

Learn You a Haskell
for Great Good!





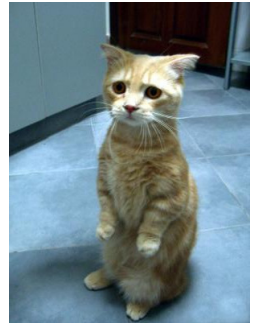
- **Nicht:** Anweisungen, die nacheinander ausgeführt werden.



- **Nicht:** Anweisungen, die nacheinander ausgeführt werden.
- „Besteht ein Programm die Typprüfung, so ist es korrekt.“

Quicksort in C

```
1  // von rosettacode.org
2  void quick_sort (int *a, int n) {
3      int i, j, p, t;
4      if (n < 2)
5          return;
6      p = a[n / 2];
7      for (i = 0, j = n - 1;; i++, j--) {
8          while (a[i] < p)
9              i++;
10         while (p < a[j])
11             j--;
12         if (i >= j)
13             break;
14         t = a[i];
15         a[i] = a[j];
16         a[j] = t;
17     }
18     quick_sort(a, i);
19     quick_sort(a + i, n - i);
20 }
```



Haskell, eine rein funktionale Sprache

```
1  qsort []          = []
2  qsort (x:xs) =
3      qsort kleinere ++ [x] ++ qsort groessere
4  where
5      kleinere  = [y | y <- xs, y <= x]
6      groessere = [y | y <- xs, y > x]
```



Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

fibs = 1 : 1 : ??

tail fibs = 1 : ??

+ 2 : ??

Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

fibs = 1 : 1 : 2 : ??

tail fibs = 1 : ??

+ 2 : ??

Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

fibs = 1 : 1 : 2 : ??

tail fibs = 1 : 2 : ??

+ 2 : ??

Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

fibs = 1 : 1 : 2 : ??

tail fibs = 1 : 2 : ??

+ 2 : 3 : ??

Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

fibs = 1 : 1 : 2 : 3 : ??

tail fibs = 1 : 2 : ??

+ 2 : 3 : ??

Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

fibs = 1 : 1 : 2 : 3 : ??

tail fibs = 1 : 2 : 3 : ??

+ 2 : 3 : ??

Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

fibs = 1 : 1 : 2 : 3 : ??

tail fibs = 1 : 2 : 3 : ??

+ 2 : 3 : 5 : ??

Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

fibs = 1 : 1 : 2 : 3 : 5 : 8 : 13 : ...

tail fibs = 1 : 2 : 3 : 5 : 8 : 13 : 21 : ...

+ 2 : 3 : 5 : 8 : 13 : 21 : 34 : ...

Haskell, eine rein funktionale Sprache

```
1 qsort [] = []
2 qsort (x:xs) =
3     qsort kleinere ++ [x] ++ qsort groessere
4     where
5     kleinere = [y | y <- xs, y <= x]
6     groessere = [y | y <- xs, y > x]
```

Die Fibonaccizahlen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
1 fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

♥ Statisches Typsystem mit Typerschließung ♥
rein funktional • nebenläufig • lazy • 7000⁺ Pakete