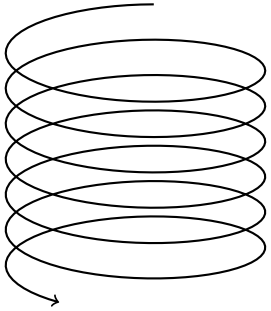
Nonprogrammer



... that programming is magic, that programmers can do things that normal people can't...

(Aaron, The Internet's Own Boy)

English

Normal people are good at communication. They have a language for that.

- nouns: "book"
- adjectives: "magic"
- numerals: 0 3.14 -9
- verbs: add combine
- ...

capital → word order → . ! ?

↑ syntax ↑ ↓ semantics ↓

Learn the meaning since childhood, understand intuitively.

Add 0 and 3.14 and -9!
Combine "magic" and "book". ✓

Add 0 and "book". ✗

Magic language Racket

"book"
"magic"
0 3.14 -9 } nouns contain *information*

+
string-append } verb says *what* to do with nouns

(verb the nouns)

↑ syntax ↑ ↓ semantics ↓

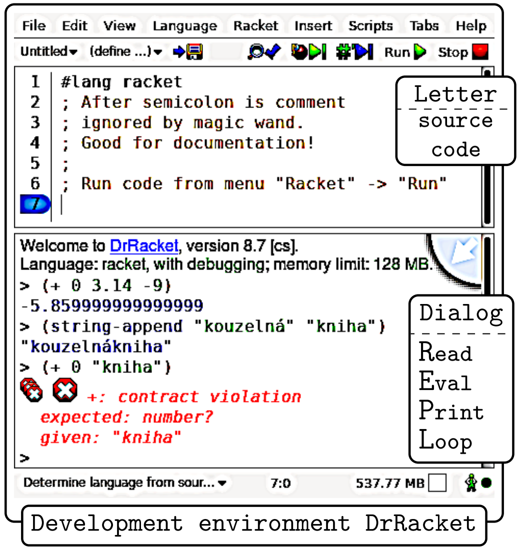
The meaning is given by the specification, but guessable!

```
1 (+ 0 3.14 -9)
2 (string-append "magic" "book")
```
✓

```
3 (+ 0 "book")
```
✗

Magic wand

... *evaluates* a magic sentence by making it into a magic word.



```
#lang racket
; After semicolon is comment
; ignored by magic wand.
; Good for documentation!
;
; Run code from menu "Racket" -> "Run"
```

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (+ 0 3.14 -9)
-5.859999999999999
> (string-append "kouzelná" "kniha")
"kouzelnákniha"
> (+ 0 "kniha")
+: contract violation
  expected: number?
  given: "kniha"
>
```

Letter
source
code

Dialog
Read
Eval
Print
Loop

Development environment DrRacket

nouns
primitive data types

numbers	*text*	*logic*
0 3.14 -9	"book" "3.14"	#t #f

verbs

+ - * / sqrt expt	substring	and
modulo round abs	string->number	or
sin max log atan	number->string	not

predicates
result of evaluation is #t or #f

= < > <= >= odd? | string=? | equal?

naming the information
definition of variables

```
4 (define number-1 3.14)
5 (define number-2 -9)
6 (define greater-1 (> number-1 number-2))
7 (define longer-1
8   (> (string-length "book")
9      (string-length "magic")))
```

new verb
definition of function

```
10 (define (longer-1? word-1 word-2)
11   (> (string-length word-1)
12      (string-length word-2)))
13 (longer-1? "book" "magic")
```

list
nouns connected into the single noun

```
14 (define nums (list 0 3.14 -9))
15 (equal? nums '(0 3.14 -9))
16 (equal? 0 (car nums))
17 (equal? '(3.14 -9) (cdr nums))
18 (equal? nums ; (cons 0 '(3.14 -9))
19          (cons (car nums) (cdr nums)))
```

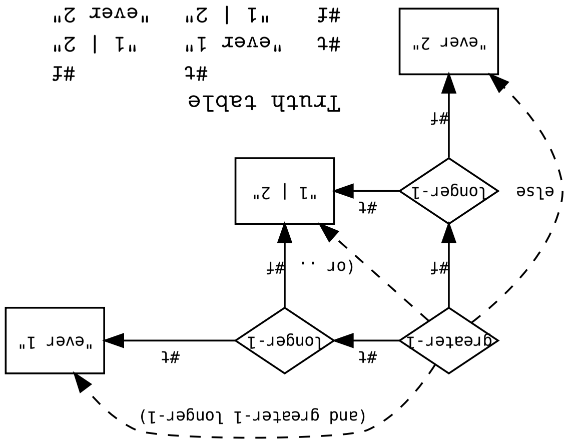
recursion
using function during its definition

```
20 (define (sum from upto) ; including
21   (if (= from upto)
22       upto ; base step / induction step
23       (+ from (sum (+ 1 from) upto))))
24 (sum 3 6)
```

Conditional expressions

```
25 (if greater-1
26     (if longer-1
27         "ever 1"
28         "1 | 2")
29     (if longer-1
30         "1 | 2"
31         "ever 2"))
```

```
32 (cond
33   ((and greater-1
34         longer-1)
35    "ever 1")
36   ((or greater-1
37        longer-1)
38    "1 | 2")
39   (else "ever 2"))
```



Truth table

	#t	#f
#t	"ever 1"	"1 | 2"
#f	"1 | 2"	"ever 2"

Results of evaluation

```
1 -5.859999999999999
2 "magicbook"

3 . . +: contract violation
    expected: number?
    given: "book"

13 #f

15 #t
16 #t
17 #t
18 #t

24 18

25 "1 | 2"

32 "1 | 2"
```

Magic reading

Language is one of the ways in which ideas can be shared. It is used in speaking and listening and writing and reading. The fact that the magic wand can execute sentences written in a language does not affect the ability to share thoughts in the language. However, the execution of a magic sentence with a magic wand is related to how one can work with ideas expressed in a magic language, because the magic wand does enough of the work.

Short mathematical idea

```
(define (Pythagorean-theorem a b c)
  ; a, b are ordinates, c is hypotenuse
  (= (* c c) (+ (* a a) (* b b))))
(define (compute-hypotenuse-for-ordinates a b)
  (sqrt (+ (* a a) (* b b))))
(define (compute-ordinate-for-hypotenuse-and-ordinate c a)
  (sqrt (- (* c c) (* a a))))
```

Some ideas about logic

```
; basic definitions and reverse (negation)
; of a single sentence
(define true #t)
(define (is-true? sentence)
  (if (equal? true sentence)
      true
      false))
(define false #f)
(define (is-false? sentence)
  (if (equal? false sentence)
      true
      false))
(define (not-holds-that sentence)
  (cond ((is-true? sentence) false)
        ((is-false? sentence) true)))

; possible combinations of two sentences
(define (hold-both? sentence-1 sentence-2)
  (if (is-true? sentence-1)
      (if (is-true? sentence-2)
          true
          false)
      false))
(define (first-holds-second-not?
         sentence-1
         sentence-2)
  (if (is-true? sentence-1)
      (if (is-false? sentence-2)
          true
          false)
      false))
(define (second-holds-first-not?
         sentence-1
         sentence-2)
  (if (is-true? sentence-2)
      (if (is-false? sentence-1)
          true
          false)
      false))
(define (neither-holds? sentence-1 sentence-2)
  (if (is-false? sentence-1)
      (if (is-false? sentence-2)
          true
          false)
      false))
(define (say-if-holds-that should-be sentence)
  (cond ((hold-both? should-be sentence) "holds")
        ((neither-holds? should-be sentence) "holds")
        (else "doesn't hold")))
```

```
; compound sentence of two sentences
(define (compound-with-and sentence-1 sentence-2)
  (cond
    ((hold-both? sentence-1 sentence-2) true)
    ((first-holds-second-not? sentence-1 sentence-2) false)
    ((second-holds-first-not? sentence-1 sentence-2) false)
    ((neither-holds? sentence-1 sentence-2) false)))
(define (compound-with-or sentence-1 sentence-2)
  (cond
    ((hold-both? sentence-1 sentence-2) true)
    ((first-holds-second-not? sentence-1 sentence-2) true)
    ((second-holds-first-not? sentence-1 sentence-2) true)
    ((neither-holds? sentence-1 sentence-2) false)))
(define (compound-with-exclusive-or sentence-1 sentence-2)
  (cond
    ((hold-both? sentence-1 sentence-2) false)
    ((first-holds-second-not? sentence-1 sentence-2) true)
    ((second-holds-first-not? sentence-1 sentence-2) true)
    ((neither-holds? sentence-1 sentence-2) false)))
(define (whenever sentence-1 sentence-2)
  (cond
    ((hold-both? sentence-1 sentence-2) true)
    ((first-holds-second-not? sentence-1 sentence-2) false)
    ((second-holds-first-not? sentence-1 sentence-2) true)
    ((neither-holds? sentence-1 sentence-2) true)))
(define (just-whenever sentence-1 sentence-2)
  (cond
    ((hold-both? sentence-1 sentence-2) true)
    ((first-holds-second-not? sentence-1 sentence-2) false)
    ((second-holds-first-not? sentence-1 sentence-2) false)
    ((neither-holds? sentence-1 sentence-2) true)))

; techical terms
(define conjunction compound-with-and)
(define disjunction compound-with-or)
(define negation not-holds-that)
(define implication whenever)
(define equivalence just-whenever)

; What is the negation of 'When it's raining it is wet'?
(define it-is-raining true)
(define it-is-wet true)
(say-if-holds-that
 (not-holds-that (whenever it-is-raining it-is-wet))
 (whenever (not-holds-that it-is-raining) it-is-wet))
 ; Whenever it isn't raining it's wet.
(say-if-holds-that
 (not-holds-that (whenever it-is-raining it-is-wet))
 (whenever it-is-raining (not-holds-that it-is-wet)))
 ; Whenever it's raining it isn't wet.
(say-if-holds-that
 (not-holds-that (whenever it-is-raining it-is-wet))
 (whenever (not-holds-that it-is-raining)
           (not-holds-that it-is-wet)))
 ; Whenever it isn't raining it isn't wet.
```