.y);
    int getX() => this.x;
    int getY() => this.y;
}
```

Generics

```
abstract class Point<T> {
    T getX();
    T getY();
}
```

Generics

```
abstract class Point<T> {
   T getX();
   T getY();
}

class MyPoint extends Point<int> {
   int x; int y;
   MyPoint(this.x, this.y);
   int getX() => this.x;
   int getY() => this.y;
}
```

Generics

```
abstract class Point<T> {
   T getX();
   T getY();
}

class MyPoint extends Point<int> {
   int x; int y;
   MyPoint(this.x, this.y);
   int getX() => this.x;
   int getY() => this.y;
}
```

And so many more: `static`, `const`, `final`, ...

Isolates

Dart provides Isolates as a mean to do concurrency:

Isolates

Dart provides Isolates as a mean to do concurrency:

- Isolates run in separate threads

Isolates

Dart provides Isolates as a mean to do concurrency:

- Isolates run in separate threads
- In a isolate, only one callback can run at the same time (no non-explicit concurrency)

Isolates

Dart provides Isolates as a mean to do concurrency:

- Isolates run in separate threads
- In a isolate, only one callback can run at the same time (no non-explicit concurrency)
- Isolates don’t share memory

Isolates

Dart provides Isolates as a mean to do concurrency:

- Isolates run in separate threads
- In a isolate, only one callback can run at the same time (no non-explicit concurrency)
- Isolates don’t share memory
- Isolates can communicate using a dedicate message system

2

Cool features

String interpolation

```
num a = 42;
```

String interpolation

```
num a = 42;
print(a);
```

String interpolation

```
num a = 42;
print(a);                                   // 42
```

String interpolation

```
num a = 42;
print(a);                                   // 42
print("a = $a");
```

String interpolation

```
num a = 42;
print(a);                          // 42
print("a = $a");                   // a = 42
```

String interpolation

```
num a = 42;
print(a);                                  // 42
print("a = $a");                           // a = 42
print("a + 1 = ${a+1}");
```

String interpolation

```
num a = 42;
print(a);                          // 42
print("a = $a");                   // a = 42
print("a + 1 = ${a+1}");           // a + 1 = 43
```

String interpolation

```
num a = 42;
print(a);                                  // 42
print("a = $a");                           // a = 42
print("a + 1 = ${a+1}");                   // a + 1 = 43

Map map = {'1': 2, '2': 3};
```

String interpolation

```
num a = 42;
print(a);                                  // 42
print("a = $a");                           // a = 42
print("a + 1 = ${a+1}");                   // a + 1 = 43

Map map = {'1': 2, '2': 3};
print("map = $map");
```

String interpolation

```
num a = 42;
print(a);                                  // 42
print("a = $a");                           // a = 42
print("a + 1 = ${a+1}");                   // a + 1 = 43

Map map = {'1': 2, '2': 3};
print("map = $map");                       // map = {1: 2, 2: 3}
```

String interpolation

```
num a = 42;
print(a);                                   // 42
print("a = $a");                            // a = 42
print("a + 1 = ${a+1}");                    // a + 1 = 43

Map map = {'1': 2, '2': 3};
print("map = $map");                        // map = {1: 2, 2: 3}
print("map[1] = ${map['1']}");
```

String interpolation

```
num a = 42;
print(a);                                  // 42
print("a = $a");                           // a = 42
print("a + 1 = ${a+1}");                   // a + 1 = 43

Map map = {'1': 2, '2': 3};
print("map = $map");                       // map = {1: 2, 2: 3}
print("map[1] = ${map['1']}");             // map[1] = 2
```

String interpolation

```
num a = 42;
print(a);                                  // 42
print("a = $a");                           // a = 42
print("a + 1 = ${a+1}");                   // a + 1 = 43

Map map = {'1': 2, '2': 3};
print("map = $map");                       // map = {1: 2, 2: 3}
print("map[1] = ${map['1']}");             // map[1] = 2
print("map[2]+1 = ${map['2']+1}");
```

String interpolation

```
num a = 42;
print(a);                                  // 42
print("a = $a");                           // a = 42
print("a + 1 = ${a+1}");                   // a + 1 = 43

Map map = {'1': 2, '2': 3};
print("map = $map");                       // map = {1: 2, 2: 3}
print("map[1] = ${map['1']}");             // map[1] = 2
print("map[2]+1 = ${map['2']+1}");         // map[2]+1 = 4
```

1 line lambdas

1-line functions:

1 line lambdas

1-line functions:

```
fib(n) => (n == 0) ? 0 : ((n == 1) ? 1 : (fib(n-2) + fib(n-1)));
// fib(10) returns 55
```

1 line lambdas

1-line functions:

```
fib(n) => (n == 0) ? 0 : ((n == 1) ? 1 : (fib(n-2) + fib(n-1)));
// fib(10) returns 55
```

So 1-line lambdas:

1 line lambdas

1-line functions:

```
fib(n) => (n == 0) ? 0 : ((n == 1) ? 1 : (fib(n-2) + fib(n-1)));
// fib(10) returns 55
```

So 1-line lambdas:

```
((x) => x+1)(2);
var f = (a, b) => a + b;
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
h(1) = 0
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
h(1) = 0, h(1, 2, true) = 3
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
h(1) = 0, h(1, 2, true) = 3

i(x, {y: 0, mul: false}) => mul ? x*y : x + y;
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
h(1) = 0, h(1, 2, true) = 3

i(x, {y: 0, mul: false}) => mul ? x*y : x + y;
i(1) = 1
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
h(1) = 0, h(1, 2, true) = 3

i(x, {y: 0, mul: false}) => mul ? x*y : x + y;
i(1) = 1, i(1, mul:true, y:2) = 2
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
h(1) = 0, h(1, 2, true) = 3

i(x, {y: 0, mul: false}) => mul ? x*y : x + y;
i(1) = 1, i(1, mul:true, y:2) = 2

j(x, [y = 0]) => ?y ? x*y : x;
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
h(1) = 0, h(1, 2, true) = 3

i(x, {y: 0, mul: false}) => mul ? x*y : x + y;
i(1) = 1, i(1, mul:true, y:2) = 2

j(x, [y = 0]) => ?y ? x*y : x;
j(1) = 1
```

Named and optional parameters

```
f(x, [y = 0]) => x + y;
f(1) = 1, f(1, 2) = 3

g(x, {int y: 0}) => x + y;
g(1) = 1, g(1, y:2) = 3

h(x, [y = 0, mul = false]) => mul ? x + y : x * y;
h(1) = 0, h(1, 2, true) = 3

i(x, {y: 0, mul: false}) => mul ? x*y : x + y;
i(1) = 1, i(1, mul:true, y:2) = 2

j(x, [y = 0]) => ?y ? x*y : x;
j(1) = 1, j(1, 0) = 0
```

Getters and setters

```
abstract class Point<T> {
    T get x;
    T get y;
    toString() => "<${this.x}, ${this.y}>";
}
```

Getters and setters

```
abstract class Point<T> {
   T get x;
   T get y;
   toString() => "<${this.x}, ${this.y}>";
}
class MyPoint extends Point<int> {
   int _x; int _y;
   MyPoint(this._x, this._y);
   int get x => this._x;
   int get y => this._y;

}
```

Getters and setters

```
abstract class Point<T> {
   T get x;
   T get y;
   toString() => "<${this.x}, ${this.y}>";
}
class MyPoint extends Point<int> {
   int _x; int _y;
   MyPoint(this._x, this._y);
   int get x => this._x;
   int get y => this._y;

}
Point<int> p = new MyPoint(0, 1);
// p = <0, 1>, p.x = 0, p.y = 1
```

Getters and setters

```
abstract class Point<T> {
   T get x;
   T get y;
   toString() => "<${this.x}, ${this.y}>";
}
class MyPoint extends Point<int> {
   int _x; int _y;
   MyPoint(this._x, this._y);
   int get x => this._x;
   int get y => this._y;
   set x(xval) { this._x = xval; }
}
Point<int> p = new MyPoint(0, 1);
// p = <0, 1>, p.x = 0, p.y = 1
```

Getters and setters

```
abstract class Point<T> {
   T get x;
   T get y;
   toString() => "<${this.x}, ${this.y}>";
}
class MyPoint extends Point<int> {
   int _x; int _y;
   MyPoint(this._x, this._y);
   int get x => this._x;
   int get y => this._y;
   set x(xval) { this._x = xval; }
}
Point<int> p = new MyPoint(0, 1);
// p = <0, 1>, p.x = 0, p.y = 1
p.x = 2;
// p = <2, 1>, p.x = 2, p.y = 1
```

Other syntactic sugars

- Cascade operator:
  `objectWithMoveNextMethod..moveNext()..moveNext()`

Other syntactic sugars

- Cascade operator:
  ```
  objectWithMoveNextMethod..moveNext()..moveNext()
  ```
- Multiline strings: ```'''this
  is a
  multiline string'''```

Other syntactic sugars

- Cascade operator:
  `objectWithMoveNextMethod..moveNext()..moveNext()`
- Multiline strings: `'''this`
  `is a`
  `multiline string'''`
- Collection iteration: `for(var el in [1, 2, 3]);`

Automatic interfaces

```
class IntValue {
   int _val;
   IntValue(this._val);
   int get value => this._val;
   toString() => "${this.value}";
}
```

Automatic interfaces

```
class IntValue {
   int _val;
   IntValue(this._val);
   int get value => this._val;
   toString() => "${this.value}";
}
class MutableIntValue implements IntValue {
   int _val;
   MutableIntValue(this._val);
   int get value => this._val;
   set value(val) { this._val = val; }
}
```

Automatic interfaces

```
class IntValue {
   int _val;
   IntValue(this._val);
   int get value => this._val;
   toString() => "${this.value}";
}
class MutableIntValue implements IntValue {
   int _val;
   MutableIntValue(this._val);
   int get value => this._val;
   set value(val) { this._val = val; }
}
IntValue val = new IntValue(0);
```

Automatic interfaces

```
class IntValue {
   int _val;
   IntValue(this._val);
   int get value => this._val;
   toString() => "${this.value}";
}
class MutableIntValue implements IntValue {
   int _val;
   MutableIntValue(this._val);
   int get value => this._val;
   set value(val) { this._val = val; }
}
IntValue val = new IntValue(0);
// val.value = 0, val = 0
```

Automatic interfaces

```
class IntValue {
   int _val;
   IntValue(this._val);
   int get value => this._val;
   toString() => "${this.value}";
}
class MutableIntValue implements IntValue {
   int _val;
   MutableIntValue(this._val);
   int get value => this._val;
   set value(val) { this._val = val; }
}
IntValue val = new IntValue(0);
// val.value = 0, val = 0
val = new MutableIntValue(2);
```

Automatic interfaces

```
class IntValue {
   int _val;
   IntValue(this._val);
   int get value => this._val;
   toString() => "${this.value}";
}
class MutableIntValue implements IntValue {
   int _val;
   MutableIntValue(this._val);
   int get value => this._val;
   set value(val) { this._val = val; }
}
IntValue val = new IntValue(0);
// val.value = 0, val = 0
val = new MutableIntValue(2);
// val.value = 2, val = Instance of 'MutableIntValue'
```

Future

```
Future billionCount() {
  var completer = new Completer();
  for(int i = 0; i < 1000000000; i++);
  completer.complete(true); // returns true
  return completer.future;
}
```

Future

```
Future billionCount() {
  var completer = new Completer();
  for(int i = 0; i < 1000000000; i++);
  completer.complete(true); // returns true
  return completer.future;
}
main() {
  var result = billionCount(); // Returns immediately.
  result.then((success) {
   print('Counted to a billion');
  });
}
```

3

Libraries

Libraries in the Dart framework

- The basic `dart:core`,

Libraries in the Dart framework

- The basic `dart:core`,
- `dart:math` for basic mathematics,

Libraries in the Dart framework

- The basic `dart:core`,
- `dart:math` for basic mathematics,
- `dart:html` for manipulating the DOM,

Libraries in the Dart framework

- The basic `dart:core`,
- `dart:math` for basic mathematics,
- `dart:html` for manipulating the DOM,
- `dart:isolate` for manipulating isolates,

Libraries in the Dart framework

- The basic `dart:core`,
- `dart:math` for basic mathematics,
- `dart:html` for manipulating the DOM,
- `dart:isolate` for manipulating isolates,
- `dart:io` for basic input/output,

Libraries in the Dart framework

- The basic `dart:core`,
- `dart:math` for basic mathematics,
- `dart:html` for manipulating the DOM,
- `dart:isolate` for manipulating isolates,
- `dart:io` for basic input/output,
- `dart:json` to speak JSON,

Libraries in the Dart framework

- The basic `dart:core`,
- `dart:math` for basic mathematics,
- `dart:html` for manipulating the DOM,
- `dart:isolate` for manipulating isolates,
- `dart:io` for basic input/output,
- `dart:json` to speak JSON,
- `dart:uri` for URI manipulation,

Libraries in the Dart framework

- The basic `dart:core`,
- `dart:math` for basic mathematics,
- `dart:html` for manipulating the DOM,
- `dart:isolate` for manipulating isolates,
- `dart:io` for basic input/output,
- `dart:json` to speak JSON,
- `dart:uri` for URI manipulation,
- `dart:utf` for UTF encoding,

Libraries in the Dart framework

- The basic `dart:core`,
- `dart:math` for basic mathematics,
- `dart:html` for manipulating the DOM,
- `dart:isolate` for manipulating isolates,
- `dart:io` for basic input/output,
- `dart:json` to speak JSON,
- `dart:uri` for URI manipulation,
- `dart:utf` for UTF encoding,
- and `dart:crypto` for cryptographic musts.

dart:html

```
import 'dart:html';
main() {
  // Find an element by id (an-id).
  Element elem1 = query('#an-id');
  // Find an element by class (a-class).
  Element elem2 = query('.a-class');
```

dart:html

```
import 'dart:html';
main() {
  // Find an element by id (an-id).
  Element elem1 = query('#an-id');
  // Find an element by class (a-class).
  Element elem2 = query('.a-class');

  // Find all elements by tag (<div>).
  List<Element> elems1 = queryAll('div');
  // Find all text inputs.
  List<Element> elems2 = queryAll('input[type="text"]');
```

dart:html

```
import 'dart:html';
main() {
  // Find an element by id (an-id).
  Element elem1 = query('#an-id');
  // Find an element by class (a-class).
  Element elem2 = query('.a-class');

  // Find all elements by tag (<div>).
  List<Element> elems1 = queryAll('div');
  // Find all text inputs.
  List<Element> elems2 = queryAll('input[type="text"]');

  query('#example').href = 'http://dartlang.org';
}
```

dart:io

Enables HTTP servers:

```
import 'dart:io';

main() {
  dartHandler(HttpRequest request, HttpResponse response) {
    print('New request');
    response.outputStream.writeString('Dart is optionally typed');
    response.outputStream.close();
  };

  var httpServer = new HttpServer();
  httpServer.addRequestHandler(
      (req) => req.path == '/languages/dart',
      dartHandler);

  httpServer.listen('127.0.0.1', 8888);
}
```

It was just a quick look

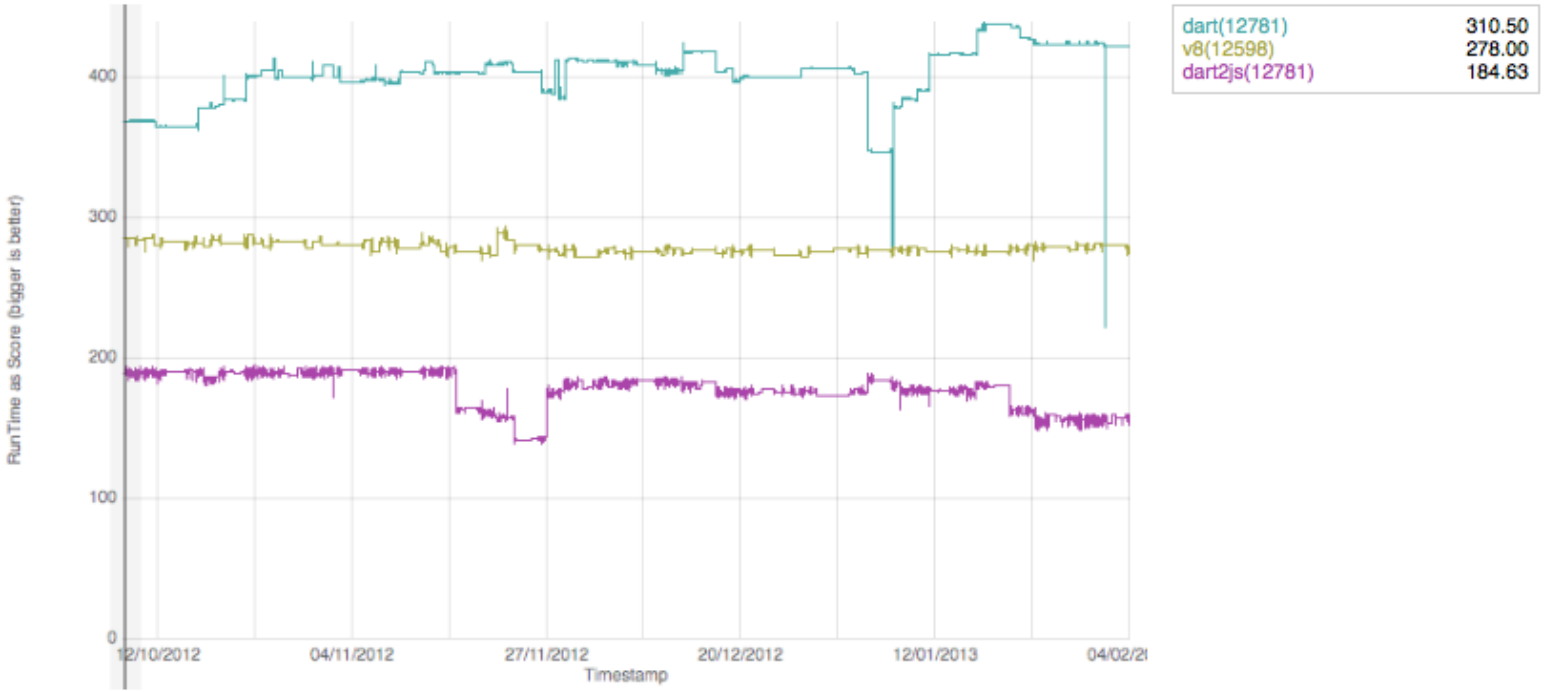
See more at `https://www.dartlang.org/docs/dart-up-and-running/contents/ch03.html`

4

Tools

Dart Virtual Machine & dart2js

Dart come along with a virtual machine and a compiler to JavaScript that are efficient:

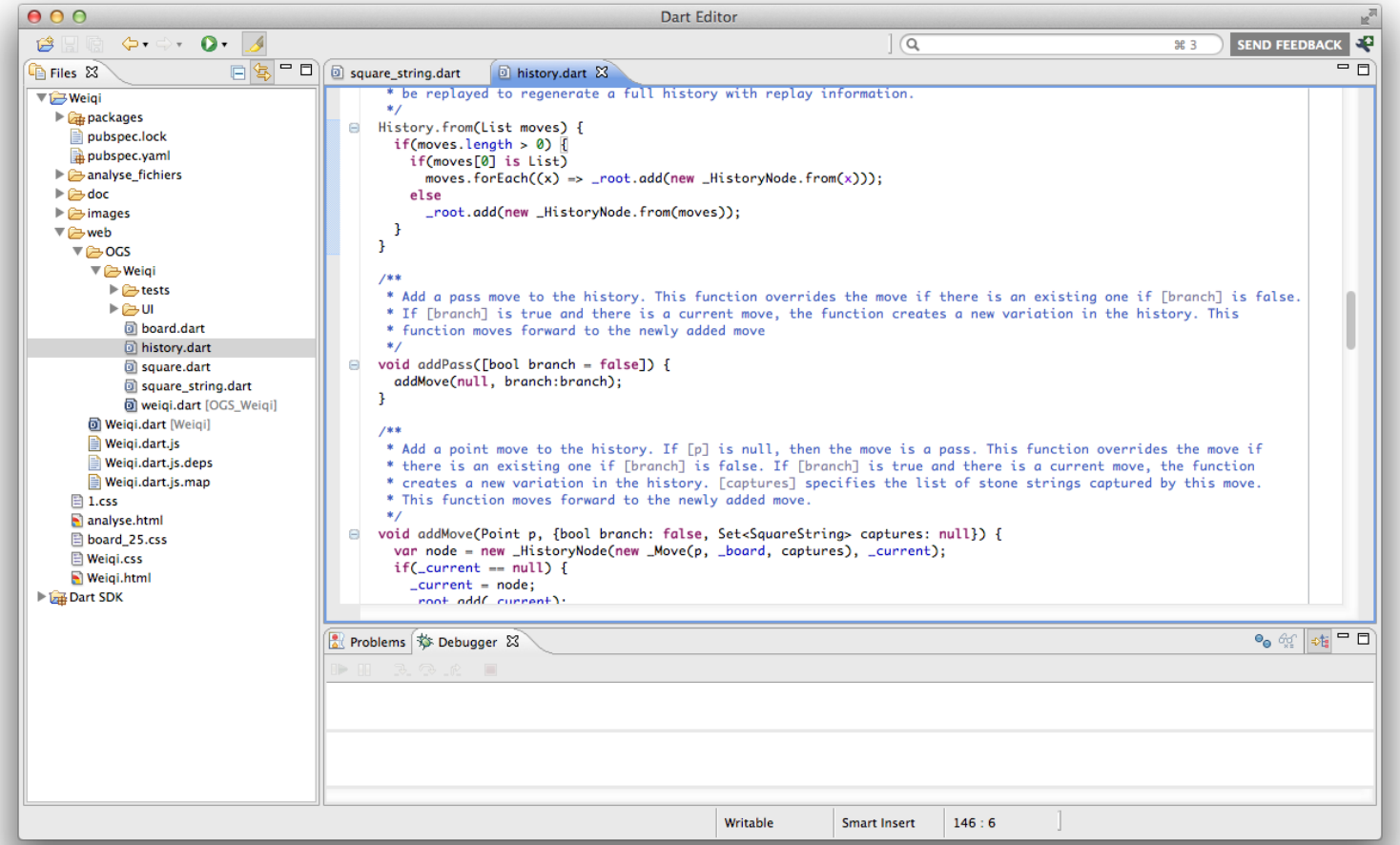


`https://www.dartlang.org/performance/`

And the dart virtual machine can be embedded in Chromium.

Dart Editor

An editor based on Eclipse is provided:



dartdoc

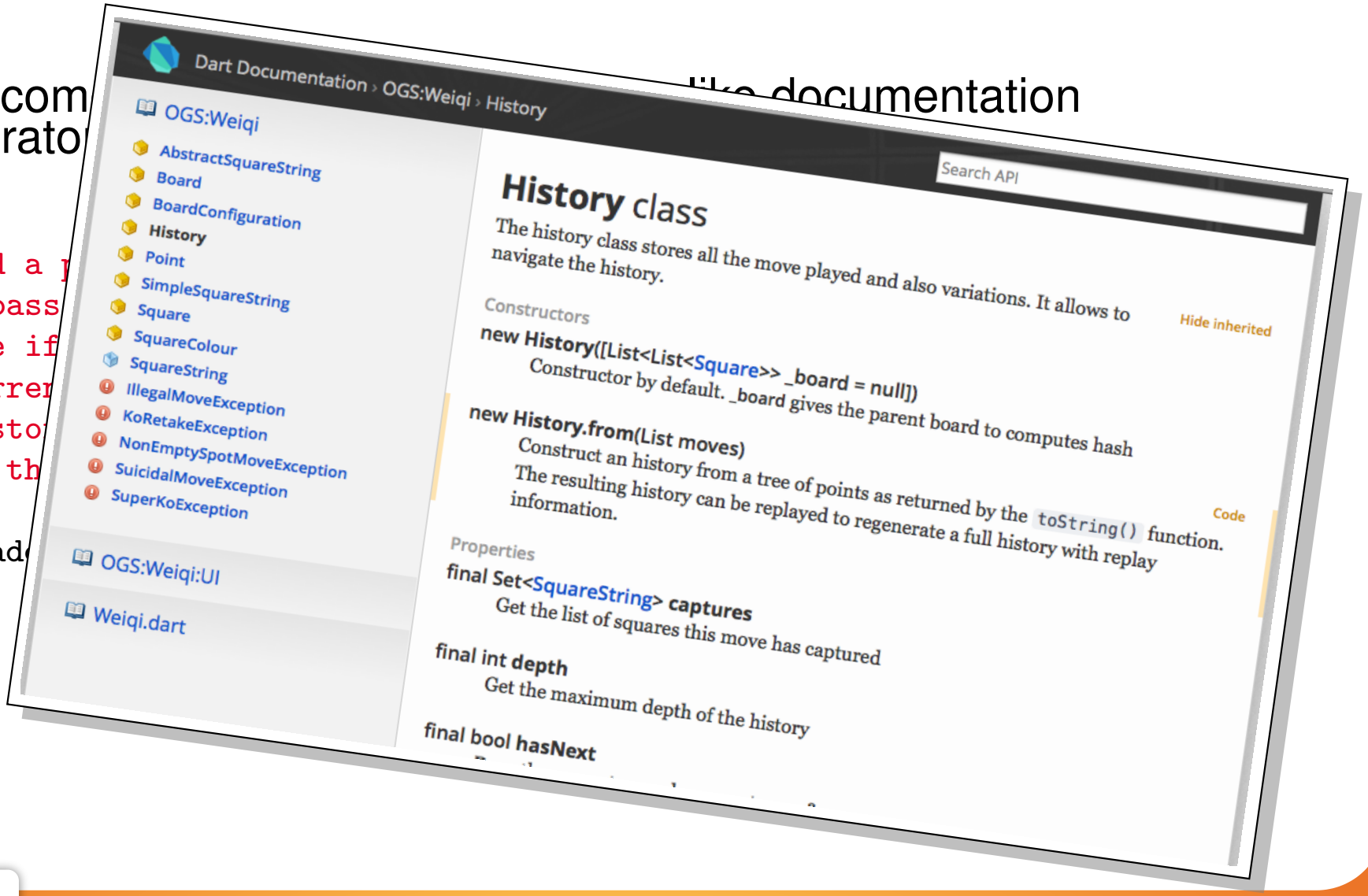
Dart comes with `dartdoc`, a JavaDoc-like documentation generator:

```
/**
 * Add a point move to the history. If [p] is null, then the move is
 * a pass. This function overrides the move if there is an existing
 * one if [branch] is false. If [branch] is true and there is a
 * current move, the function creates a new variation in the
 * history. [captures] specifies the list of stone strings captured
 * by this move. This function moves forward to the newly added move.
 */
void addMove(Point p, {bool branch: false,
        Set<SquareString> captures: null})
```

dartdoc

Dart com… …like documentation generato…

```
/**
 * Add a p
 * a pass
 * one if
 * curren
 * histo
 * by th
 */
void ad
```



5

Quick demo

6

Summary

Summary

- Dart tries to fix web development

Summary

- Dart tries to fix web development
- Dart offers a new language and a set of libraries

Summary

- Dart tries to fix web development
- Dart offers a new language and a set of libraries
- It compiles efficiently to JavaScript and have an efficient virtual machine.

Summary

- Dart tries to fix web development
- Dart offers a new language and a set of libraries
- It compiles efficiently to JavaScript and have an efficient virtual machine.

`https://www.dartlang.org`