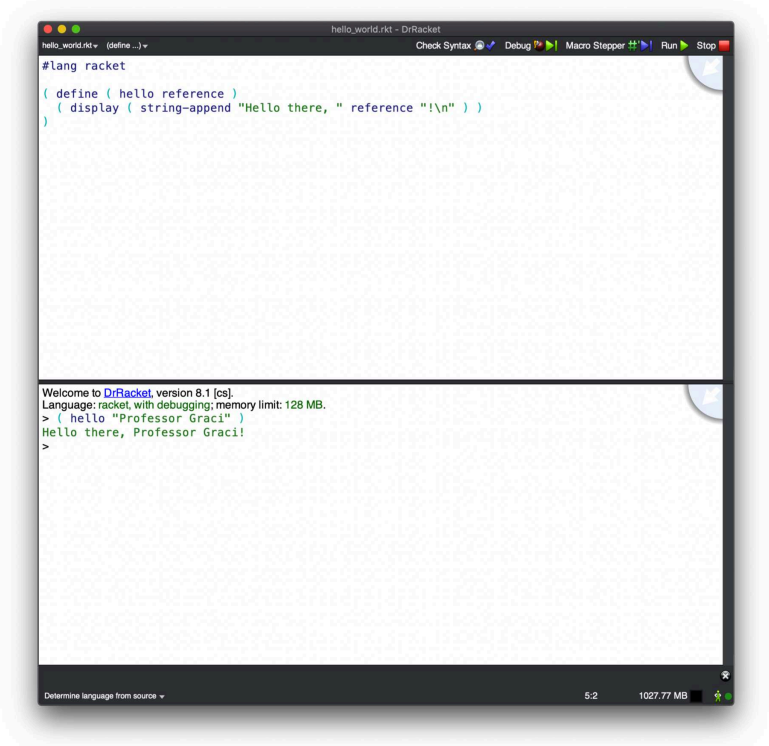
Racket Assignment #1: Getting Acquainted with Racket/DrRacket + LEL Sentence Generation

What's It All About?

1. The first thing you will need to do in order that you can do some Racket programming in the DrRacket program development environment (PDE) is to download the software. The first part of this lesson points you in the right direction to do just that.
2. Then, you will want to get acquainted with the nature of Racket and the functionality of DrRacket. The program featured in this assignment, the LEL sentence generator, is intended to facilitate your introduction to Racket and DrRacket.

Preliminary Task: Installing Racket/DrRacket

Find your way in a browser to `racket-lang.org` and then to the download button in the top righthand area of the page. Click it, and proceed to do what you need to do to download Racket/DrRacket to your machine. Your goal is to launch DrRacket, which should present itself in a window partitioned into a Definitions area and an Interactions area, along with surrounding bits of functionality. When you succeed in launching DrRacket, you should see something like this:



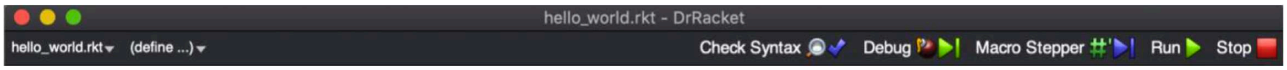
As you consider the window, imagine that:

1. I downloaded the software and then launched DrRacket.
2. I typed some code into the Definitions area, the top buffer in the arrangement that was displayed. Following the line that declares the language to be Racket (which is generally inserted automatically by the system), I typed in a variant of the traditional hello world program.

```
#lang racket

( define ( hello reference )
  ( display ( string-append "Hello there, " reference "!\n" ) ) )
)
```

3. I then clicked on the Run icon, the green triangle towards the right of the "control bar", which normally appears at the top of the PDE window, but which appears in a disembodied manner below. This action caused the Interactions area, the bottom buffer in the arrangement that was displayed, to become aware of the function definition that I placed in the Definitions area.



4. In the Interactions area (the bottom buffer), I typed in a form asking that the hello world function definition be executed. Once I hit the return key, the interpreter presented the result of the execution.

```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( hello "Professor Graci" )
Hello there, Professor Graci!
>
```

Please download the software and mimic what I did in writing and running the "Hello world!" variant that is represented in the depiction of the DrRacket PDE.

About Racket and DrRacket

1. Racket is a variant of Scheme, which was created at MIT during the 1970s by Guy Steele and Gerald Sussman. Scheme is a language influenced by Lisp and Algol. From Lisp, conceived by John McCarthy at MIT in the late 1950s, it takes the the syntax of S-expressions (and all that goes with that), the treatment of functions as first class entities, and garbage collection. From Algol, designed by committee in Europe around 1960, it takes static scoping and block structure. When the time is right, we will talk about all of these ideas.
2. Most people programming in Racket do so within the DrRacket program development environment (PDE), which was presented in brief in the preceding section.

3. You will soon note that I prefer to configure DrRacket so that the two windows are aligned horizontally, rather than vertically. There are lots of things that you can do to change the "look and feel" of DrRacket. Just look in the menus if you want to find the PDE functionality for those things.

Main Task #1: Mindfully Type a Program into the Definitions Area

Mindfully type the program presented in Appendix 1 into the definitions area of Dr Racket, and save the program under the name `lel_generator.rkt`. This program will generate random sentences consistent with the LEL grammar that appears in the lesson on BNF. However, before you commence typing, please note that there are two basic ways to proceed with your task of **typing** the given program in your Racket world:

1. Type the entire program from beginning to end. Then, try to run it. Fix it, if need be.
2. Type in just the first function. Run it. Fix it if need be. Then, type in just the second funtion. Run it. Fix it if need be. Continue in this fashion, one function at a time, until you have entered and verified each of the functions.

In which way should you choose to proceed? My recommendation is the "slower," second way. Why? Three reasons: (1) You are more likely to be successful. (2) You will stand in a different relationship to the program once you have entered it, one steeped in a richer understanding of the program and Racket. (3) You will have an opportunity to hone the habit of incremental programming, which is a practice that you should generally engage in when writing programs.

Why not simply copy and paste the code and be done with it? Two reasons: (1) You will learn almost nothing if you do the copy/paste operation. You won't have an opportunity to think about white space, to think about syntax, to think about the organization of the program. (2) The result might not run! Regularly I have seen students new to a language blindly copy programs from postscript files, oblivious to the fact that copy/paste operations can have subtle side effects pertaining to the translation of particular characters that corrupt the program.

NOTE: You are not really writing a program for this assignment. You are merely entering a program that you have been given, running it, and thinking about it. Thus, you do not need to know the language (Racket) to do this task. That said, after mindfully entering the program, running it, and reflecting upon it, you might know a little bit about the Racket language.

Main Task #2: Generate a Demo by Mimicking a Given Demo

Please generate a nice, clean demo for the program, by precisely mimicking the demo appearing in Appendix 2. Note that in this demo:

1. The `pick` function was run eight times, four times one one list, and four times on another list.
2. The `noun` function was run four times.
3. The `verb` function was run four times.
4. The `article` function was run four times.
5. The `qualifier` function was run sixteen times.
6. The `noun-phrase` function was run eight times.
7. The `sentence` function was run eight times.
8. The `ds` "function" was run twelve times.

The demo that you generate should look just like the demo that appears in Appendix 2, except that the output for each function call will probably be different

Main Task #3: Document Preparation and Posting

Prepare a single "solution document" for this assignment, consisting of a tile, followed by a short learning abstract, followed by the source code for the featured program, followed by the required demo. Please model your solution document on the template given in Appendix 3.

Then, save your document as a pdf file.

Finally, post your work for this assignment to your web work site, being sure to reference both this assignment specification and your solution. (This work should appear right after that for the "survey of programming languages" assignment in which you were to find and write about a half dozen programming languages.)

Due date: February 7, 2023

Appendix 1: LEL Sentence Generator

```
#lang racket

;-----------------------------------------------
; LEL sentence generator, with helper PICK,
; serveral applications of APPEND, several
; applications of LIST, and one use of MAP
; with a LAMBDA function.

( define ( pick list )
  ( list-ref list ( random ( length list ) ) )
)

( define ( noun )
  ( list ( pick '( robot baby todler hat dog ) ) )
)

( define ( verb )
  ( list ( pick '( kissed hugged protected chased hornswoggled ) ) )
)

( define ( article )
  ( list ( pick '( a the ) ) )
)

( define ( qualifier )
  ( pick '( ( howling ) ( talking ) ( dancing )
            ( barking ) ( happy ) ( laughing )
            () () () () () ()
          )
  )
)

( define ( noun-phrase )
  ( append ( article ) ( qualifier ) ( noun ) )
)

( define ( sentence )
  ( append ( noun-phrase ) ( verb ) ( noun-phrase ) )
)

( define ( ds ) ; display a sentence
  ( map
    ( lambda ( w ) ( display w ) ( display " " ) )
    ( sentence )
  )
  ( display "") ; an artificial something
)
```

Appendix 2: Demo

```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( pick '( red yellow blue ) )
'red
> ( pick '( red yellow blue ) )
'yellow
> ( pick '( red yellow blue ) )
'blue
> ( pick '( red yellow blue ) )
'yellow
> ( pick '( Racket Prolog Haskell Rust ) )
'Rust
> ( pick '( Racket Prolog Haskell Rust ) )
'Prolog
> ( pick '( Racket Prolog Haskell Rust ) )
'Rust
> ( pick '( Racket Prolog Haskell Rust ) )
'Haskell
> ( noun )
'(baby)
> ( noun )
'(hat)
> ( noun )
'(baby)
> ( noun )
'(hat)
> ( verb )
'(hornswoggled)
> ( verb )
'(kissed)
> ( verb )
'(hornswoggled)
> ( verb )
'(kissed)
> ( article )
'(a)
> ( article )
'(the)
> ( article )
'(the)
> ( article )
'(the)
> ( qualifier )
'()
> ( qualifier )
'()
> ( qualifier )
'(talking)
> ( qualifier )
'()
```

```
> ( qualifier )
'()
> ( qualifier )
'()
> ( qualifier )
'()
> ( qualifier )
'()
> ( qualifier )
'()
> ( qualifier )
'()
> ( qualifier )
'(howling)
> ( qualifier )
'(dancing)
> ( qualifier )
'(talking)
> ( qualifier )
'()
> ( qualifier )
'()
> ( qualifier )
'()
> ( noun-phrase )
'(the barking robot)
> ( noun-phrase )
'(a howling baby)
> ( noun-phrase )
'(the baby)
> ( noun-phrase )
'(a laughing dog)
> ( noun-phrase )
'(a happy hat)
> ( noun-phrase )
'(a hat)
> ( noun-phrase )
'(a barking todler)
> ( noun-phrase )
'(a happy baby)
> ( sentence )
'(the happy todler chased a dancing hat)
> ( sentence )
'(the howling dog hugged the hat)
> ( sentence )
'(the todler protected a baby)
> ( sentence )
'(the howling baby chased a baby)
> ( ds )
a todler hugged the happy todler
> ( ds )
a baby hornswoggled a dog
> ( ds )
a robot chased a baby
```

```
> ( ds )
a talking hat protected the dog
> ( ds )
the dancing dog protected the talking todler
> ( ds )
the robot hornswoggled a talking dog
> ( ds )
the todler kissed the robot
> ( ds )
the hat protected the talking baby
> ( ds )
a robot hugged a baby
> ( ds )
a baby hugged a laughing robot
> ( ds )
the talking baby hornswoggled a robot
> ( ds )
a baby hornswoggled the happy robot
>
```

Appendix 3: Solution Document Template

Title: **Racket Assignment #1: Getting Acquainted with Racket/DrRacket + LEL Sentence Generation**

Abstract

<<Abstract goes here>>

Code for the LEL Sentence Generator

<<Code goes here>>

Demo for the LEL Sentence Generator

<<Demo goes here>>