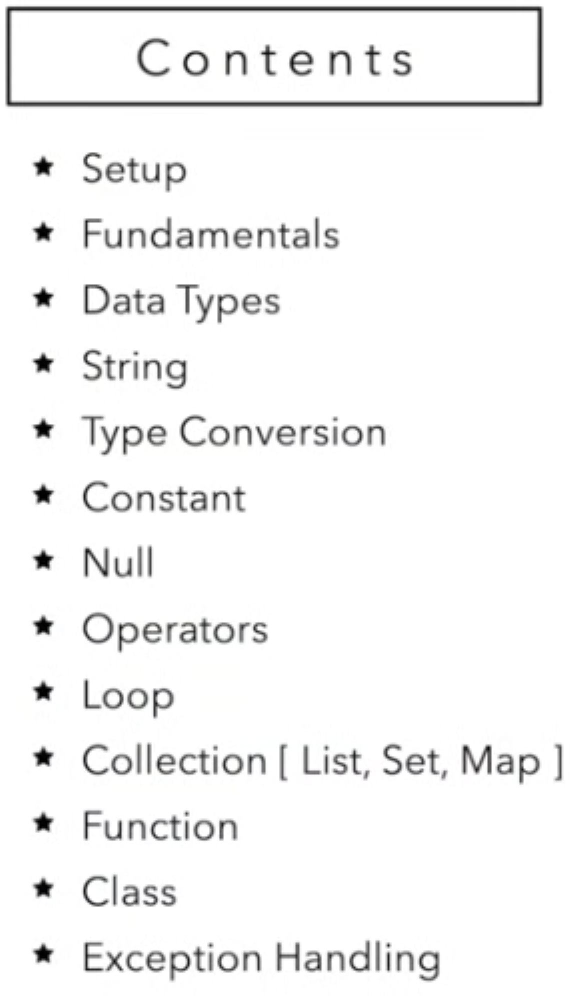
Dart



Contents

Setup

Dart-pad: https://dartpad.dev/

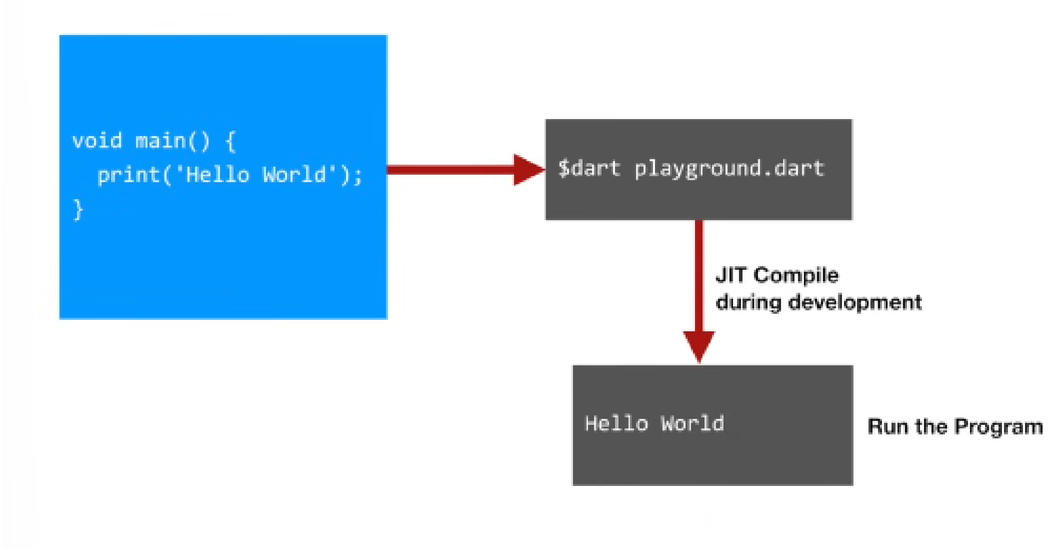
VS Code

Fundamentals

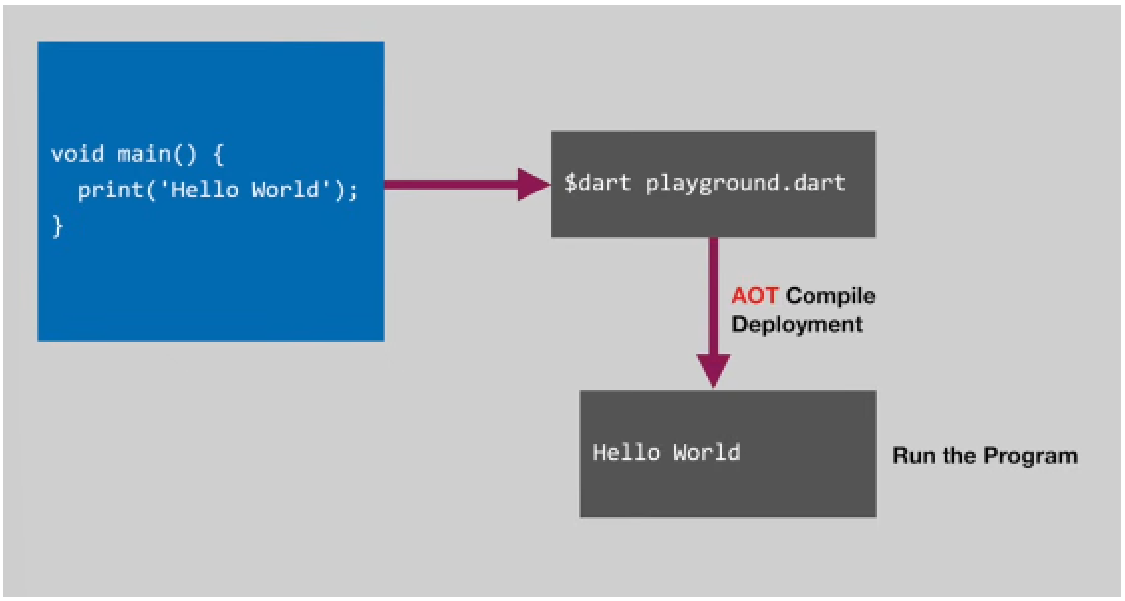
- Static type programming language
- Complie type language

→ Dart supports 2 types of compliation:

- AOT (Ahead of Time)
- JIT (Just In Time)



```
void main() {
  print('Hello World');
}
```



```
void main() {
  print('Hello World');
}
```

```
void main () {
   var firstName = "Kaustubh";
   String lastName = "Mukdam";
   print (firstName + ' ' + lastName);
}
// JIT Compilation
```

If we want to use built-in packages, then we have to use **dart:core**

```
import 'dart:core';  //This is taken sutomatically by compiler and we dont need to write it again and again
```

Input and Output

```
import 'dart:io';

void main() {
   stdout.writeln("What is your name? ");
   String name = stdin.readLineSync();
   print ("My name is $name");
}
```

string interpolation

Comments in Dart

```
// Type-1: In-Line Comment

/*
Block comment
mulitple line comment
*/

/// Documentation
```

Data Types

→ There are 2 types of Data types:

1. Strongly typed
2. Dynamically typed

```
/*
Strongly Typed Language: The type of a variable is known at compile time
Ex: C++, Java, Swift

Dynamic Typed Language: The type of a variable is known at run time
Ex: Python, Ruby, JS
*/

void main() {
   /* int, double, string, bool, dynamic */
   int amount1 = 100;
   var amount2 = 200;

   print('Amount-1: $amount1 | Amount-2: $amount2 \n');

   double damount1 = 100;
   var damount2 = 200;

   print('Amount-1: $damount1 | Amount-2: $damount2 \n');

   String samount1 = 'Kaustubh';
   var samount2 = 'Mukdam';

   print('Amount-1: $samount1 | Amount-2: $samount2 \n');

   bool bamount1 = true;
   var bamount2 = false;

   print('Amount-1: $bamount1 | Amount-2: $bamount2 \n');

   dynamic weakVariable = 100;
   print('Weak variable: $weakVariable \n');
```

```
    weakVariable = 'Dart Programming'
    print('Weak variable: $weakVariable \n');

    weakVariable = null;
    print(weakVariable);  //It will print null meaning that null is also an object
}
```

Dart is an Object Oriented Programming Language meaning everything here is an object (including null)

Strings, Type Conversion, Constant and Null

```
void main() {
    var s1 = 'Single Quotes for string literals\n';
    var s2 = "Double Quotes for string literals\n";
    var s3 = 'It\'s easy to escape the string delimiter';
    var s4 = "It's even easier to use other delimiter";

    print(s1);
    print(s2);
    print(s3);
    print(s4);

    //RAW String
    var s = r'In a raw string, not even \n gets special treatment';
    print(s);
}
```

Strings

```
void main() {
    var age = 21;
    var str = 'My age is $age';
    print(str);

    var s1 = '''
    you can create
    Multi-line string. \n''';

    var s2 = """This is also
    Multi-line string. \n""";

    print(s1);
    print(s2);
}
```

String conversion

```
void main() {
    //string → int
    var one = int.parse('1');
    assert(one == 1);

    //String → double
    var onePointOne = double.parse('1.1');
    assert(onePointOne == 1.1);

    //Error
    /*
    var error = int.parse('str');
    assert(error == 1);

    Output: Unhandled exception
    FormatException: Invalid radix-10 number (at character 1)
    str
    */

    //int → string
    String oneAsString = 1.toString();
    assert(oneAsString == '1');

    //double → string
    String piAsString = 3.14159.toStringAsFixed(2);
    assert(piAsString == '3.14');
}
```

Constant

```
void main() {
    const aConstNum = 0;  //int constant
  const aConstBool = true;  //bool constant
  const aConstString = "Hello";  //string constant

  print(aConstNum);
  print(aConstBool);
  print(aConstString);

  print(aConstNum.runtimeType);
  print(aConstBool.runtimeType);
  print(aConstString.runtimeType);
}
```

Null

```
void main() {
  int num;
  print(num);
}
```

Operators

```dart
void main() {
  int num = 10 + 20;
  num = num + 2;

  print(num);

  num = num % 5;
  print(num);

  //Relational operators: ==, !=, >=, <=
  if (num == 0) {
    print('Zero');
  }

  num = 100;
  num *= 2;
  print(num);

  //Unary operator
  ++num;
  num++;
  num += 1;
  num -= 1;
  print(num);

  //Logical Operators (&& and ||)
  if (num > 200 && num < 203) {
    print('201 or 202');
  }

  //!= (Not equal)
  if (num != 100) {
    print('Num is not equal to 100');
    }
}
```

Null Aware Operator

```
// (?.), (??), (??=)

class Num {
    int num = 10;
}

void main() {
    var n = Num();
    int number;

    if (n != null) {
        number = n.num;
    }

    print(number);  //This will print 10

    var n1;
    int number;

    if (n1 != null) {
        number = n1.num;
    }

    print(number);  //This will print null

    var n2;
    int number;

    //if (n2 != null) {
        number = n2.num;
    //}

    print(number);  //Now this will show an error

    //So if we dont want this error, we can use null aware operator
    var n3;
    int number;

    //if (n3 != null) {
    number = n3?.num;
    //}

    print(number);  //This will print null

    var n3 = Num();
    int number;

    //if (n3 != null) {
    number = n3?.num;
    //}

    print(number);  //This will print 10
}
```

Variation-1

```
//If we want to have a default value instead of null using null aware opperator

class Num {
  int num = 10;
}

void main() {
  var n = Num();
  int number = n?.num ?? 0;
  print(number);
}
```

Ternary Operator

```
//Ternary Operator

void main() {
  int x = 100;
  var result = x % 2 == 0 ? 'Even' : 'Odd';
  print(result);
}
```

Type Test

```
//Type Test

void main() {
    var x = 100;

    if (x is int) {
        print('Integer');
    }
}
```

Conditional Statement

```
//Conditional Statement

void main() {
    int number = 100;

    if (number % 2 == 0) {
        print('Even');
    }
    else if (number % 3 == 0) {
        print("Odd");
    }
    else {
        print("Confused");
    }
}
```

Switch Statement

```
//Switch Statement

void main() {
    int number = 0;

    switch (number) {
        case 0:
            print('Even');
            break;

        case 1:
            print('Odd');
            break;

        default:
            print('Confused');
    }
}
```

Loops

1. For loop

```
void main() {
   //Standard for loop
   for (var i = 1; i < 11; i++) {
      print(i);
   }
}
```

2. For-in loop

```
void main() {
   //For in loop
   var numbers = [1, 2, 3, 4];

   for (var a in numbers) {
      print(a);
   }

   //Method-2
   for (var b = 0; b < numbers.length; b++) {
      print(numbers[b]);
   }
}
```

3. For-each loop

```
void main() {
   //for-each loop
   var fiveNumbers = [1,2,3,4,5];

   fiveNumbers.forEach( (n) => print(n) );
}
```

4. While loop

```
void main() {
   int num = 10;

   while (num > 0) {
      print(num);
      num -= 1;
   }
}
```

5. Do-while loop

```
void main() {
   int num1 = 10;

   do {
      print(num1);
      num1 -= 1;
   }while (num1 > 0);
}
```

Break and Continue

```
void main() {
   for (var i = 0; i < 10; i++) {
      if (i > 5) break;
      print(i);
   }
}
```

Question: Print the odd numbers using break statement

```
void main() {
   for (var j = 0; j < 10; j++) {
        if (j % 2 == 0) continue;
        print("Odd: $j");
    }
}
```

Collection [List, Set, Map]

There are 3 types of collection techniques in Dart:

1. List

→ Ordered Collection of values

```
//Collection

void main() {
    //List
    List names = ['Kaustubh', 'Kabir', 'Om'];
    print(names[0]);

    //Find the length of the list
    print(names.length);

    //Instead of List, we can use var
  var names1 = ['Kaustubh', 'Kabir', 'Om'];
  print(names1[0]);
  print(names1.length);

  //Accessing values from list
  for (var n in names) {
      print('names is $n');
     }

    //We can also have different type of data in a single list
    var diffdata = ['Kaustubh', 'Kabir', 'Om', 0.1, 100, 101.234];  //Default type of
this list is Object
```

```
    print(diffdata);

    //If we want to set a specific datatype to the list, we can do that too
    List<String>diffdata1 = ['Kaustubh', 'Kabir', 'Om', 0.1, 100, 101.234];
    //Then we have to pass only String data and other data won't be allowed
    print(diffdata1);

    names[0] = 'Max';  //As List is mutable
    print(names);

    //If we want to not change the data at runtime, we can use the 'const' keyw
ord
    List names2 = const ['Kaustubh', 'Kabir', 'Om'];

    names2[0] = 'Max';  //As List is mutable
    print(names2);  //It will show error 'Unsupported operation'

    //Sharing List to antother variable of type list
    var shareList = names2;
    print(shareList);

    //But here both 'names2' and 'shareList' are pointing to the same list so any
change in the list will change both of them and this is not exactly copying

    var shareList1 = [...names];
  names[0] = 'Charles';
  print(shareList);
    print(shareList1);
}
```

→ List is mutable

2. Set

→ Set is a collection of unique items in unordered fashion

```
//Collection

void main() {
  //Set
  var names = {'Max', 'Charles', 'Max'};

  for (var n in names) {
    print(n);
  }

  //If we want to show empty set then we cannot directly write it as
  var emptySet = {};

  print(emptySet.runtimeType);  //This will show LinkedHashMap

  //To show a Set
  var emptySet1 = <String>{};
  print(emptySet1.runtimeType);
}
```

3. Map

```
//Collection

void main() {
  //Map
  var grandPrix = {
    //Key : Value
    'Japanese' : 'Suzuka',
    'Abu Dhabi' : 'Yas Marinas',
    'Qatar' : 'Bahrain',
    'India' : 'Buddha',
    'Italy' : 'Monza'
  };
```

```
    var grandPrixLocation = {
        //Key : Value
        1 : 'Suzuka',
        2 : 'Yas Marinas',
        3 : 'Bahrain',
        4 : 'Buddha',
        5 : 'Monza'
    };

    var f1terms = Map();
  f1terms['Max Verstappen'] = 'Red Bull';
  f1terms['Charles Leclerc'] = 'Ferrari';

    print(grandPrix['India']);
    print(grandPrixLocation[3]);
    print(f1terms['Max Verstappen']);
}
```

functions

```
void main() {
    showOutput(square(2));
    print(square.runtimeType);
}

dynamic square(var num) {
    return num * num;
}

dynamic showOutput(var message) {
    print(message);
}
```

```
//Arrow function ( ⇒ )
dynamic add(var num) ⇒ num + num;
```

```
//Anonymous function: Function that has no name

void main() {
  var list = ['Apple', 'Banana', 'Orange'];
  list.forEach(printf);
  list.forEach((item) {
    print(item);
  });
}

void printf(item) {
  print(item);
}
```

Parameters in functions

There are 2 types of parameters in dart:

1. Positional Parameter/Argument
2. Named Parameter/Argument
3. Mixed Parameter: We can use both positional and named in one function
4. Default parameter: We can set a value as default while intializing the function

```
//Parameters in dart

void main() {
  print(positionalSum(2, 3));
  print(namedSum(n2: 4, n1: 3));
  print(mixedSum(10, n2: 8));
  print(mixedSum1(10));
  print(mixedSum1(10, n2: 2));
  print(defaultSum(20));
  print(defaultSum(15, n2: 3));
}

dynamic positionalSum(var n1, var n2) ⇒ n1 + n2;  //Postional Parameters

dynamic namedSum({var n1, var n2}) ⇒ n1 + n2;  //Named Parameters

dynamic mixedSum(var n1, {var n2}) ⇒ n1 + n2;  //Mixed Parameters

//We can have default value to a named parameter
dynamic mixedSum1(var n1, {var n2}) ⇒ n1 + (n2 ?? 0);

dynamic defaultSum(var n1, {var n2 = 2}) ⇒ n1 + n2;
```

Class

```
//Class

class Person {
   late String name;
   late int age;

   //Constructor: Method without any return type
   /*Person(String name, [int age = 18]) {
      this.name = name;
      this.age = age;
   }*/

  Person(this.name, [this.age = 18]);

   void showOutput() {
      print(name);
      print(age);
    }

}

void main() {
    Person p1 = Person('Kaustubh', 21);
    p1.showOutput();

  var p2 = Person('Kabir');
  p2.showOutput();
}
```

Final Keyword in class

```
//Class

class X {
  final name; //type will be defined by inferred value
  static const int age = 18;

  X(this.name);
}

void main() {
  var x = X('Kaustubh');
  print(x.name);
    print(X.age);
    x.name = 'Kabir'  //This will show error as it cannot have 2 different value b
ecause of final keyword
}
```

Inheritance

```
//Class - Inheritance

class Vehicle {
  String model;
  int year;

  Vehicle(this.model, this.year) {
    print(this.model);
    print(this.year);
  }

  void showOutput() {
    print(model);
    print(year);
  }
}

class Car extends Vehicle {
  double price;

  Car(String model, int year, this.price) : super(model, year);

  void showOutput() {
    super.showOutput();
    print(this.price);
  }
}

void main() {
  Car c1 = new Car('BMW', 2019, 1000000.0);
  c1.showOutput();
}
```

Method Overriding

```
//Class - Method overriding

class X {
  String name;

  X(this.name);

  void showOutput() {
    print(this.name);
  }

  dynamic square(dynamic val) {
    return val * val;
  }
}

class Y extends X {
  Y(String name) : super(name);

  @override
  void showOutput() {
    print(this.name);
    print('This is Y class');
  }
}

void main() {
  var x = X('Kaustubh');
  x.showOutput();
  var y = Y('Kabir');
  y.showOutput();
}
```

Getters and Setters

```
//Class - Getters and Setters

class Rectangle {
   num left, top, right, bottom;
   Rectangle(this.left, this.top, this.right, this.bottom);

   //Define two calculated properties: width and height
   num get width => left + right;
   set width(num value) => left = value - right;

   num get height => top + bottom;
   set height(num value) => top = value - bottom;
}

void main() {
   var rect = new Rectangle(10, 20, 30, 40);
   rect.right = 50;
   assert(rect.right == 50);
   rect.bottom = 60;
   print(rect.left);
}
```

Error Handling

```
//Exception Handling

int greaterThanZero (int val) {
  if (val <= 0) {
   throw Exception('Value must be greater than 0');
  }
  return val;
}

void letVerifyValue(int val) {
  try {
    greaterThanZero(val);
  } catch (e) {
    print(e);
  }
  finally {
    print('Done');
  }
}

void main() {
  letVerifyValue(-1);
  letVerifyValue(1);
}
```