

Objective Caml version 3.08.3

```
(* ===== *)
(* Einfache arithmetische und boolesche Ausdrücke *)
(* ===== *)

# 2 + 2;;

- : int = 4

# 3.14;;

- : float = 3.14

(* ----- *)
(* sqrt verlangt ein Argument vom Typ float. Eine automatische Typanpassung, *)
(* z.B. von int nach float, gibt es in O'Caml nicht. *)
(* ----- *)

# sqrt 2;;

Characters 5-6:
  sqrt 2;;
    ^
This expression has type int but is here used with type float

# sqrt 2.0;;

- : float = 1.41421356237309515

# sqrt 2.;;

- : float = 1.41421356237309515

(* ----- *)
(* Auch überladene Operatoren gibt es nicht in O'Caml. Stattdessen gibt es je *)
(* zwei Versionen der arithmetischen Operationen Addition, Subtraktion, Multi- *)
(* plikation und Division, nämlich eine für integers und eine für floats. *)
(* ----- *)

# sqrt 2. * sqrt 2.;;

Characters 0-7:
  sqrt 2. * sqrt 2.;;
  ^^^^^^^
This expression has type float but is here used with type int

# sqrt 2. *. sqrt 2.;;

- : float = 2.00000000000000044

# 1 / 2;;

- : int = 0

# 3 mod 2;;

- : int = 1

# 1. /. 2.;;

- : float = 0.5

(* ----- *)
(* Die booleschen Operatoren && und || sind NICHT STRIKT im zweiten Argument, *)
(* d.h. ihr zweites Argument wird nicht immer ausgewertet. Das sieht man an *)
(* Ausdrücken, bei denen die Auswertung des zweiten Arguments divergieren oder *)
(* zu einer exception führen würde. *)
(* ----- *)

# true;;

- : bool = true

# true && false;;

- : bool = false
```

```
# 1 = 0;;
- : bool = false

# 1/0 = 0;;
Exception: Division_by_zero.

# false && 1 / 0 = 1;;
- : bool = false

# true || 1 / 0 = 1;;
- : bool = true

(* ----- *)
(* Vergleichsoperatoren wie =, <, >, ... sind in O'Caml polymorph. Ihre beiden *)
(* Argumente müssen vom gleichen Typ 'a sein, wobei 'a ein beliebiger Typ ist. *)
(* ----- *)

# 1 < 2;;
- : bool = true

# 1. < 2.;;
- : bool = true

# 1 < 2.;;
Characters 4-6:
 1 < 2.;;
   ^^
This expression has type float but is here used with type int

# "abc" < "xyz";;
- : bool = true

# 'a' < 'A';;
- : bool = false

#
```