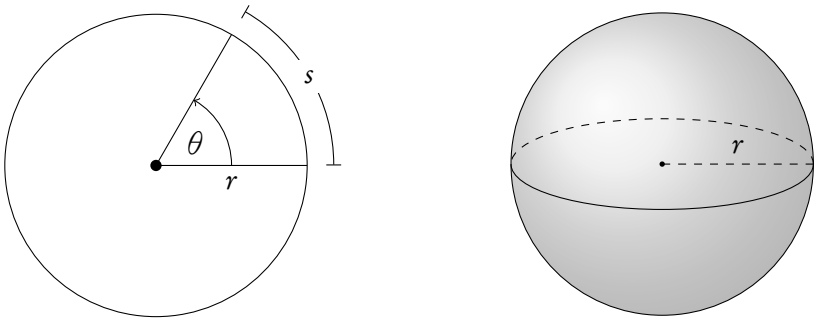
CSE 1729 - Introduction to Principles of Programming — October 6, 2020

Prelim 1 review Solutions

1. A circle is a shape consisting of all points in a plane that are a given distance, r, from a given point, the center. Likewise, a sphere is defined mathematically as the set of points that are all at the same distance r from a given point, but in a three-dimensional space.



(a) (2 points) Define a SCHEME function that calculates the area of a circle with radius r. Name your function `(circle-area r)` and use the well-known formula πr^2. Be sure to include definitions for any variables or helper functions necessary.

Solution:

```
(define pi 3.14159)
(define (circle-area r)
  (* pi r r))
```

(b) (2 points) The surface area of a sphere is given by the equation $S = 4\pi r^2$. Define a SCHEME function for the surface area of a sphere, named `(S r)`, where r is the radius of the sphere.

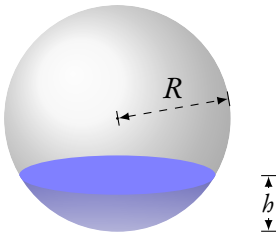
Solution:

```
(define (S r)
  (* 4 (circle-area r)))
```

(c) (2 points) The volume of a sphere is given by the equation $V = \frac{4}{3}\pi r^3$. Define a SCHEME function, named `(v r)` which computes the volume of a sphere, *in terms of the function for the surface area.* That is, your function for the volume of a sphere must call, and use, the function that computes the surface area of a sphere.

Solution:

```
(define (v r)
  (* r (S r) (/ 1 3)))
```



(d) (2 points) In geometry, a spherical cap is a portion of a sphere cut off by a plane. If the plane passes through the center of the sphere, so that the height of the cap is equal to the radius of the sphere, the spherical cap is called a hemisphere. The area of the curved surface of a spherical cap is given by the equation $A = 2\pi r h$. Define a SCHEME function, named `(A r h)` which computes the surface area of the curved portion of a spherical cap with radius r and height h.

Solution:

```
(define (A r h)
  (* 2 pi r h))
```

(e) (2 points) The volume of the spherical cap can be calculated using the following equation, $V = \frac{\pi h^2}{3}(3r - h)$. Define a SCHEME function, named `(V r h)` to compute the volume of a spherical cap with radius r and height h.

Solution:

```
(define (v r h)
  (* (/ (* pi h h) 3)
     (+ r r r (- h))))
```

2. The Bakhshali manuscript is a mathematical text written on birch bark that was found in 1881 in the village of Bakhshali in present-day Pakistan. It is the oldest known Indian manuscript in mathematics, with portions dated to AD 224-383. The Bakhshali method for finding an approximation to a square root was described in this manuscript. It is equivalent to two iterations of the Babylonian method beginning with x_0. Thus, the algorithm is *quartically* convergent, which means that the number of correct digits of the approximation roughly quadruples with each iteration. The original presentation, using modern notation, is as follows: To calculate $\sqrt{S}$, let x_0^2 be the initial approximation to S. Then, successively iterate as:

$$a_n = \frac{S - x_n^2}{2x_n},$$

$$x_{n+1} = (x_n + a_n) - \frac{a_n^2}{2(x_n + a_n)}.$$

Give a SCHEME function that computes roots using the Bakhshali method. To make this precise, you need to specify a "stopping criterion" (because the method itself yields an infinite sequence of values $x_0, x_1, \ldots$ that converge to the square root). One natural criterion is that the difference $|x_n^2 - S| \leq$

0.0001.)
Note: the value a_n and even the term $(x_n + a_n)$ are used several times in this computation. Using a `let` form could save you some ink (or graphite). You may use $\frac{S}{2}$ as an initial "guess" for $\sqrt{S}$. In that case, $x_0 = \frac{S}{2}$.

Solution:

```
(define (bakhshali S)
  (define (square x) (* x x))
  (define (b-aux guess error)
    (if (< (abs (- (square guess) S)) error) guess
        (let* ((a (/ (- S (square guess))
                     (* 2 guess)))
               (term (+ guess a)))
          (b-aux (- term
                    (/ (square a)
                       (* 2 term)))
                 error))))

   (b-aux (/ S 2) (/ 1 100000)))
```

3. Gregory's Series is an infinite Taylor Series expansion for the inverse tangent function. The series is,

$$\int_0^x \frac{du}{1+u^2} = \arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$

Write a SCHEME function `(gregory x k)` that computes an approximation for arctan x containing the first $k+1$ terms of the series above, i.e., it computes

$$\arctan(x) \approx \sum_{i=0}^{k} (-1)^i \frac{x^{(2i+1)}}{2i+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots \pm \frac{x^{2k+1}}{2k+1}$$

Solution:

```
(define (gregory x k)
  (if (= k 0)
      x
      (let ((y (+ (* 2 k) 1)))
        (+ (* (pow -1 k) (/ (pow x y) y)) (gregory x (- k 1))))))
```

4. The Gauss-Legendre algorithm is an algorithm to compute the digits of π. It is notable for being rapidly convergent, with only 25 iterations producing 45 million correct digits of π. The algorithm starts with the following initial values for four variables:

$$a_0 = 1 \qquad b_0 = \frac{1}{\sqrt{2}} \qquad t_0 = \frac{1}{4} \qquad p_0 = 1$$

Next, the following instructions are repeated until the difference of a_n and b_n are within the desired accuracy:

$$a_{n+1} = \frac{a_n + b_n}{2} \qquad b_{n+1} = \sqrt{a_n b_n}$$
$$t_{n+1} = t_n - p_n(a_n - a_{n+1})^2 \qquad p_{n+1} = 2p_n$$

Then, π is approximated by $\pi \approx \frac{(a_{n+1}+b_{n+1})^2}{4t_{n+1}}$

Define a SCHEME function, named `(gauss-legendre tol)`, that takes one parameter representing the tolerance (i.e. the desired accuracy) and computes an approximation to π using the Gauss-Legendre algorithm. You may once again find that a helper function makes this easier. If you do, make sure it gets hidden within the implementation of `gauss-legendre`.

Solution:

```
(define (gauss-legendre tol)
  (define (square x) (* x x))
  (define (gl-help a b t p)
    (let  ((new-a (/ (+ a b) 2))
           (new-b (sqrt (* a b)))
           (new-t (- t (* p (square (- a (/ (+ a b) 2))))))
           (new-p (* 2 p)))
      (if (< (abs (- new-a new-b)) tol)
          (/ (square (+ new-a new-b))
             (* 4 new-t))
          (gl-help new-a new-b new-t new-p))))
  (gl-help 1 (/ 1 (sqrt 2)) (/ 1 4) 1))
```

5. **SICP Exercise 1.44** - The idea of smoothing a function is an important concept in signal processing. If f is a function and dx is some small number, then the smoothed version of f is the function whose value at a point x is the average of $f(x-dx)$, $f(x)$, and $f(x+dx)$.

   (a) Write a SCHEME procedure, named `(smooth f dx)`, that takes as input a procedure that computes f and returns a procedure that computes the smoothed f.

   Solution:

```
(define (smooth f dx)
  (lambda (x) (/ (+ (f (- x dx))
                    (f x)
                    (f (+ x dx)))
                 3)))
```

(b) It is sometimes valuable to repeatedly smooth a function (that is, smooth the smoothed function, and so on) to obtain the *n-fold smoothed* function. Write a SCHEME procedure, named `(n-fold-smooth f dx n)`, to generate the n-fold smoothed function of any given function using `smooth` and `repeated` from SICP Exercise 1.43 and this week's lab assignment.

Remark 1. *In mathematics and computer science,* ***currying*** *is the technique of translating the evaluation of a function that takes multiple arguments into evaluating a sequence of functions, each with a single argument. You may find it helpful to define a curried function to use the same dx for each iteration of* `smooth`.

```
(define (smoother dx) (lambda (f) (smooth f dx)))
```

Solution:

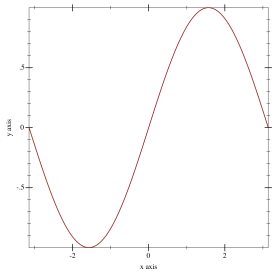
```
(define (n-fold-smooth f dx n)
  (define smoothing (smoother 0.00025))
  ((repeated smoothing n) f))
```

Try This at Home! Once you have completed the deriviative (`der`) and n^{th} derivative (`nth-deriv`) functions for the problem set, you can see these functions and your smoothing function in action. In a separate Racket file, you can use the Racket language to plot the sin function. Change the language to "Determine language from source" and place the following at the top of your new file:

```
#lang racket
(require plot)
```

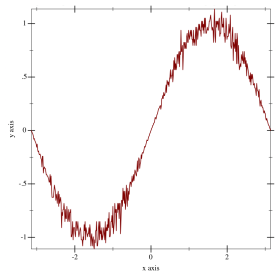
Now, you can plot the sin function:

```
(plot (function sin (- pi) pi))
```



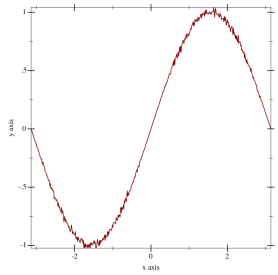
Next, copy over your derivative function solutions from the problem set to plot the n^{th} derivative of the sin function.

```
(define fourth-deriv-sin ((nth-deriv 4 0.00025) sin))
(plot (function fourth-deriv-sin (- pi) pi))
```



You should see something like this. Perhaps not what we expected. You can use the `smooth` function to reduce the noise, but it may not be enough.

```
(define (smoother dx) (lambda (f) (smooth f dx)))
(define smoothed ((smoother 0.00025) fourth-deriv-sin))
(plot (function smoothed (- pi) pi))
```



We can use the `n-fold-smooth` function to smooth out the fourth derivative of the sin function even further.

```
(plot (function (n-fold-smooth fourth-deriv-sin 0.00025 3) (- pi) pi))
```