Deep Dive: Position & Named Arguments

In the previous lecture, you learned about "**positional**" and "**named**" arguments / parameters.

In general, function parameters / arguments (the term is used interchangeably here) are a **key concept**.

You use arguments to pass values into a function. The function may then use these parameter values to work with them - e.g., to display them on the screen, use them in a calculation or send them to another function.

In Dart (and therefore Flutter, since it uses Dart), you have two kinds of parameters you can accept in functions:

- **Positional**: The position of an argument determines which parameter receives the value

```
void add(a, b) { // a & b are positional parameters
  print(a + b); // print() is a built-in function that will be explained later
}

add(5, 10); // 5 is used as a value for a, because it's the first argument; 10 is used as a value for b
```

- **Named**: The name of an argument determines which parameter receives the value

```
void add({a, b}) { // a & b are named parameters (because of the curly braces)
  print(a + b);
}

add(b: 5, a: 10); // 5 is used as a value for b, because it's assigned to that name; 10 is used as a value for a
```

Besides the different usage, there's **one very important difference** between positional and named arguments: By default, positional parameters are required and must not be omitted - on the other hand, named arguments are optional and can be omitted.

In the example above, when using **named parameters**, you **could** call `add()` like this:

```
add();
```

or

```
add(b: 5);
```

When using **positional parameters**, calling `add()` like this would be **invalid** and hence cause an **error**!

You can change these behaviors, though. You can make positional arguments **optional** and named arguments **required**.

Positional arguments can be made optional by wrapping them with square brackets (`[]`):

```
void add(a, [b]) { // b is optional
  print(a + b);
}
```

Once a parameter is optional, you can also assign a **default value** to it - this value would be used if no value is provided for the argument:

```
void add(a, [b = 5]) { // b is optional, 5 will be used as a default value
  print(a + b);
}
add(10); // b would still be 5 because it's not overwritten
add(10, 6); // here, b would be 6
```

Default values can also be assigned to named parameters - which are optional by default:

```
void add({a, b = 5}) { // b has a default value of 5
  print(a + b);
}

add(b: 10); // for b, 10 would be used instead of 5; a has no default value
   and would be "null" here => a special value type you'll learn about throughout
   this course
```

You can also make named parameters required by using the built-in `required` keyword:

```
void add({required a, required b}) { // a & b are no longer optional
  print(a + b);
}
```

You will, of course, see these different use-cases in action throughout the course.