Scheme

**Research 1**

Read the following chapter to get some context of functional programming in relation to other paradigms (especially section 1.3). [Chapter from ai book](https://www.cs.unm.edu/~luger/ai-final/supplement-ch1final.pdf)

Technical aspects of Racket compiler: <http://docs.racket-lang.org/guide/performance.html?q>=

**Reading Material 1**

Install Dr. Racket (previously known as Dr. Scheme) or any version of scheme you wish in your computer before reading.

<https://racket-lang.org/>

Read and make notes of the following links, tests some of the code examples to get used to the syntax.

* [Numbers, Expressions, Simple Programs](http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-5.html#node_chap_2).
* [Programs are Function Plus Variable Definitions](http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-6.html#node_chap_3).
* [Conditional Expressions and Functions](http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-7.html#node_chap_4).
* Read the Functions\_are\_Values section in the documentation of Racket located at Racket/doc/quick/index.html, you can also access it through the Dr. Racket help.

Do a small program that converts from Celsius to Fahrenheit.

**During class:**

* What is functional programming?
* What are some examples of languages used for functional programming?
* Where does scheme come from?
* How does scheme relate to the other languages you have seen so far?
* Is functional programming and parallelism compatible?
* Why has the functional programming paradigm become popular recently?
* How does Scheme compiler work?
* What is it optimized to do?
* **Lambda Syntax** https://docs.racket-lang.org/guide/lambda.html
* **(implement “**simple functions and lambda sintax.rkt” and “lambda.rkt”)
* **Cheat Sheet** [/Racket/doc/racket-cheat/index.html](file:///C:/Program%20Files/Racket/doc/racket-cheat/index.html)

**Lab 1**

After doing this lab you should be able to understand the basic structure of scheme and how to use its basic operators.

Define the function triangle-area which receives the base and the height and returns the area of a triangle.

Define the functions a, b, and c that represent each of the following numerical statement respectively

* n2 + 10
* (1/2)\*n2 + 20
* 2 - (1/n)

Single variable quadratic equations have the following form:

**a\*x2 + b\*x + c = 0**

where a b and c can be replaced by regular values. E.g.:

**2\*x2 + 4\*x + 2 = 0**

Or

**(-1) \* x2 + 0\*x + (-1) = 0**

The number of possible solutions of a quadratic equation depends on the values of a, b, and c. The possible number of solutions are dictated by the following rules (We assume that x can’t be 0).  
  
   two solutions if b2 > 4 · a · c,  
  
   one solution if b2 = 4 · a · c, and  
  
   no solution if b2 < 4 · a · c.  
  
Define the function solutions that receives a, b, and c and returns the number of possible solutions for a quadratic equation.

For example:

(solutions 1 0 -1) = 2  
(solutions 2 4 2) = 1

**Reading Material 2**

Read the following links to understand how lists work, and how recursion is used in scheme:

* [Tail Recursion](http://c2.com/cgi/wiki?TailRecursion).
* [Recursion in Scheme](http://www.shido.info/lisp/scheme7_e.html).
* [Compound Data, Part 2: Lists](http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-13.html#node_chap_9).
* [More on Processing Lists](http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-14.html#node_chap_10).
* [Natural Numbers](http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-15.html#node_chap_11).

With what you read you should at least be able to solve the following, if you can’t solve at least this, then read again and write down your doubts.

Define the function addition that performs the following operation:

Make 2 implementations, the first with head recursion and then second with tail recursion.

What is the difference between both of these recursions?

Define the function multiplications, which receives a list with n elements and multiplies all of them. E.g.:

mults((cons 1(cons 3(cons 4 (cons 2))))) should return 24.

**During class:**

* Explain the difference between head and tail. **Context in Memory**
* Factorial example of head recursion.
* Factorial example of tail recursion. (**implement** recursion examples.rkt)
* Calculate pi.
* Lists in scheme (**implement** list examples.rkt)
* Parallelism in scheme <http://docs.racket-lang.org/guide/parallelism.html?q>=

**Lab 2**

After doing this lab you should be able to manipulate lists and use both tail and head recursion

Define the functions power-head and power-tail. The functions much calculate the power of the given number. The difference is that power-head should use **head recursion** and power-tail should use **tail recursion**.

> (power-head 4 3) should return 32

> (power-tail 4 3) should return 32

Define the function *third* which returns the third element of a list.

> (third (cons 1(cons 2 (cons 3 (cons 4 (cons 5 empty)))))) should return 3

Define function just-two? Which returns true if a list contains only two elements

> (just-two? (cons 1 empty)) #f

> (just-two? (cons 1 (cons 4 empty))) #t

Define the function *how-many-x?* that receives a list and should return the number of elements that match *x*

> (define list (cons 1(cons 2 (cons 3 (cons 4 (cons 3 empty))))))

> (how-many-x? list 3) 2

Define the function *all-x?* that returns true if every element of the list is *x*

Define the function get which receives a list and a position and returns the value in that position of the list.

> (define list (cons 1(cons 2 (cons 3 (cons 4 (cons 3 empty))))))

> (get list 2) 2

> (get list 5) 3

Define the function difference that receives 2 lists A and B. The function must return a new list that contains the elements in A that re not present in B. E.g.:

> (difference '(12 44 55 77 66 1 2 3 4) '(1 2 3))

'(4 66 77 55 44 12)

Define the function append that receives 2 lists A and B. The function return only 1 list with the elements of A followed by the elements of B.

> (append '(a b c d) '(e f g))   
(a b c d e f g)   
  
> (append '( ) '(a b c))  
(a b c)

Define the function invert which inverts the order of the elements in a list

> (invert '(a b c d))   
(d c b a)   
  
> (invert '( ))  
( )

Define the function sign that receives a list of numbers and returns a list of 1 or -1 depending on whether each number is greater or lesser than 0.

> (sign '(2 -4 -6))  
  (1 -1 -1)

Define the function negatives that receives a list of positive numbers and returns a list with the corresponding negative numbers.  
 > (negatives '(2 -4 6))  
  (-2 -4 -6)

**Reading Material 3**

Read the following link to understand [Lists in lists](http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-19.html#node_sec_14.3) and [trees](https://www.cs.berkeley.edu/~bh/ssch18/trees.html) . If you still have doubts about the list and recursion here is a link with examples that can help you out [Recursion examples](http://www.cs.rpi.edu/courses/fall00/ai/scheme/reference/schintro-v14/schintro_46.html).

* Represent a 3 level tree with arity 2 (number of children per node) as a list.
* Count the number of elements in the tree.

If you want to see some of the reasons why functional languages became so very popular, here are two more links which might provide you with a good starting point [Quick sort](http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-32.html#node_sec_25.2) , [Lambda](https://www.cs.berkeley.edu/~bh/ssch9/lambda.html) , and [let](https://mitpress.mit.edu/sicp/full-text/book/book-Z-H-12.html#%_sec_1.3.2)

**During class**

* Represent a tree as a list. (Tygarian Family)
* Parse the tree using mutual recursion.
* Benchmark of functional languages:

<https://en.wikipedia.org/wiki/Comparison_of_functional_programming_languages>

**Lab 3**

After doing this lab you should be able to work with deep lists and represent trees

Define function deep-all-x? which receives a list containing other lists (deep list) and an element x. The function returns true if every single element in the list of lists is x. Otherwise it returns false.

Define function deep-reverse which receives a deep list. The function returns a list of lists with its elements in reverse order.   
(deep-reverse '(a (b (c d)) e (f g)))   
   > ((g f) e ((d c) b) a)

Define function flatten that receives a deep list and return a list containing all the elements in a single 1 level list

(flatten '(a (b (c d)) e (f g)))   
   > (a b c d e f g)

Define the function count-levels which counts the max depth of a tree

(count-leves '(a

(b

(c) (d))

(e

(f) (g))))

> 3

Define the function count-arity which counts the max number of children a single node of the tree has

(count-arity '(a

(b

(c) (d))

(e

(f) (g) (h) (i))))

> 4

**Parallel Scheme:**

Map and Places functions (implement map function)

Parallel example.rtk

**Cool Examples:**

**Genetic algorithm in racket:** <https://github.com/vollmerm/racket-ga>