Dart cheat sheet

Build-in types

```
Numbers: num, int, double
Strings: String, StringBuffer
Booleans: bool, true, false
Lists (arrays): [0,1,2,3]
Sets (unique): {'A', 'B', 'C'}
Maps: {'key': 'value'}
```

Variables

```
var name = 'CodeSabai';
dynamic name = 'CodeSabai';
String name = 'Code' + 'Sabai';
List<String> myList = ['C','O'];
var mySet = <String> {};
var myMap = {54: 'xenon'};
final name = 'Bob'; // set only once
const bar = 1000000; // compile-time
```

Functions

```
int addNumber (int num1, int num2) {
      return num1 + num2;
}
// omit the types
addNumber (num1, num2) {
      return num1 + num2;
}
// named parameters
void enableFlags ({bool bold, bool
hidden}) {...}
enableFlags (bold: true, hidden:
false);
// required
Scrollbar ({Key key, @required Widget
child})
// default parameter values
enableFlags ({bool bold = false, bool
hidden = false}) {...}
// anonymous functions
var list = ['apples','bananas'];
list.forEach ( (item) =>
print('${list.indexOf(item)}: $item'));
});
```

Control flow statements

```
// if else
if (isRaining()) {
      you.bringRainCoat();
} else if (isSnowing()) {
      you.wearJacket();
} else {
      car.putTopDown();
}
// for loops
for (var i = 0; i < 5; i++) {
      print(i);
}
// while
while (!isDone()) {
doSomething();
}
do {
      printLine();
} while (!atEndOfPage());
// switch case
var command = 'OPEN';
switch (command) {
      case 'CLOSED':
            executeClosed();
            break;
      case 'OPEN':
            executeOpen();
            break;
      default:
            executeUnknown();
}
// assert (development only)
assert (number < 100);
```

Exceptions

```
try {
      breedMoreLlamas();
} catch (e) {
      print('Error: $e');
} finally {
      cleanLlamaStalls(); }
```

 https://github.com/codesabai https://www.facebook.com/codesabai

Dart cheat sheet

Classes

```
class Point {
  num x, y;
  // static variable
  static const fixedNumber = 16;
  // constructor
  Point(this.x, this.y);
  // named constructor
  Point.origin() {
  x = 0;
  y = 0;
  }
  // initializer constructor
  Point.fromJson(Map<String, num> json)
    : x = json['x'],
      y = json['y'] {
    print('In Point.fromJson(): ($x, $y)');
  }
}
// invoking non-default constructor
class Employee extends Person {
  Employee.fromJson(Map data) :
            super.fromJson(data) {
      // do something
  }
}
// factory constructors
class Logger {
  final String name;
  bool mute = false;
   static final Map<String, Logger> _cache =
    <String, Logger>{};

  factory Logger(String name) {
    if (_cache.containsKey(name)) {
      return _cache[name];
    } else {
      final logger =
            Logger._internal(name);
      _cache[name] = logger;
      return logger;
    }
  }
  Logger._internal(this.name);
  void log(String msg) {
    if (!mute) print(name + ' ' + msg);
  }
}
```

Abstract Classes

```
abstract class Doer {
      void doSomething();
}
class EffectiveDoer extends Doer {
      void doSomething() {
            print('something');
      }
}
class Greeter implements
EffectiveDoer {
      doSomething () {
            print('Hello');
      }
}
```

Mixins

```
// multiple class hierarchies
class Musician extends Performer with
Musical, Conductor, Composer {
}
mixin Musical {
      bool canPlayPiano = true;
      void entertainMe() {
            print('Playing piano');
      }
}
```

Asynchrony

```
Future checkVersion() async {
      try {
        version = await lookUpVersion();
      } catch (e) {
        Print(e.toString);
      }
// Do something with version
}
```