

Table of Contents

- [1. Домашнее задание №1](#)
 - [1.1. 1. Определение дня недели по дате](#)
 - [1.1.1. алгоритм Сакамото](#)
 - [1.1.2. алгоритм Сакамото без string-ref](#)
 - [1.1.3. алгоритм с wikibooks](#)
 - [1.2. 2. Действительные корни квадратного уравнения](#)
 - [1.2.1. решение](#)
 - [1.2.2. тесты](#)
 - [1.3. 3. НОД, НОК и проверка числа на простоту](#)
 - [1.3.1. решение](#)
 - [1.3.2. тесты](#)

1. Домашнее задание №1

При выполнении заданий **не используйте** присваивание и циклы. Избегайте возврата логических значений из условных конструкций. Подготовьте примеры для демонстрации работы разработанных вами процедур.

1.1. 1. Определение дня недели по дате

Определите процедуру day-of-week, вычисляющую день недели по дате по григорианскому календарю. Воспользуйтесь алгоритмом, описанным в литературе. Пусть процедура принимает три формальных аргумента (день месяца, месяц и год в виде целых чисел) и возвращает целое число — номер дня в неделе (0 — воскресенье, 1 — понедельник, ... 6 — суббота).

Пример вызова процедуры:

```
(day-of-week 04 12 1975) ⇒ 4
(day-of-week 04 12 2006) ⇒ 1
(day-of-week 29 05 2013) ⇒ 3
```

1.1.1. алгоритм Сакамото

[ИСТОЧНИК](#)

```
(define (day-of-week-fixed-month day month year)
  (remainder (+ year
    (quotient year 4)
    (- (quotient year 100))
    (quotient year 400)
    (char->integer (string-ref "-bed=pen+mad." month)))
    day)
  7))

(define (day-of-week day month year)
  (if (< month 3)
    (day-of-week-fixed-month day month (- year 1))
    (day-of-week-fixed-month day month year)))
```

1. тесты

```
(define (test-day-of-week name)
  (display name)
  (newline)
  (display "04 12 1975 ")
  (display (day-of-week 04 12 1975))
  (newline)
  (display "04 12 2006 ")
  (display (day-of-week 04 12 2006))
  (newline)
  (display "29 05 2013 ")
  (display (day-of-week 29 05 2013))
  (newline)
  (display "01 01 1970 ")
  (display (day-of-week 01 01 1970))
  (newline)
  (display "02 01 1970 ")
  (display (day-of-week 02 01 1970))
  (newline)
  (display "03 01 1970 ")
  (display (day-of-week 03 01 1970))
  (newline)
  (display "04 01 1970 ")
  (display (day-of-week 04 01 1970))
  (newline)
  (display "05 01 1970 ")
  (display (day-of-week 05 01 1970))
  (newline)
  (display "06 01 1970 ")
  (display (day-of-week 06 01 1970))
  (newline)
  (newline))

(test-day-of-week "Sakamoto's:")
```

```
Sakamoto's:
04 12 1975 4
04 12 2006 1
29 05 2013 3
01 01 1970 4
02 01 1970 5
03 01 1970 6
04 01 1970 0
05 01 1970 1
06 01 1970 2
```

1.1.2. алгоритм Сакамото без string-ref

```
(define (day-of-week-fixed-month day month year)
  (remainder (+ year
    (quotient year 4)
    (- (quotient year 100))
    (quotient year 400))
    (or
      (and (= month 1) 0)
      (and (= month 2) 3)
      (and (= month 3) 2)
      (and (= month 4) 5)
```

```

        (and (= month 5) 0)
        (and (= month 6) 3)
        (and (= month 7) 5)
        (and (= month 8) 1)
        (and (= month 9) 4)
        (and (= month 10) 6)
        (and (= month 11) 2)
        (and (= month 12) 4))
    day)
  7))
(define (day-of-week day month year)
  (if (< month 3)
      (day-of-week-fixed-month day month (- year 1))
      (day-of-week-fixed-month day month year)))

```

1. тесты

```
(test-day-of-week "Sakamoto 2:")
```

```

Sakamoto 2:
04 12 1975 4
04 12 2006 1
29 05 2013 3
01 01 1970 4
02 01 1970 5
03 01 1970 6
04 01 1970 0
05 01 1970 1
06 01 1970 2

```

1.1.3. алгоритм с wikibooks

[ссылка](#)

```

(define (calc-day-of-week2 day month year)
  (remainder (+ day
    (quotient (* 31 month) 12)
    year
    (quotient year 4)
    (- (quotient year 100))
    (quotient year 400))
    7))

(define (day-of-week day month year)
  (if (or (= month 1) (= month 2))
      (calc-day-of-week2 day (+ month 10) (- year 1))
      (calc-day-of-week2 day (- month 2) year)))

```

1. тесты

```
(test-day-of-week "ru.wikibooks.org:")
```

```

ru.wikibooks.org:
04 12 1975 4

```

```

04 12 2006 1
29 05 2013 3
01 01 1970 4
02 01 1970 5
03 01 1970 6
04 01 1970 0
05 01 1970 1
06 01 1970 2

```

1.2. 2. Действительные корни квадратного уравнения

Определите процедуру, принимающую коэффициенты a , b и c квадратного уравнения вида $ax^2+bx+c=0$ и возвращающую список чисел — корней уравнения (один или два корня, или пустой список, если корней нет).

Указание: для формирования списка используйте функцию (`list ...`):

```

(list)      → ()
(list 10)   → (10)
(list 10 11) → (10 11)

```

1.2.1. решение

```

(define (D a b c)
  (- (* b b)
     (* 4 a c)))

(define (quadratic_equation_by_D a b D)
  (if (>= D 0)
      (if (> D 0)
          (list (/ (+ (- b) (sqrt D)) (* 2 a))
                (/ (- (- b) (sqrt D)) (* 2 a)))
          (list (/ (- b) (* 2 a))))
      (list)))

(define (quadratic_equation a b c)
  (quadratic_equation_by_D a b (D a b c)))

```

1.2.2. тесты

```

(display (quadratic_equation 2 5 -3)) ;; -3 1/2
(newline)
(display (quadratic_equation 4 21 5)) ;; -5 -1/4
(newline)
(display (quadratic_equation 4 -12 9)) ;; 3/2
(newline)
(display (quadratic_equation 1 2 17)) ;; нет корней
(newline)

```

```

(1/2 -3)
(-1/4 -5)
(3/2)
()

```

1.3. 3. НОД, НОК и проверка числа на простоту

Определите:

- Процедуру (`my-gcd a b`), возвращающую наибольший общий делитель чисел `a` и `b`. Поведение вашей процедуры должно быть идентично поведению встроенной процедуры `gcd`.
- Процедуру (`my-lcm a b`), возвращающую наименьшее общее кратное чисел `a` и `b`. Используйте процедуру `my-gcd`, определенную вами ранее. Поведение вашей процедуры должно быть идентично поведению встроенной процедуры `lcm`.
- Процедуру (`prime? n`), выполняющую проверку числа `n` на простоту и возвращающую `#t`, если число простое и `#f` в противном случае.
- Примеры вызова процедур:

```
(my-gcd 3542 2464) ⇒ 154
(my-lcm 3 4)      ⇒ 12
(prime? 11)       ⇒ #t
(prime? 12)       ⇒ #f
```

1.3.1. решение

```
(define (my-gcd a b)
  (if (= b 0)
      a
      (my-gcd b (remainder a b))))

(define (my-lcm a b)
  (quotient (* a b) (my-gcd a b)))

(define (recursive-prime-test n i)
  (or (> (* i i) n)
      (and
        (> (remainder n i) 0)
        (> (remainder n (+ i 2)) 0)
        (recursive-prime-test n (+ i 6)))))

(define (prime? n)
  (or (= n 2)
      (= n 3)
      (and (>= n 5)
            (> (remainder n 2) 0)
            (> (remainder n 3) 0)
            (recursive-prime-test n 5))))
```

1.3.2. тесты

```
(display (prime? 1))
(newline)
(display (prime? 2))
(newline)
(display (prime? 3))
(newline)
(display (prime? 4))
(newline)
(display (prime? 5))
(newline)
(display (prime? 6))
```

```
(newline)
(display (prime? 7))
(newline)
(display (prime? 8))
(newline)
(display (prime? 13))
(newline)
```

```
#f
#t
#t
#f
#t
#f
#t
#f
#t
```

Author: Starovoytov Alexandr
Created: 2021-12-11 Sat 17:32