List Comprehensions

- Creating a list from another list, two decisions:
  - Is new list the same size as original, or smaller?
  - Are elements the same or related by some correspondence?

```
words = ["bear", "lion", "zebra", "python"]
w2 = [w for w in words if some_property(w)]
w3 = [f(w) for w in words]
w4 = [1 for w in words if some_property(w)]
```

- Once we have list can apply list functions
  - We have: len, sum, max, min
  - Can "invent" others by writing functions

List Comprehensions Again

- Transformative approach can scale differently
  - Functional programming: code generates and doesn't modify
  - Basis for (ultra) large scale mapreduce/Google coding

```
w = [expr for elt in list if bool_expr]
w = [f(w) for w in list if bool_expr(w)]
w = [list.count(x) for x in range(1,7)]
```

- Why are abstractions important?
  - Reason independently of concrete examples
    - Generalize from concrete examples
  - http://www.joelonsoftware.com/articles/LeakyAbstractions.html

Indefinite loop: while … *interactivity*

```
wrong = 0
while wrong < max_wrong:
    guess = raw_input()
    if not good_guess(guess):
        wrong += 1
    else:
        #process the guess here
```

- Suppose, for example, play http://www.hangman.no
  - What happens if you loop while True:
  - Break out of loop with break
  - See code in `GuessNumber.py`

Edsger Dijkstra

- Turing Award, 1972
- Algol-60 programming language
- Goto considered harmful
- Shortest path algorithm
- Structured programming

  "Program testing can show the presence of bugs, but never their absence"

For me, the first challenge for computing science is to discover how to maintain order in a finite, but very large, discrete universe that is intricately intertwined. And a second, but not less important challenge is how to mould what you have achieved in solving the first problem, into a teachable discipline: it does not suffice to hone your own intellect (that will join you in your grave), you must teach others how to hone theirs. The more you concentrate on these two challenges, the clearer you will see that they are only two sides of the same coin: teaching yourself is discovering what is teachable EWD 709