Dart

Operators

Control flow statements

Exceptions

Operators

Arithmetic operators

- + , - , * , /
- ~/ : Divide, returning an integer result
- % : Get the remainder of an integer division

```
assert(5 / 2 == 2.5); // Result is a double
assert(5 ~/ 2 == 2); // Result is an int
assert(5 % 2 == 1); // Remainder
```

Arithmetic operators - prefix and postfix

- ++ var -> var = var + 1 (expression value is var + 1)
- -- var -> var = var – 1 (expression value is var - 1)
- Var ++ -> var = var + 1 (expression value is var)
- Var -- -> var = var – 1 (expression value is var)

- a = 0;
  b = ++a; // Increment a before b gets its value.
  assert(a == b); // 1 == 1
- a = 0;
  b = a++; // Increment a AFTER b gets its value.
  assert(a != b); // 1 != 0

Equality and relational operators

- == , != , > , < , >= , <=

-
  ```
  assert(2 == 2);
  assert(2 != 3);
  assert(3 > 2);
  assert(2 < 3);
  assert(3 >= 3);
  assert(2 <= 3);
  ```

Type test operators

- as : Typecast
- is : True if the object has the specified type
- is! : False if the object has the specified type

- if (emp is Person) {
  emp.firstName = 'Bob’;
  }
- (emp as Person).firstName = 'Bob';

Assignment operators

- =

- // Assign value to a
- a = value;
- // Assign value to b if b is null; otherwise, b stays the same
- b ??= value;

Compound assignment operators

- = , += , -= , *= , /= , ~/= , %= , <<= , >>= , &= , ^= , |=

- a op= b -> a = a op b
- Ex. a += b -> a = a + b

- var a = 2; // Assign using =
- a *= 3; // Assign and multiply: a = a * 3
- assert(a == 6);

Logical operators

- ! : inverts the following expression
- || : OR
- && : AND

Bitwise and shift operators

- & : AND
- | : OR
- ^ : XOR
- ~expr : Unary bitwise complement
- << : Shift left
- >> : Shift right

Conditional expressions

- condition ? expr1 : expr2
  - true then return expr1, otherwise return expr2.
- expr1 ?? expr2
  - If expr1 is non-null, returns its value, otherwise return expr2.

- var visibility = isPublic ? 'public' : 'private’;
- String playerName(String name) => name ?? 'Guest';

Cascade notation (..)

-
  ```
  querySelector('#confirm') // Get an object.
    ..text = 'Confirm' // Use its members.
    ..classes.add('important’)
    ..onClick.listen((e) => window.alert('Confirmed!’));
  ```

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

Other operators

- () -> Represents a function call
- [] -> Refers to the value at the specified index in the list
- . -> Refers to a property of an expression;
- ?. -> Like ., but the leftmost operand can be null;

- Ex. foo.bar //selects property bar
- Ex. foo?.bar //selects property bar unless foo is null

Control flow statements

If and else

```
if (isRaining()) {
  you.bringRainCoat();
} else if (isSnowing()) {
  you.wearJacket();
} else {
  car.putTopDown();
}
```

For loops

- var message = StringBuffer('Dart is fun');
- for (var i = 0; i < 5; i++) {
-   message.write('!');
- }
- //Dart is fun!!!!!

Iterable – foreach , for-in

- Using forEach() is a good option if you don’t need to know the current iteration counter:
- candidates.forEach((candidate) => candidate.interview());

- Iterable classes such as List and Set also support the for-in form of iteration:

-
  ```
  var collection = [0, 1, 2];
  for (var x in collection) {
    print(x); // 0 1 2
  }
  ```

While and do-while

-
  ```
  while (!isDone()) {
    doSomething();
  }
  ```

-
  ```
  do {
    printLine();
  } while (!atEndOfPage());
  ```

Break and continue

-
  ```
  while (true) {
    if (shutDownRequested()) break;
    processIncomingRequests();
  }
  ```

-
  ```
  for (int i = 0; i < candidates.length; i++) {
    var candidate = candidates[i];
    if (candidate.yearsExperience < 5) {
      continue;
    }
    candidate.interview();
  }
  ```

Switch and case

- Each non-empty case clause ends with a break statement, as a rule. Other valid ways to end a non-empty case clause are a continue, throw, or return statement.

-
  ```
  var command = 'CLOSED’;
  switch (command) {
    case 'CLOSED': // Empty case falls through.
    case 'NOW_CLOSED’:
      // Runs for both CLOSED and NOW_CLOSED.
      executeNowClosed();
      break;
  }
  ```

Switch and case - continue

-
  ```
  var command = 'CLOSED’;
  switch (command) {
    case 'CLOSED’:
      executeClosed();
      continue nowClosed;
  // Continues executing at the nowClosed label.

    nowClosed:
    case 'NOW_CLOSED’:
      // Runs for both CLOSED and NOW_CLOSED.
      executeNowClosed();
      break;
  }
  ```

Assert

- // Make sure the variable has a non-null value.
- assert(text != null);

- // Make sure the value is less than 100.
- assert(number < 100);

- // Make sure this is an https URL.
- assert(urlString.startsWith('https'));

Assert - message

- To attach a message to an assertion, add a string as the second argument to assert.

- assert(urlString.startsWith('https’),
    'URL ($urlString) should start with "https".');

Exception

Throw

- throw FormatException('Expected at least 1 section’);
- //Uncaught exception:
  FormatException: Expected at least 1 section

- throw 'Out of llamas!’;
- //Uncaught exception:
  Out of llamas!

- void distanceTo(Point other) => throw UnimplementedError();

Catch

-
  ```
  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’);
  }
  ```

Catch - parameters

-
  ```
  try {
    // …
  } on Exception catch (e) {
    print('Exception details:\n $e’);
  } catch (e, s) {
    print('Exception details:\n $e’);
    print('Stack trace:\n $s’);
  }
  ```

Rethrow

```
• void misbehave() {
    try {
      dynamic foo = true;
      print(foo++);
    } catch (e) {
      print('misbehave() partially
  handled ${e.runtimeType}.’);
      rethrow;
    }
  }
```

```
void main() {
  try {
    misbehave();
  } catch (e) {
    print('main() finished handling
${e.runtimeType}.’);
  }
}
```

Finally

- To ensure that some code runs whether or not an exception is thrown, use a finally clause.

-
  ```
  try {
    breedMoreLlamas();
  } catch (e) {
    print('Error: $e'); // Handle the exception first.
  } finally {
    cleanLlamaStalls(); // Then clean up.
  }
  ```