Functions and More Data Types

By: Laams Innovation Lab

Revision

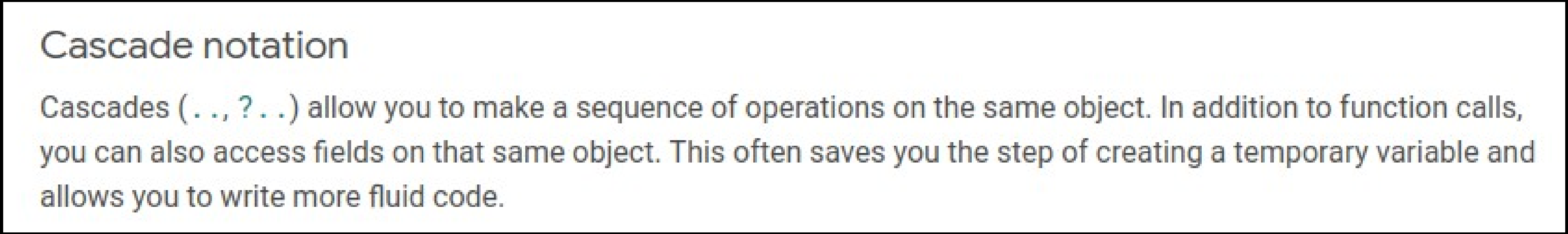
- Write a program which prints out from 1-100 except 50
- literals
  - Concatenation
  - Interpolation
- Control flow statements
  - Conditional execution, loop and iteration
  - Iterating through a string
- Create a numbers multiplication table from 1 to ..

Contents

- **TEDED**
- More on literals and operators
- Dart’s built-in Data Types
- More Operators
- What are data Structures
- Touching on generics a bit
- Creating a few CLI applications

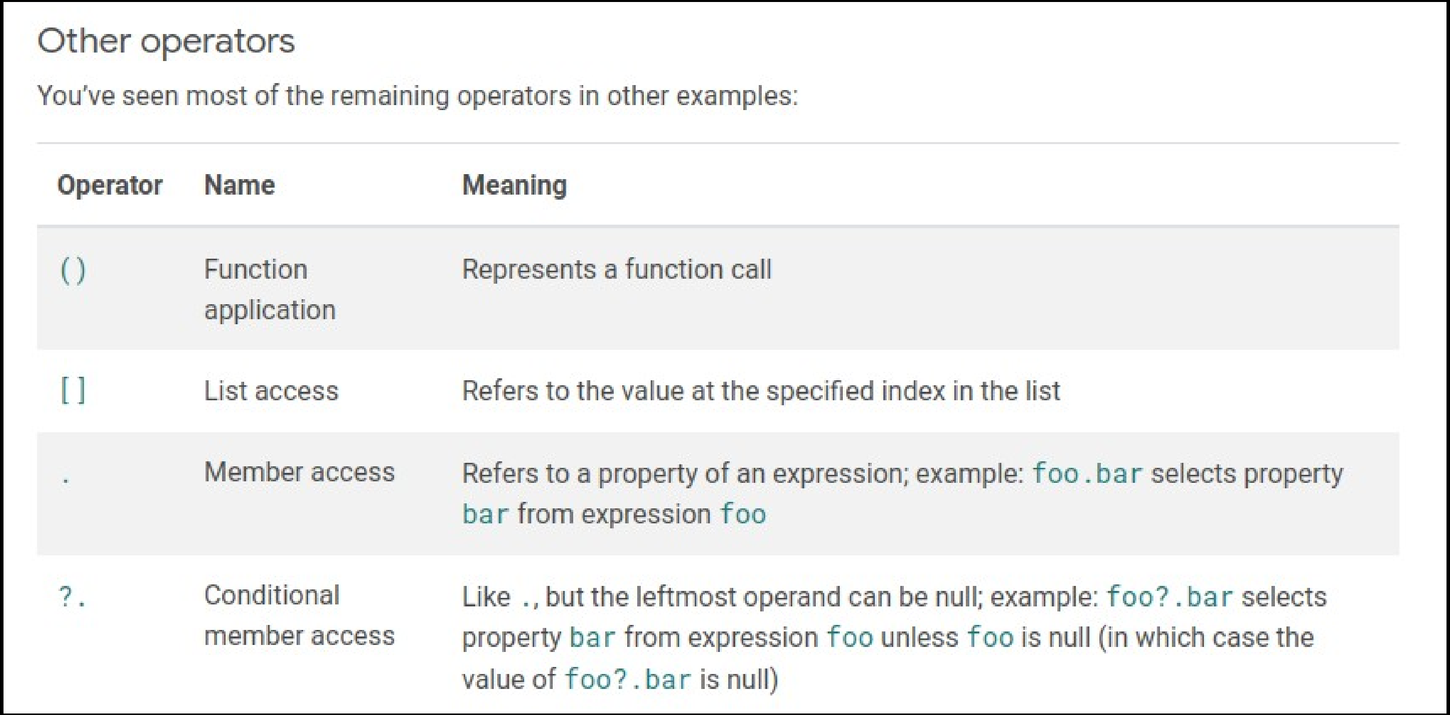
More on Literals and Objects

- \: Called escape character, are used to convert special characters into normal string, and normal strings such as n and t into special characters inside a string literal.
- $: Called called special character (format specifier), and is used for interpolation inside a String literal.
- \t: is used to [tab] space between its preceding and succeeding characters inside a String literal.
- \t: Creates a formatted new line inside a String literal
- Constructors: All instantiate-able classes have constructors, including the primitive data types. They are special functions which instantiates an object.



Cascade notation

Cascades (`..`, `?..`) allow you to make a sequence of operations on the same object. In addition to function calls, you can also access fields on that same object. This often saves you the step of creating a temporary variable and allows you to write more fluid code.



Other operators

You've seen most of the remaining operators in other examples:

Operator	Name	Meaning
`()`	Function application	Represents a function call
`[]`	List access	Refers to the value at the specified index in the list
`.`	Member access	Refers to a property of an expression; example: `foo.bar` selects property `bar` from expression `foo`
`?.`	Conditional member access	Like `.`, but the leftmost operand can be null; example: `foo?.bar` selects property `bar` from expression `foo` unless `foo` is null (in which case the value of `foo?.bar` is null)

Functions

Functions provide an unconditional way to transfer control to another part of our code; hence, prevents code repetition, and helps us think at a higher level of abstraction.

- **Benefits:**
  - Prevents code repetition
  - Decreases compilation time
  - Simplifies Code Reuse
  - Defines a clear interface
  - Provides a natural way for creating algorithms
  - Provides better way of error handling
  - Makes our code modular

Function Syntax

```
DataType  identifier ( Parameter(s) ) {
return value
}
```

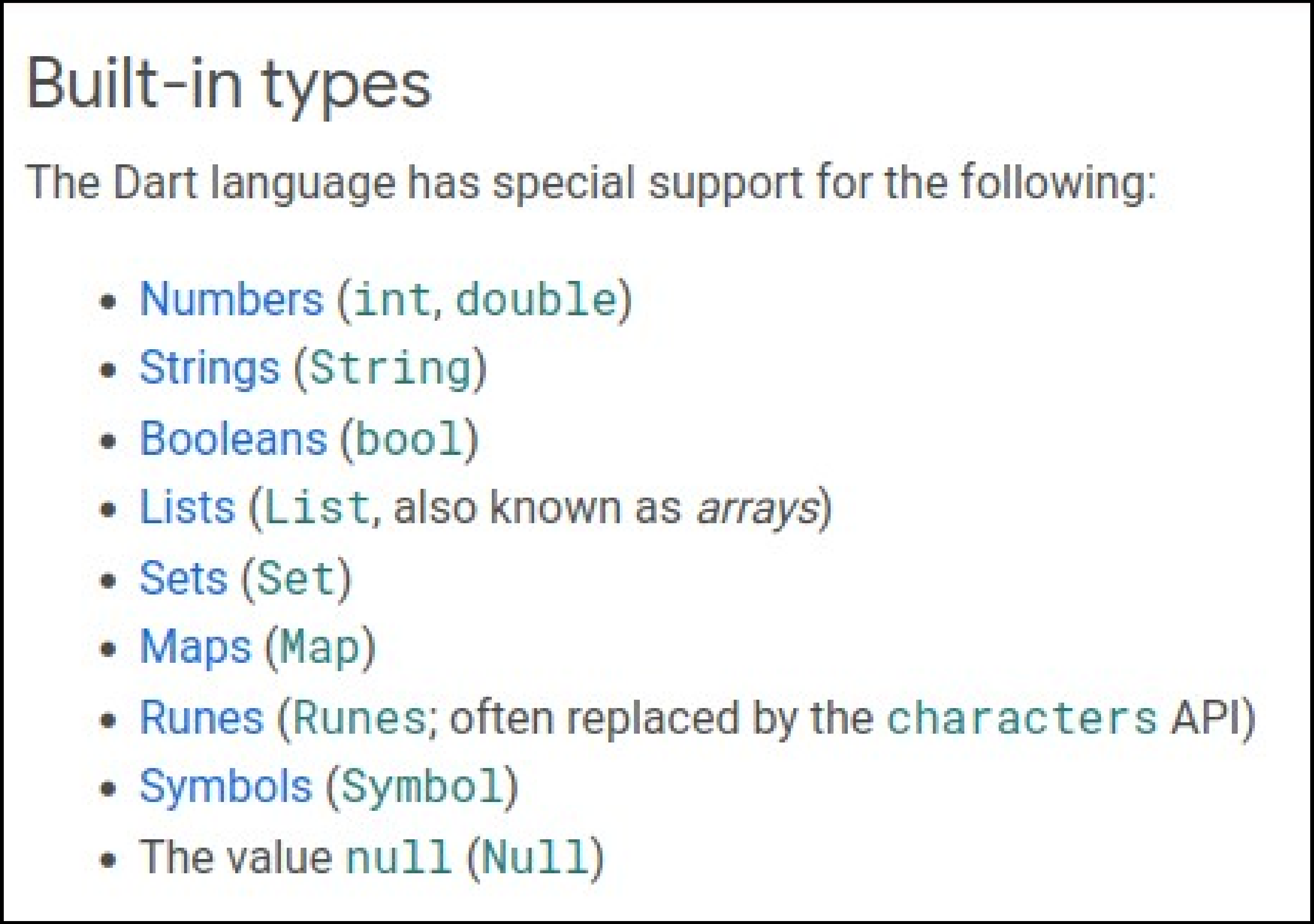
- **Parts of Function:**
- **Data Type:** signifies the type of the data that the function will give out once returned
  - void return type signifies that the function does not return any usable object.
- **Identifier:** name of the function that can be used to call it.
- **Parameters:** defined to make passing values into the function possible.
- **The Function Block:** It is where the subroutine (algorithms) are defined.
- **return**: Returns to the part of the code from where the function was called.
- **throw**: Instead of returning something the function can throw and **exception** or **error**.
- **value:** the instantiated object which must be the same as the returning data type.

Steps in creating a Function

- **Signature Declaration:** A function signature can be defined inside an abstract class to make polymorphism possible.
- **Definition:** in order for a function to be usable it must have a concrete implementation or rather be defined.
  - Function overloading is not allowed in Dart
- **Function Prototype**: includes the declaration, definition, parameters, and return type of a function.
- **Function call**: to use a function inside your code it must be called.

The main function

- It is the entry point to our application.
- It cannot be renamed, and a compile-able dart project must have a main function.
- It is the only function which is not called by the developer.
- It is takes a optional parameter of List<String> args.
- Its return type is usually void



Built-in types

The Dart language has special support for the following:

- Numbers (`int`, `double`)
- Strings (`String`)
- Booleans (`bool`)
- Lists (`List`, also known as *arrays*)
- Sets (`Set`)
- Maps (`Map`)
- Runes (`Runes`; often replaced by the `characters` API)
- Symbols (`Symbol`)
- The value `null` (`Null`)