Dart Syntax Cheatsheet

Thomas Boxall
`up2108121@myport.ac.uk`

April 2023

1 Basics of Dart

entry point to a Dart program is the `main` function:

```
void main(){

}
```

print to the console: `print("hello world");`
comments start with: `//`

2 Types

declare an integer variable: `int imAnInt = 12;`
declare a double variable: `double imADouble = 6.9;`
declare a string variable: `String imAString = "Cheese";`
declare a constant: `const pi = 3.14;`
declare a boolean: `bool dartIsBetterThanPython = true;`
declare a num: `num iCanBeDoubleOrInt = 4;`

2.1 Casting

cast to a string: `String wasANumber = numberVariable.toString();`
cast a double to string with fixed decimal places:
`String wasADouble = doubleVariable.toStringAsFixed(numbDecimalPlaces);`
cast a string to an integer: `int wasAString = int.parse(stringVariable);`
cast a string to a double: `double wasAString = double.parse(stringVariable);`
cast an int or num to double: `double wasAnInt = intVariable.toDouble();`
cast a double or num to int: `int wasADouble = doubleVariable;`
or `int wasADouble = doubleVariable.toInt();`
cast a double or num to int and round = `int wasADouble = doubleVariable.round();`
cast a double or num to int and round to upper bound: `int wasADouble = doubleVariable.ceil();`
cast a double or num to int and round to lower bound: `int wasADouble = doubleVariable.floor();`

3 Math

import math library: `import 'dart:math';`
calculate power of number: `pow(base, exponent)`
calculate square root: `sqrt(numberToSquareRoot)`

4 Strings

concatenate two strings: `print("String One" + "String Two");`
interpolate a non-string into a string: `print("I'm a string, $andImAndIntVariable");`
interpolate a non-string into a string and apply an operation to it: `print("String: ${intVariable * 2}");`
use string indexing to access the 1st character in a string: `print(stringVariable[0]);`
use substring to access particular characters in a string:
`print(stringVariable.substring(start, endOptional));`
get the length of a string: `print(stringVariable.length);`
get the index of a particular character in a string: `print(stringVariable.indexOf(queryChar));`
split a string and remove the pattern: `stringVariable.split(pattern)`
convert all characters in the string to lower case: `stringVariable.toLowerCase()`
convert character to ASCII value: `stringVariable.codeUnitAt(position)`

5 Functions

general structure to a Dart function:

```
returnType functionName(param1Type paramOne, ...){

}
```

a function made of an expression can be simplified:
`returnType functionName (param1Type, param1) => expressionToReturn;`
a function can be passed as a parameter to another function using `returnType Function(paramsType) name` which can then be called using `name()`, passing any parameters into it.

6 Program Flow Control

if statements have the following structure:

```
if (statement){

} else if (anotherStatement){

} else{

}
```

for loops have the following structure:

```
for (loopVarStartPoint; loopVarEndPoint; loopVarIteration){

}
```

while statements have the following structure:

```
while(statement){

}
```

7 Lists

declare a list with: `List<elementType> listName = [itemOne, ...];`
access an item within a list: `listName[listIndex];`

update an item within a list: `listName[listIndex] = newValue;`
get the length of a list: `listName.length;`
add an item to a list: `listName.add(itemToAdd);`
get the first item from a list: `listName.first;`
get the last item in a list: `listName.last;`
insert an item at a specific index: `listName.insert(index, itemToAdd);`
generate a list pre-filled: `List<type> listName = List<type>.filled(numberElems, elemContent);`
iterate through all elements in a list:

```
for (type individualElemIdentifier in listName){
    //do a thing
}
```

declare a multidimensional list:

```
List<List<type>> listName = [
    [element00, element01, element02],
    [element10, element11, element12],
    [element20, element21, element22, element23],
    ...
];
```

8 Maps

declare a map:

```
Map<keyType, valType> mapName = {
    key: val,
    key: val,
    ...
};
```

access a value: `mapName[key];`
update a value: `mapName[key] = newValue;`
add a new key-val pair: `mapName[newKey] = newVal;`
remove a key-val pair from a map: `mapName.remove(keyOfPairToRemove);`
iterate through a map:

```
for (type key in mapName.keys){
    // do a thing
}
```

declare a map containing a string key and list of strings value:

```
Map<String, List<String>> mapName = {...};
```

update an element of a list which is a a value in a map:

```
mapName[key]![listIndex] = newVal;
```

9 Classes & Objects

define a class with two integer values:

```
class ClassName{
    int valueOne = 0;
    int valueTwo = 0;
}
```

define the constructor method: `ClassName(this.attributeName, ...);`
add a method to get an attribute: `int get valueOne() => valueOne;`
add a `toString()` method

```
@override
String toString(){
    return "stringValueHere";
}
```

define a class which inherits another

```
class Super{
    ...
}
class Subclass{
    Subclass(this.attribute) : super(attribute);
    ...
}
```

instantiate an object: `type name = type(constructorParams);`