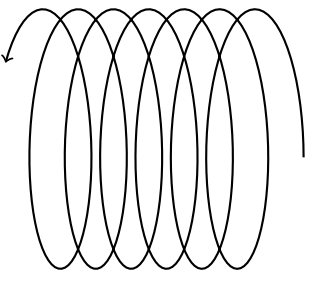


Nonprogrammer



(Aaron, The Internet's Own Boy)

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

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 child-hood, understand intuitively.

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

Add 0 and "book".



Results of evaluation

```

13 #f
18
20 #t
21 #t
22 #t
23 #t
25 "1 | 2"
32 "1 | 2"
    
```

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

```

1 -5.859999999999999999999
2 "magicbook"
    
```

Magic language Racket

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

+ verb says *what*
string-append 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")

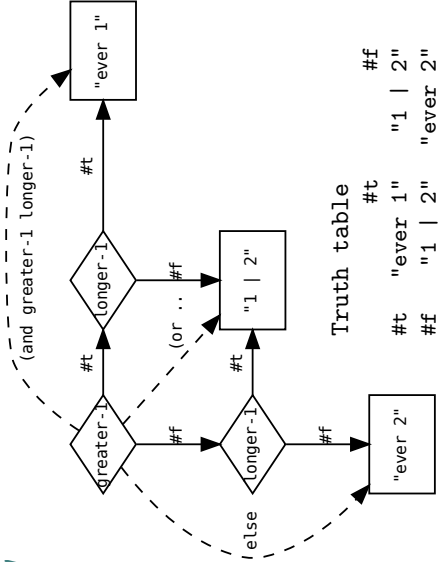
3 (+ 0 "book")



Conditional expressions

Truth table

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

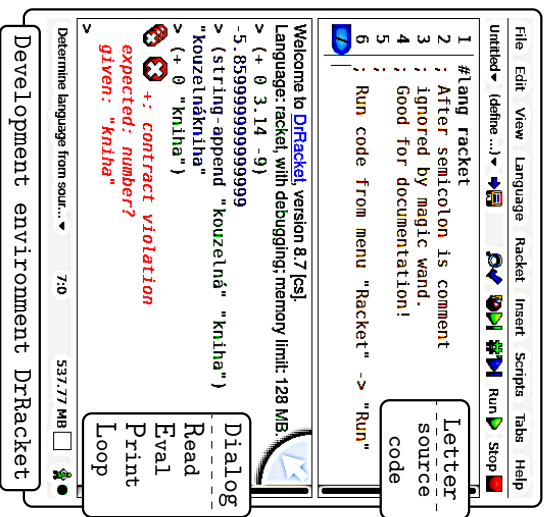


```

25 (if greater-1 (and greater-1
26 (if longer-1 "ever 1" "1"
27 longer-1 "ever 1" "1"
28 "1" "1"
29 "1" "1"
30 longer-1 "ever 2" "1"
31 "ever 2" "1"
32 (cond
33 ((and greater-1
34 longer-1
35 longer-1)
36 "ever 2" "1"
37 longer-1 "1"
38 "1" "1"
39 (else)
40 "ever 2" "1"
41 "ever 2" "1"
42 "ever 2" "1"
43 "ever 2" "1"
44 "ever 2" "1"
45 "ever 2" "1"
46 "ever 2" "1"
47 "ever 2" "1"
48 "ever 2" "1"
49 "ever 2" "1"
50 "ever 2" "1"
51 "ever 2" "1"
52 "ever 2" "1"
53 "ever 2" "1"
54 "ever 2" "1"
55 "ever 2" "1"
56 "ever 2" "1"
57 "ever 2" "1"
58 "ever 2" "1"
59 "ever 2" "1"
60 "ever 2" "1"
61 "ever 2" "1"
62 "ever 2" "1"
63 "ever 2" "1"
64 "ever 2" "1"
65 "ever 2" "1"
66 "ever 2" "1"
67 "ever 2" "1"
68 "ever 2" "1"
69 "ever 2" "1"
70 "ever 2" "1"
71 "ever 2" "1"
72 "ever 2" "1"
73 "ever 2" "1"
74 "ever 2" "1"
75 "ever 2" "1"
76 "ever 2" "1"
77 "ever 2" "1"
78 "ever 2" "1"
79 "ever 2" "1"
80 "ever 2" "1"
81 "ever 2" "1"
82 "ever 2" "1"
83 "ever 2" "1"
84 "ever 2" "1"
85 "ever 2" "1"
86 "ever 2" "1"
87 "ever 2" "1"
88 "ever 2" "1"
89 "ever 2" "1"
90 "ever 2" "1"
91 "ever 2" "1"
92 "ever 2" "1"
93 "ever 2" "1"
94 "ever 2" "1"
95 "ever 2" "1"
96 "ever 2" "1"
97 "ever 2" "1"
98 "ever 2" "1"
99 "ever 2" "1"
100 "ever 2" "1"
    
```

Magic wand

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



```

24 ((cons (cdr nums) (cdr nums)))
23 (equal? nums ; (cons 0 (3.14 -9))
22 (equal? (cdr nums) (cdr (cons 0 (3.14 -9)))
21 (equal? (car nums) (car (cons 0 (3.14 -9)))
20 ((- 4 1 3.14) 0)
19 (define nums (list 0 3.14 -9))
18 nouns connected into the single noun
17 list
16 nouns connected into the single noun
15 (9 8 edge-nums)
14 (define (edge-nums)
13 (define (upto n)
12 (define (upto n)
11 (define (upto n)
10 (define (upto n)
    
```

--- definition of function
--- recursive function

```

10 (define (word-length string)
11 (define (word-length string)
12 (define (word-length string)
13 (define (word-length string)
14 (define (word-length string)
15 (define (word-length string)
16 (define (word-length string)
17 (define (word-length string)
18 (define (word-length string)
19 (define (word-length string)
20 (define (word-length string)
21 (define (word-length string)
22 (define (word-length string)
23 (define (word-length string)
24 (define (word-length string)
    
```

--- primitive data types
--- nouns

nouns	verbs	logic
numbers	0 3.14 -9	"book" "3.14" #t #f
verbs	+ - * / sqrt expt modulo round abs sin max log atan	string->number 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")))
    
```

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)))
```

; technical 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.
```