Scheme

James Emery

Edit: 2/15/2013

Contents

1 Download

Scheme can be downloaded from the internet free. One such version is PLT Scheme, which is available for Window, Mac OSX, and Linux.

2 Running

To start scheme click icon in "PLT Scheme " in all programs window. Type functions into the lower window of Dr Scheme, or paste them in.

Type the function with parameters and hit return. For example to use the function square:

```
(square 13); produces 169
(square (square 13)) ; produces 28561
```

3 Comments

Start a comment with a semicolon.

4 Examples from Abelson and Sussman.

Square and Absolute Value Examples.

```
(define (square x) ( * x x))


(define (sum-of-squares a b)
 (+ (square a) (square b)))


(define (abs1 x)
 (cond
   (( < x 0) (- x) )
   (( = x 0) 0 )
   (( > x 0) x )
  )
)

(define (abs2 x)
 (cond ((< x 0) (- x))
       (else x)
 )
)

(define (abs3 x)
 (if (< x 0) (- x) x )
)
```

```
(and (< 3 10) (< 3 (- 10)))
```

Exercise 1.2, Sum of Squares

```
(define (f a b c)
 (if (and (<= a b) (<= a c)) (sum-of-squares b c)
  (if (and (<= b a) (<= b c)) (sum-of-squares a c) (sum-of-squares a b)
  )
 )
)
```

Exercise 1.3, Test Zero .

```
(define (p) (p))

(define (test x y)
    (if (= x 0)
        0
        y))

(test 0 (p))
```

Why does this lock up Dr scheme?

Newton's Method for a Square Root, page 20

```
(define (sqrt-iter guess x)
 (if (good-enough guess x)
    guess
    (sqrt-iter (improve guess x)
                    x)))

(define (improve  guess x)
```

```
  (average guess (/ x guess)))

(define (average x y)
 (/ (+ x y) 2))

(define (good-enough guess x)
     (< (abs (- (square guess) x)) .001))

(define (sqrt1 x)
  (sqrt-iter 1 x))
```

Number of Ways of Making Change, page 38

```
(define (count-change amount)
  (cc amount 5))

(define (cc amount kinds-of-coins)
 (cond ((= amount 0) 1)
   ((or (< amount 0) (= kinds-of-coins 0)) 0)
   (else (+ (cc (- amount
                   (first-denomination kinds-of-coins))
                     kinds-of-coins)
              (cc amount
                 (- kinds-of-coins 1))))))

(define (first-denomination kinds-of-coins)
  (cond ((= kinds-of-coins 1) 1)
        ((= kinds-of-coins 2) 5)
        ((= kinds-of-coins 3) 10)
        ((= kinds-of-coins 4) 25)
        ((= kinds-of-coins 5) 50)))

(count-change 100)
```

Fast Exponential, page 42

```
(define (fast-exp b n)
  (cond ((= n 0 ) 1 )
    ((even? n) (square (fast-exp b (/ n 2))))
    (else (* b (fast-exp b (- n 1))))))
```

Greatest Common Denominator page. 44

```
(define (gcd1 a b)
    (if (= b 0) a (gcd1 b (remainder a b))))
```

The Fermat Test for a Prime, Page 47,

Reference: Fermat's Theorem, in Griffin, page 89. Fermat's Theorem is the following. If a and m are relatively prime, then

$$a^{\phi(m)} = 1 \bmod(m),$$

where ϕ is the Euler ϕ function. $\phi(m)$ is the number of integers less than m, and relatively prime to m. If p is prime, then

$$\phi(p) = p - 1.$$

So

$$a^{p-1} = 1 \bmod(p)$$

or

$$a^p = a \bmod(p).$$

This is Fermat's little theorem.

Raise an Integer to a Power Mod(m), Page 44

The function expmod computes

$$b^e \mathrm{m}od(m).$$

```
(define (expmod b e m)
  (cond ((= e 0) 1 )
   ((even? e)
          (remainder (square (expmod b (/ e 2) m)) m))
   (else
      (remainder (* b (expmod b (- e 1) m ))
          m))))
```

Perform the Fermat Test for a Prime by Using Fermat's Little Theorem.

```
(define (fermat-test n)
 (define a (+ 2 (random (- n 2))))
 (= (expmod  a n n) a))
```

The following evaluation must return 47 if 97 is a prime, according to Fermat's little theorem.

```
(remainder (fast-exp 47 97) 97)
```

Fast Prime Determination, page 44

Perform the Fermat Test for a prime p by raising a set of random numbers to power p mod(p).

```
(define (fast-prime? n times)
  (cond ((= times 0) #t)
     ((fermat-test n)
       (fast-prime? n (- times 1 )))
       (else #f )))
```

5 Lambda

The Lambda Calculus was invented by the logician and mathematician Alonzo Church. Church was partially motivated by the mathematical program set out by David Hilbert to find proof and consistency in mathematics by a mechanical manipulation of symbols. Kurt Godel famously proved that Hilbert's program could not be accomplished.
Reference: See chapter 8 of **Simply Scheme**.

6 Bibliography

[1]Harvey Brian, Wright Matthew , **Simply Scheme**, The MIT Press, 1994.
[2]Abelson Harold, Sussman Gerald Jay, with Sussman Julie, **Structure and Interpretation of Computer Programs**, the MIT Press, 1985
[3]Griffin Harriet , **Elementary Theory of Numbers**, McGraw-Hill 1954.
[4]Emery James , **Number Theory**, (numbertheory.pdf).