Types

Effective Programming in Scala

Expressions Have Types

Consider the following program:

```
true && "false"
```

Expressions Have Types

Consider the following program:

```
true && "false"

        ^^^^^^
error: type mismatch;
 found    : String("false")
 required: Boolean
```

Types Classify Values

Integer numbers (1, 2, etc.) belong to the `Int` type.

Text values (`"Alice"`, `"Bob"`, etc.) belong to the `String` type.

And so on...

Another way to look at it is to say that a type defines a **set of possible values**.

For instance, the type `Boolean` has two possible values: `true` and `false`.

We will see that some types (e.g., `String`) have an unbounded number of possible values.

Predefined Types

The Scala language defines the following types:

Type	Description	Values
`Boolean`	Boolean	`true` and `false`
`Int`	32-bit signed integer	-2^{31} to $2^{31} - 1$
`Double`	64-bit floating point number	1.0, 3.14, etc.
`String`	Text	`"Alice"`, `"Bob"`, etc.

We will discover some other types later.

And we will see how to create new types!

Types Define Operations

The types define how the expressions can be combined by applying operations to them.

For this reason, operations are also called **members** of types.

For instance, the `&&` operation is available on the type `Boolean` and expects another `Boolean` value on its right-hand side. We can say that the type `Boolean` has a member named `&&`, which takes another `Boolean` value as parameter.

Type Error: Member Not Found

If you try to apply an operation to an expression whose type does not provide such an operation, it's an error:

```
true.combine(false)

     ^^^^^^^
error: value combine is not a member of Boolean
```

Type Error: Type Mismatch

If you try to apply an operation to an operand of a type incompatible with the type expected by the operation, it's a type mismatch error:

```
true && "false"

        ^^^^^^
error: type mismatch;
 found   : String("false")
 required: Boolean
```

Static Type Checking

The Scala compiler can check that programs are **well typed** before they are evaluated.

This ensures that some kind of errors can't happen at run-time.

Note that in worksheets, there is no distinction between compilation and evaluation, but you will see the difference between compilation errors and run-time errors when you will work on Scala projects outside of worksheets.

Types Are (Generally) Inferred

If we want, we can explicitly indicate the type of a definition:

```
val facade: Int = 5 * 3
```

This can sometimes improve readability.

However, in general the compiler is able to **infer** such types.

Summary

Types define the rules for combining expressions together.

Before a Scala program is executed, the compiler checks that it is well typed.

The compiler is usually able to infer the types of the definitions of a program, but you can add them explicitly to improve code readability.