COL100: Introduction to Computer Science

1.2: The functional model and Standard ML

A functional model of computation

An algorithm takes some input value(s) and computes the desired output value

→ An algorithm is a means of computing a function!

Primitives: some built-in sets and functions

Operations: applying functions, combining them to build new functions

Mathematical background:
Sets and functions

Sets

A *set* is a collection of things, in no specified order

- Set of all natural numbers: $\mathbb{N} = \{0, 1, 2, \ldots\}$
- Set of all integers: $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- Set of all real numbers: $\mathbb{R}$
- Set of all Boolean values: $\mathbb{B} = \{\text{true}, \text{false}\}$

Cartesian products

The *Cartesian product* of two sets A and B is the set $A \times B$ containing all pairs (a, b) with $a \in A$ and $b \in B$.

$$A = \{x, y\}$$

$$B = \{1, 2, 3\}$$

$$\begin{aligned} A \times B = \{&(x, 1), (x, 2), (x, 3), \\ &(y, 1), (y, 2), (y, 3)\} \end{aligned}$$

An n-ary Cartesian product $A_1 \times A_2 \times \cdots \times A_n$ contains n-tuples $(a_1, a_2, \ldots, a_n)$.

Functions

A *function* $f : X \to Y$ associates each value $x \in X$ to exactly one value $f(x) \in Y$.

- What about a function of the form $f(x, y, z)$? The domain is just $X \times Y \times Z$

We will also include partial functions (e.g. division : $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$):
f associates each $x \in X$ to *at most* one value $f(x) \in Y$.

Basics of Standard ML

Built-in sets / types

Usually we want to compute with elements of the sets $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{B}$.
In Standard ML, these correspond to *values* / *constants* of different *types*

- `2`, `0`, `~5`, … etc. of type `int`
- `0.0`, `9.8`, `3.14159`… etc. of type `real`
- `true` and `false` of type `bool`

Each value has a type… and only one type!

- `2 : int` but `2.0 : real`

Built-in functions

We have the basic functions you'd need to work with these values

- Arithmetic on integers: `+`, `-`, `*`, `div`, `mod`
- Arithmetic on reals: `+`, `-`, `*`, `/`
- Comparison relations: `=`, `<>`, `>`, `<`, `<=`, `>=`
- Logical operators: `andalso`, `orelse`, `not`

Strings

Pieces of text are represented by the `string` type

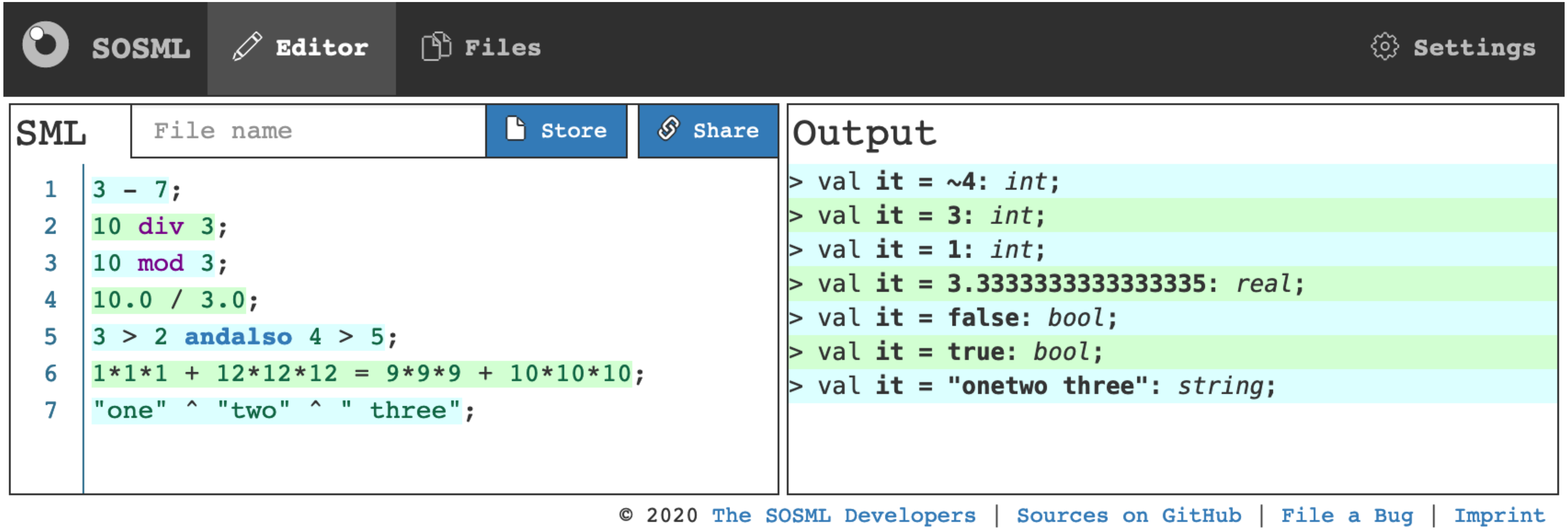
- e.g. `"Hello, world!"`, `"p@s$w0Rd"`, `"We, the People of India, having..."`

Built-in functions: `size : string -> int`, `^ : string * string -> string`

- `size("Hello")` evaluates to `5`
- `"COL" ^ "100"` evaluates to `"COL100"`

Try it yourself!

https://sosml.org/editor



```
3 - 7;
10 div 3;
10 mod 3;
10.0 / 3.0;
3 > 2 andalso 4 > 5;
1*1*1 + 12*12*12 = 9*9*9 + 10*10*10;
"one" ^ "two" ^ " three";
```

```
> val it = ~4: int;
> val it = 3: int;
> val it = 1: int;
> val it = 3.3333333333333335: real;
> val it = false: bool;
> val it = true: bool;
> val it = "onetwo three": string;
```

© 2020 The SOSML Developers | Sources on GitHub | File a Bug | Imprint

Variables

In mathematics, we can define a variable, say $x = 5$

We can do the same in SML:

```
val x = 5;
```

Unsurprisingly, now `x + 1` evaluates to `6`.

Be careful:

- `val x = 5;` binds the variable `x` to the value `5`
- `x = 7;` compares `x` to `7` and returns `false`

Functions

We may also want to define our own functions, e.g. *square* : $\mathbb{Z} \rightarrow \mathbb{Z}$,

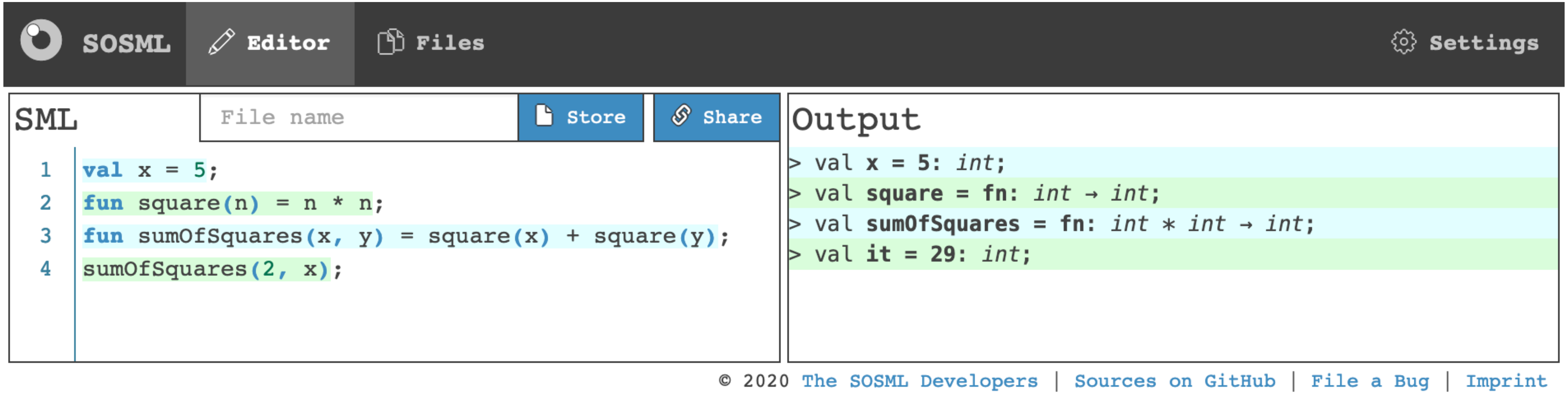
$$square(n) = n \times n$$

In SML,

```
fun square(n) = n * n;
```

The SML interpreter reports back `val square = fn: int -> int;`

Now we can write `square(4)` and get back `16`.



```
val x = 5;
fun square(n) = n * n;
fun sumOfSquares(x, y) = square(x) + square(y);
sumOfSquares(2, x);
```

```
> val x = 5: int;
> val square = fn: int → int;
> val sumOfSquares = fn: int ∗ int → int;
> val it = 29: int;
```

© 2020 The SOSML Developers | Sources on GitHub | File a Bug | Imprint

After this lecture

- Read Ch. 2.0.1, 2.0.2, 3.1, 3.2 of the notes.
- Using the comparison relations and logical operators, write a function `isTriangle : int * int * int -> bool` that returns `true` or `false` depending on whether the three given lengths can form a triangle.
- Think of some other simple mathematical functions you can implement in SML.