# Anonymous Functions

# Anonymous Functions

We already know that we can have values that are not bound to names. The integer 42, for example, can be entered at the toplevel without giving it a name:

```
42 ;;
```

```
- : int = 42
```

# Anonymous Functions

Or we can bind it to a name:

```
let x = 42 ;;
```

```
val x : int = 42
```

# Anonymous Functions

Similarly, OCaml functions do not have to have names; they may be *anonymous*. For example, here is an anonymous function that increments its input: `fun x -> x + 1`. Here, `fun` is a keyword indicating an anonymous function, `x` is the argument, and `->` separates the argument from the body. We now have two ways we could write an increment function:

```
let inc x = x + 1
let inc = fun x -> x + 1
```

```
val inc : int -> int = <fun>
```

# Anonymous Functions

Anonymous functions are also called *lambda expressions*, a term that comes from the *lambda calculus*, which is a mathematical model of computation in the same sense that Turing machines are a model of computation. In the lambda calculus, `fun x -> e` would be written $\lambda x.e$ The $\lambda$ denotes an anonymous function.

# Anonymous Functions

Anonymous functions are also called *lambda expressions*, a term that comes from the *lambda calculus*, which is a mathematical model of computation in the same sense that Turing machines are a model of computation. In the lambda calculus, `fun x -> e` would be written $\lambda x.e$ The $\lambda$ denotes an anonymous function.

→ **Higher-Order Programming**