CSE1729 - Introduction to Programming | February 22, 2017

Laboratory Assignment 5

Objectives

- Work with higher order functions
- Work with the `plot` package

Activities

1. Let f and g be two functions, mapping integers to integers. Define a function dominate so that `(dominate f g)` outputs the smallest positive integer k for which $f(k) > g(k)$. Thus, if $f(1) > g(1)$, `(dominate f g)` should output 1. If $f(1) \leq g(1)$ but $f(2) > g(2)$, then `(dominate f g)` should output 2, etc.

Solution:

```
(define (dominate f g)
  (define (dominate-iter n)
    (if (> (f n)(g n)) n (dominate-iter (+ n 1))))
  (dominate-iter 1))

(define (times2 num) (* 2 num ))

(dominate times2 sqrt)
```

2. **[SICP Exercise 1.41]**
Define a procedure double that takes a procedure of one argument as an argument and returns a procedure that applies the original procedure twice. For example, if `inc` is a procedure that adds 1 to its argument, then `(double inc)` should be a procedure that adds 2. What value is returned by `(((double (double double)) inc) 5)`?

Solution:

```
(define (double f)
  (lambda (x) (f (f x))))
(define (inc x) (+ x 1))

((double inc) 1)
```

3. So far this semester, we have written several functions which find the n^{th} number in a sequence which satisfies some property (e.g. the n^{th} even number). It would be helpful if we wrote a higher order function which could take a function which generated the `sequence` and a function which `test`ed for the property in which we are interested as well as the integer n and returned the n^{th} value in that `sequence` which satisfied the property (i.e. the `test` function returns true when passed that value).

(a) Write a function, named `find`, which takes three parameters (`sequence`, `test`, and `n`) and returns the n^{th} value in `sequence` for which the `test` function returns true (`#t`) when passed a value in `sequence`.

Solution:

```
(define (find sequence test n)
  (define (find-aux x found)
    (let* ((fx (sequence x))
           (satisfies-test (test fx)))
    (cond ((and satisfies-test
                (= (+ found 1) n))
           fx)
           (satisfies-test (find-aux (+ x 1) (+ found 1)))
           (else (find-aux (+ x 1) found)))))
  (find-aux 1 0))
```

(b) Test your program with by finding the 5^{th} of even number and the 5^{th} of odd number. Verify your results with the solutions to the problem sets and in the lecture slides.

Solution:

```
(define (even x) (= (modulo x 2) 0))
(define (odd x) (not (even x)))
(define (id x) x)
(find id even 5)
(find id odd 5)
```

(c) A Fibonacci prime is a Fibonacci number which is prime. Use your higher-order function to find the 5^{th} Fibonacci prime.

Solution:

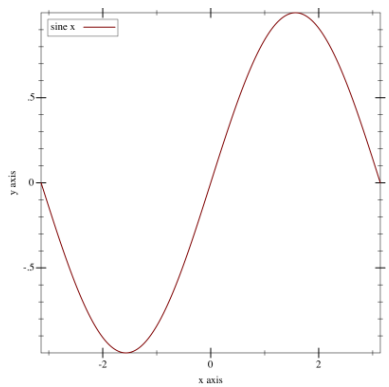
```
(define (fib x)
  (cond ((< x 2) 1)
        (else (+ (fib (- x 1)) (fib (- x 2))))))
(define (divides a b) (= (modulo b a) 0))
(define (smooth n k)
  (and (>= k 2)
       (or (divides k n)
           (smooth n (- k 1)))))
(define (isprime p) (and (> p 1) (not (smooth p (floor (sqrt p))))))
(find fib isprime 5)
```

4. Racket also provides a graphing package, named `plot`, with which you can graph the functions you write in SCHEME. With Racket 5.2.1, change your language to "Use the language declared in the source" and add `#lang racket` as the first line in your Definitions window. To plot a function, start with the following two instructions which will import the plotting functions and create graphs in a new window.

```
(require plot)
```

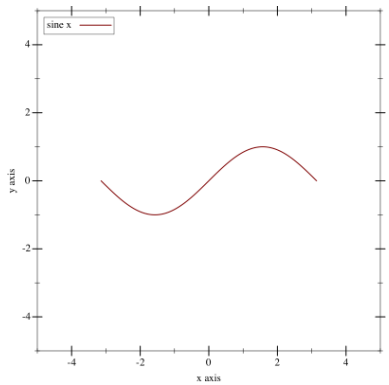
Next you can graph a function by using `plot`:

```
(plot (function sin (- pi) pi #:label "sine␣x"))
```



where `(- pi)` and `pi` determine the x-axis bounds. Additional plot keywords can be used to specify additional features of the plotted functions such as a legend and x and y bounds of the graph (separate from the bounds of the function to be graphed):

```
(plot (function sin (- pi) pi #:label "sine␣x")
      #:x-min -5 #:x-max 5 #:y-min -5 #:y-max 5)
```

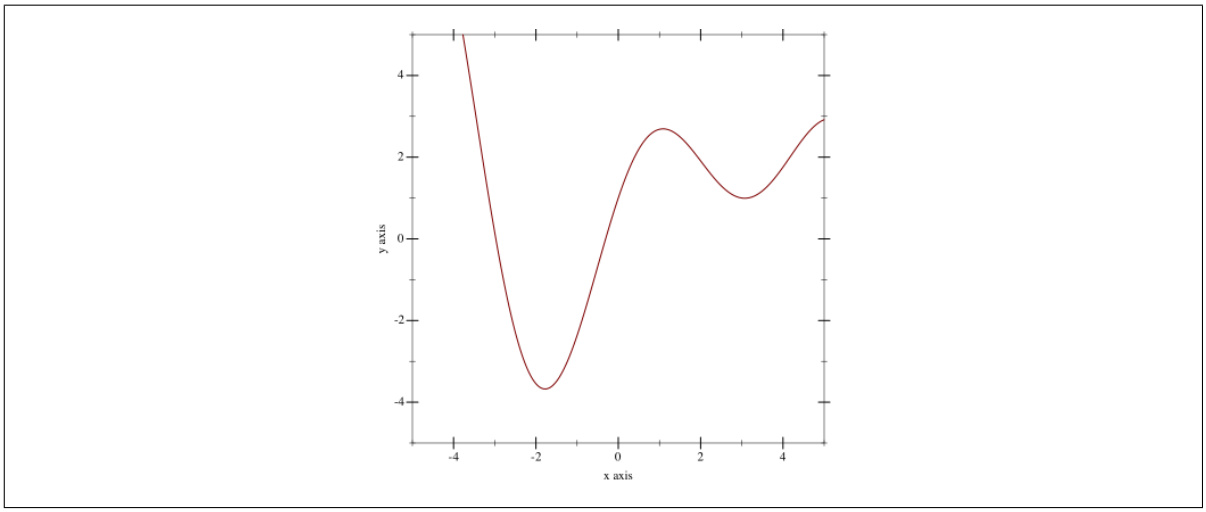


Plot the following function in the interval $[-5, 5]$, using Racket's `plot` capabilities:

```
(define (f x) (+ (* (sin x) (- 3 x)) 1))
```

Solution:

```
(plot (function f) #:x-min -5 #:x-max 5 #:y-min -5 #:y-max 5)
```


4
2
0
-2
-4
y axis
-4
-2
0
2
4
x axis