This is a summary about the Dart basics I covered in the previous two lectures. You also find this summary attached as a PDF.

What is Dart?

Dart is an **object-oriented programming language** developed by Google. Whilst technically not restricted, it's primarily used for creating **frontend user interfaces** for the web (with AngularDart or Flutter for Web) and mobile apps (Flutter).

It's under active development, **compiled to native machine code** (when used for building mobile apps), inspired by modern features of other programming languages (mainly Java, JavaScript, C#) and **strongly typed**.

As already mentioned, Dart is a compiled language. That means, that your code isn't executed like you write it, but instead, a compiler parses + transform it (to machine code).

What are "Types"?

Not all programming languages are strongly typed - but Dart is. What does that mean though?

Every value you use in your program (e.g. some user input you're saving) **has a type** - it could be "text" for example. In Dart (and in pretty much all other programming languages), that would not be called "text" though but "**string**".

On the other hand, if you're working with the age of a user, you might be using a **number** without any decimal places - a so-called "**integer**" value. Numbers with decimal places are called "**doubles**" (or "float" - in other programming languages).

These are some basic types - Dart has way more than these basic types though. You will learn way more about these (and all the other important) types throughout the course.

At the moment, these are types you should keep in mind:

`String` => `"Hello"` *or* `'Hello'` => Text, has to be wrapped in quotes in your code (you can use double or single quotes, just be consistent once you made your choice).

`int` => `20` *or* `-54` => Numbers without decimal places

`double` => `-3.99` *or* `85.9421` => Numbers WITH decimal places

`num` => The "parent type" of double and int. You should rarely use it - be more specific about which type of value you need (int or double) if possible!

It's also important to keep in mind that EVERY value in Dart is an object. More on that can be found below (=> "*What does "Object-oriented" mean?*")

Variables & Functions

In your programs, you typically need to store some values. Not necessarily in a database or in a file but in memory. You might need to store some intermediate result, the input of a user before you process it or some information about your Flutter widget (e.g. "Should it currently be displayed on the screen?").

You store such information (= values) in so-called **variables**.

Variables are data containers - they have names and store values of any type.

For example:

```
var myAge = 30;
```

is a variable that stores an integer (`int`) of value `30` .

The `var` keyword tells Dart that `myAge` is a variable. Alternatively to var, you could also use the type name - in addition to informing Dart about the variable, you would then also "inform Dart" about the type of data stored in the variable:

```
int myAge = 30;
```

However, Dart has a feature called "**type inference**". This means, that Dart is pretty smart about inferring types of values. If you created the variable with `var` , Dart is still able to **infer** that `myAge` is of type `int` because you initialize the variable (i.e. you assign a value right from the start) with an integer value.

Because of that built-in type inference, it's considered a **good practice** to NOT explicitly define the type but instead use var when defining variables. Hence this snippet would be preferred:

```
var myAge = 30;
```

That changes if you create a variable without an initial value - then, you should inform Dart about which type of data you plan on saving in there:

```
int myAge;
myAge = 30;
```

Besides variables, another core feature of ANY programming language are "functions".

Functions allow you to "outsource" code into "re-usable code snippets". Here's an example:

```
var price1 = 9.99;
var price2 = 10.99;
var total = price1 + price2;

var numOfRounds1 = 10;
var numOfRounds2 = 45;
var totalRounds = numOfRounds1 + numOfRounds2;
```

In this example, we have the same logic of adding two numbers in two different places of our code. Instead of repeating ourselves, it would be great to put that logic into a function that we can call whenever as often as we want. Here's the changed snippet, using a function:

```
num addNumbers(num n1, num n2) { // use num as a type because it should work with int a
    return n1 + n2;
}

var price1 = 9.99;
var price2 = 10.99;
var total = addNumbers(price1, price2);

var numOfRounds1 = 10;
var numOfRounds2 = 45;
var totalRounds = addNumbers(numOfRounds1, numOfRounds2);
```

Of course, on first sight, this code now got longer and more complex. But the advantage is, that if you ever need to change your addition logic, you do it in **one place** => Inside of the function. You don't have to edit multiple places in code just to change the `+` into a `-` for example (if you wanted to do that).

Functions can take arguments (the data between the `( )`) which are basically variables, scoped to the function. "**Scoping**" means that you can use the variables only inside of the block statement (= function body, the code between the `{ }`) where you defined them.

In addition, functions can also `return` values - like the result of the addition in this example.

What does "Object-oriented" mean?

Dart is an object-oriented programming language - that means that **every value in Dart is an object**. Even a simple number.

What are “objects” though?

Objects are data structures - you find them in a lot of programming languages. In Dart, every value is an object, even primitive values like text (= String) or numbers (= Integers and Doubles). But you also have more complex built-in objects (e.g. Lists of data) and you can build your own objects.

You often build your own objects if you want to express more complex relations between data or if you want to encapsulate certain functionality in “one building block”.

Objects are created with the help of “**Classes**” because every object needs a **blueprint** (=> the class) based on which you can then create (“instantiate”) it.

Here’s an example class definition:

```
class Person {
    var name = 'Max';
    var age = 30;

    void greet() {
        print('Hi, I am ' + name + ' and I am ' + age.toString() + ' years old!';
    }
}
```

In this example, we define a `Person` class which has two class-level variables (also called “instance fields” or “**properties**”) and one class-level function (also called **“method"**).

As you can see, we also use types in classes - for both properties (variables) and methods (functions).

You can also see, that inside of the `greet` method, we can access the class properties `name` and `age` without issues (`age.toString()` is used to covert the integer value to a string whilst outputting it in a longer string).

The class only serves as a blueprint though! On its own, it does not give you an object! Instead, you can now create objects based on this class:

```
class Person {
    var name = 'Max';
    var age = 30;

    void greet() {
        print('Hi, I am ' + name + ' and I am ' + age.toString() + ' years old!';
    }
}

void main() {
    var myself = Person();
    print(myself.name); // use the . to access class properties & methods
}
```

As a side note: The `main` function is a special function in Dart - **it's the function which Dart will execute first**, when your app starts.
Inside of `main`, we then create a new object based on `Person` by using `Person()`. This process is called "instantiating the class", hence we create "an instance of Person".

The **type of** myself would then be Person because **classes always also act as types**!