Resumen Progrmación I

Julián Paz

Contenidos

Introducción

Programación I es una materia que se dicta en el primer cuatrimestre del 1er año de Licenciatura en Ciencias de la Computación (LCC) en la FCEIA - UNR

En esta materia **COMPLETAR Introducción al finalizar apunte**

Unidad 1

Introducción a Dr.Racket

Como adelantamos en la introducción a la materia el lenguaje de programación que usaremos en esta materia es Dr. Racket, dicho lenguaje está basado en *Lisp* y *Scheme*, y pertenece a la rama de lenguajes funcionales.
Antes de programar en Dr.Racket debemos conocer una serie de conceptos que van a permitirnos escribir código en dicho lenguaje de programación. En general para poder programar en un lenguaje debemos conocer los siguientes elementos del lenguaje:

- Vocabulario o alfabeto: Son los elementos que podemos encontrar en nuestro lenguaje de programación por ejemplo algunos elementos del alfabeto de Dr.Racket son: *(,),+,-,and, or,1,2,3,$\cdots$,etc.*
- Sintaxis: La sintaxis de un lenguaje de programación se refiere a las reglas que tiene el lenguaje para formar expresiones, por ejemplo Dr.Racket admite la expresión *(* 8 3)*, pero no admite la expresión *8 * 3*.
- Semántica: Es el significado que adquiere cada expresión en un lenguaje de programación, por ejemplo la expresión *(* 3 8)* indica que la el número 3 es multiplicado por el 8 y reduce al número 24.

Ahora que conocemos los elementos básicos que nos van a poder permitir programar en un lenguaje, pasemos al caso general de Dr.Racket.
Lo primero que veremos son los diferentes elementos (*básicos*) que podemos encontrar en Dr.Racket. Entre los elementos más básicos podemos encontrar datos los cuales van a tener diferentes tipos (*Number, Boolean, Strings, Image*), operadores (*Cada tipo de dato tendrá sus operadores*), y funciones.
Por otro lado Dr.Racket utiliza una sintaxis prefija y paréntesis, es decir, para poder escribir una expresión que haga uso de algún operador primero deberemos escribir el operador y luego los operandos en orden normal, todo entre paréntesis.
Pasemos a ver cada tipo de dato en particular

1.1 - Numbers

Los datos de tipo Number van a representar números. algunos ejemplos son las expresiones 1, 5, π, $\cdots$, etc, por otro lado para poder operar datos de tipo Number tendremos los operadores +. -, /, sqr, sqrt, $\cdots$, etc. Veamos algunos ejemplos de como representar expresiones:

- $1 \times 2 + \sqrt{10\pi} \Rightarrow$ (+ (* 1 2) (sqrt (*10 π)))
- $\frac{1}{10}$ - $\frac{15^2}{2} \Rightarrow$ (- (/ 1 10) (/ (sqr 15 2) 2))

1.2 - Strings

Los datos de tipo String van a permitirnos representar palabras o letras. Para poder escribir un String debemos usar comillas dobles (*""*). Alguna de las operaciones que podemos realizar entre Strings son la concatenación de Strings (*string-append*), calcular el largo de un String (*string-length*), transformar un String en un Number (*string -> number*), etc.
Una observación sobre los datos de tipo String es que podemos indexar cada letra en base a su posición comenzando con el número 0, por ejemplo dado el String **"String"**, diremos que el indice de la letra **S** es 0, el de la letra **t** es 1, y así sucesivamente hasta llegar a la letra **g**, la cual tiene indice 5.

1.3 - Boolean

El tipo de dato Boolean nos permite expresar la veracidad de una expresión, tendremos dos únicos datos de este tipo: **True, False**. Pero no siempre usaremos los datos textualmente, podremos obtener dichos datos al evaluar, por ejemplo, comparaciones de Numbers. Veamos un ejemplo:

- Las comparaciones de Numbers devolveran Booleans, (= 1 2) reduce a False y (= 5 5) reduce a True

Por otro lado podremos operar expresiones que reduzcan a Booleans, para ello usaremos operadores lógicos como lo son *and, or, not*
Una observación es que al momento de reducir expresiones que evaluan a Booleans Dr.Racket tiene un proceso de cortocircuito, esto es, si usamos el operador and y encontramos que alguno de los operandos es False, la expresión reduce automáticamente a False sin ver los operandos restantes. Por otro lado, si usamos el operador or y encontramos un True, la expresión reduce automáticamente a True sin ver los operandos restantes.

1.4 - Image

Dr.Racket también nos permite representar datos que sean figuras o imágenes (*png y jpg*), para poder traer imágenes podemos pegarlas directamente, por otro lado para generar figuras debemos usar las funciones **circle, rectangle, etc.**. Además poseemos funciones como **image-width, place-image, image-height, etc.** que nos permiten obtener el tamaño de una figuras o pegar una imagen.

1.5 - Constantes y Funciones

Dr.Racket es un lenguaje que permite definir elementos que van a tener siempre un mismo valor, mejor conocido como constante, y herramientas que nos van a permitir realizar una acción en particular tomando una serie de argumentos y devolver un valor como resultado, conocidos como funciones. Veamos como definir constantes y funciones en Dr.Racket. Para definir constantes y funciones usaremos la palabra reservada **define**, y empleamos la siguiente sintaxis:

- Si queremos definir una constante lo haremos escribiendo **(define <NOMBRE CONSTANTE> <Valor Constante>)**.

- Por otro lado si queremos definir una función lo haremos usando la sintaxis **(define <Nombre Función> <Arg 1> $\cdots$ <Arg n> <Expresión o cuerpo de la función>)**

Unidad 2
Condicionales

Ahora que conocemos lo básico de Dr.Racket, enfoquemos nuestro interés en otro tema: ¿Cómo podemos hacer que nuestro programa tome decisiones? Esto lo lograremos a través de los condicionales, para introducirnos a los condicionales debemos definir unos elementos.

- Concepto proposición: Una proposición es una expresión que reduce a un único valor de verdad.

Si lo traducimos a Dr.Racket podemos pensar una proposición como una función que reduce a un Boolean. Algunos ejemplos pueden ser las funciones *integer?, string?, even?, odd?,*$\cdots$. (Funciones que determinan si un dato es entero, string, par o impar respectivamente)

2.1 - Sentencias condicionales simples

Una sentencia condicional simple es una función que nos permite modificar el flujo de un programa, dicha función utiliza la palabra reservada *if* y poseé la siguiente sintaxis:

- **(if <Condición> <Expresión1> <Expresión2>)**

Donde si la condición es verdadera reduce a la expresión 1 y si la condición es falsa reduce a la expresión 2.

2.2 - Sentencia condicional múltiple

Notemos que hasta el momento tenemos una herramienta que nos permite tomar dos caminos en base a una condición, y si queremos tomar dos de más caminos estamos obligados a utilizar ifs anidados, lo cual genera un poco de problemas al leer el código de nuestra función, para solucionar este problema Dr.Racket nos trae una sentencia condicional múltiple la cual nos permite tomar múltiples caminos en base a múltiples condiciones, dicho condicial tiene la siguiente sintaxis:

- **(cond [<Cond1> <Expr1>] [<Cond2> <Expr2>]** $\cdots$ **[<CondN> <ExprN>])**

En este caso la expresión evalua las condiciones hasta encontrar alguna que evalue a True, en dicho caso la sentencia reduce a la expresión asociada a la primera condición verdadera. En caso de que todas las condiciones evaluen a False, la sentencia cond reduce a error

Unidad 3

Programas interactivos

Podemos categorizar los programas en dos clases según su ejecución, por una parte tenemos aquellos programas que no dependen del usuario para funcionar (*Programas por lotes*) y aquellos que dependen del usuario para poder avanzar (*Programas interactivos*). (A las acciones externas al programa por las cuales el usuario puede avanzar en la ejecución del programa son llamadas *eventos*)
Centremos nuestro interés en una nueva problemática: Crear programas con los cuales podamos interactuar nosotros como usuario. Lo que queremos hacer es intentar generar un programa que pueda realizar una función o una serie de funciones al recibir una entrada del usuario como lo puede ser un click, apretar una tecla, etc.
Consideremos el conjunto de propiedades y valores actuales en un momento determinado del programa, y llamemos a dicho conjunto estado de un programa. Luego en un programa interactivo podemos considerar a los eventos como acciones que nos permiten pasar de un estado a otro del programa. Ahora bien el evento en sí no genera un cambio en el estado de un programa interactivo, ese cambio puede ser llevado a cabo por una función encargada de detectar dichos eventos y llevar a cabo una acción que cambie el estado del programa, dicha función se llama manejador de eventos.

3.1 - Expresión Big-Bang

La expresión big-bang es una herramienta que nos ofrece el lenguaje Dr.Racket para poder manejar eventos, la misma herramienta tiene la siguiente sintaxis: