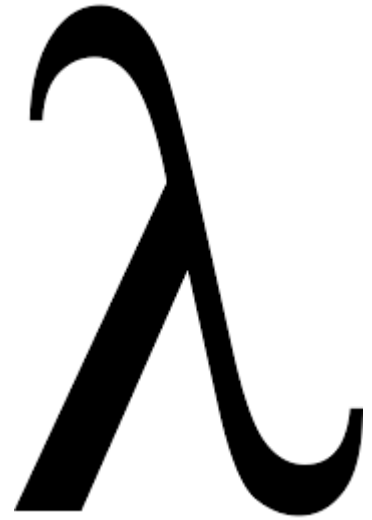
Programacion en Dr. Racket (Dr. Scheme)

Profesor Francisco Alejandro Medina

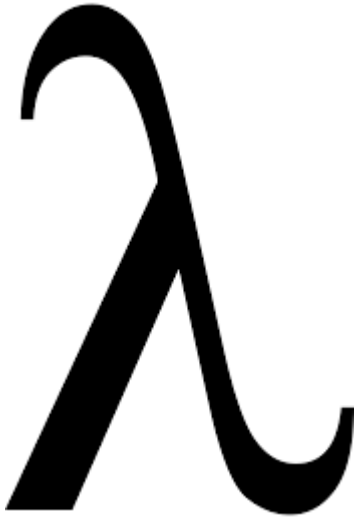
Programación Funcional

- La programación funcional, iniciada a finales de la década de los 50's, es aquella cuyo paradigma se centra en el Cálculo Lambda.
- Este paradigma es más útil para el área de inteligencia artificial (ya que satisface mejor las necesidades de los investigadores en esta área), y en sus campos secundarios: cálculo simbólico, pruebas de teoremas, sistemas basados en reglas y procesamiento del lenguaje natural.
- La característica esencial de la programación funcional es que los cálculos se ven como una función matemática que hace corresponder entradas y salidas.



El calculo Lambda

El **cálculo lambda** es un sistema formal diseñado para investigar la definición de función, la noción de aplicación de funciones y la recursión. Fue introducido por Alonzo Church y Stephen Kleene en la década de 1930; Church usó el cálculo lambda en 1936 para resolver el Entscheidungsproblem. Puede ser usado para definir de manera limpia y precisa qué es una "función computable". El interrogante de si dos expresiones del cálculo lambda son equivalentes no puede ser resuelto por un algoritmo general. Esta fue la primera pregunta, incluso antes que el problema de la parada, para el cual la indecidibilidad fue probada. El cálculo lambda tiene una gran influencia sobre los lenguajes funcionales, como Lisp, ML y Haskell.



Objetivo de la Programación Funcional

- Conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, y evitando el concepto de estado del cómputo.
- Acercar su notación a la notación normal de la matemática.

Características de la Programación Funcional

- Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones (funciones puramente matemáticas
- No hay algo como el estado de un programa, no hay variables globales.
- La no existencia de asignaciones de variables
- La falta de construcciones estructuradas como la secuencia o la iteración (no hay for, ni while, etc).
- Todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas.

Lenguajes de Programación Funcionales

Se clasifican en dos Tipos:

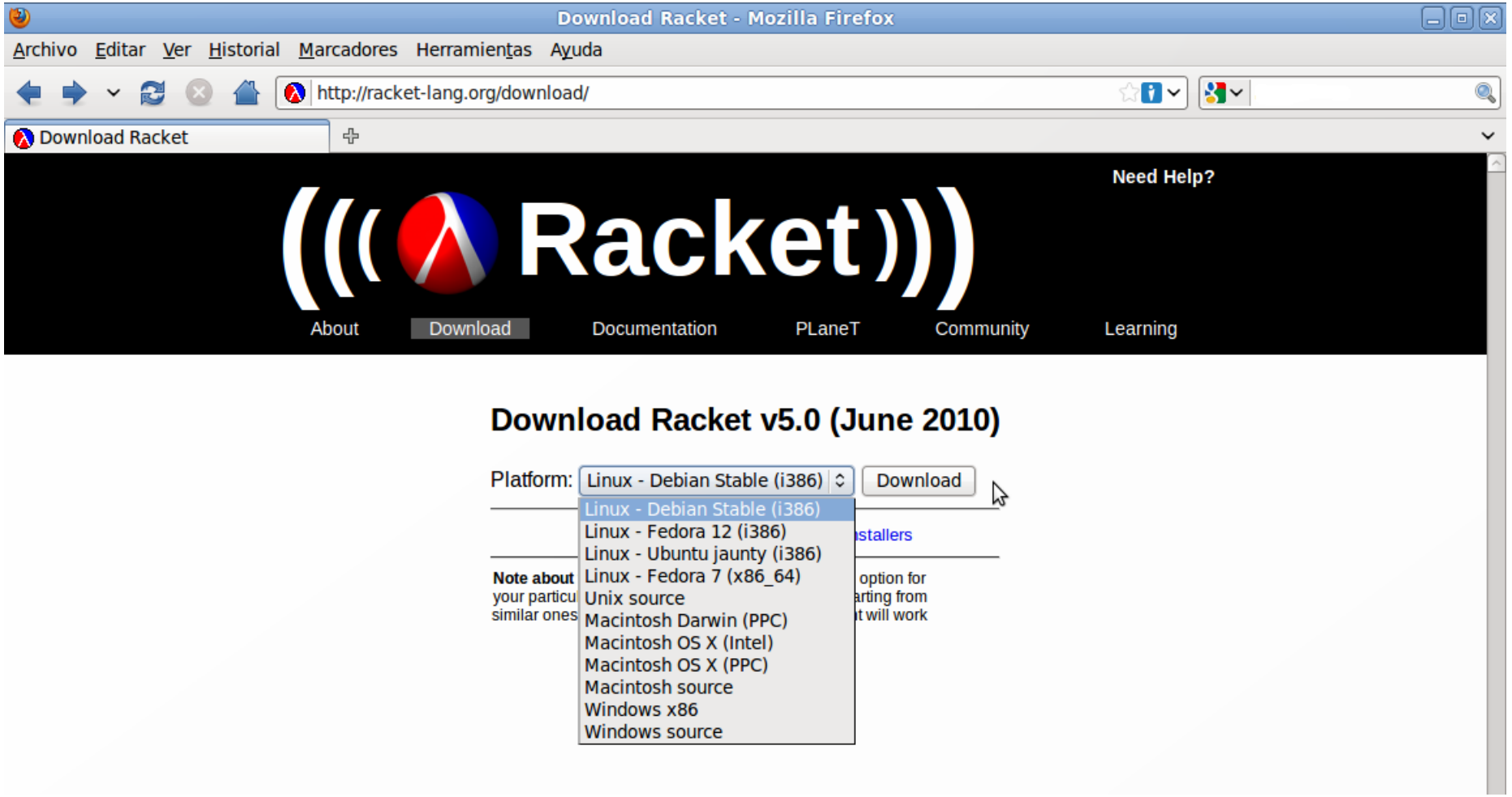
- Funcionales Puros (Tienen Mayor potencia, y se usan de forma especial en aplicaciones eminentemente matemáticas)

  Ej: Haskell y Miranda.

- Funcionales Híbridos (Son Menos Dogmaticos que los puros ya que incluyen conceptos tomados de los lenguajes imperativos (secuencia y asignación de variables

  Ej: Lisp, Scheme (Racket), Ocaml, Erlang…

El Lenguaje Dr. Racket



Notación para la sintaxis de Racket

- Todas las secuencias de caracteres delimitadas por < y > representan símbolos no terminales. Por ejemplo: <símbolo_no_terminal>.
- Todas las secuencias de caracteres no delimitadas, representan símbolos terminales.

  Por ejemplo: define, (,), let.
- El metaagrupamiento se hace con llaves: { y }.
- El metasímbolo +, indica al menos una ocurrencia del símbolo precedente.
- El metasímbolo *, indica ninguna, una o varias ocurrencias del símbolo precedente.

Expresiones Simples

- Los numerales

  >578

  578

- Procedimientos primitivos: +,-,*,/.

  >(+ 115 226)

  341

  >(- 1000 334)

  666

  >(* 5 99)

  495

  >(/ 10 6)

  1.66667

- Combinación: $\underbrace{(\underbrace{< op >}_{\text{Operador}} \overbrace{< op_1 >< op_2 > \ldots < op_n >}^{\text{Operandos}})}_{\text{NotaciónPrefija}}$

Notación Prefija e Infija

- Estamos acostumbrados a la adición de números usando la notación infija , por ejemplo :

  1+2+3+4+5=15

- Racket utiliza una notación Prefija, Esto significa que el operador está antes cada uno de sus argumentos :

  (+ 1 2 3 4 5) =>15

  (* (+ 3 4) (- 9 6)) =>21

  Su equivalente en Notación Infija seria

  (3 + 4) . (9 – 6) = 21

$$\frac{\frac{a}{b} + \frac{b}{c}}{\frac{a}{b} - \frac{b}{c}}$$

$$a + b + \frac{c}{d} + \frac{\frac{a}{b - c}}{\frac{a}{b + c}}$$

$$a + \frac{a + \frac{a + b}{c + d}}{a + \frac{a}{b}}$$

a = 2
b = 3
c = 4
d = 1

$$\frac{a + b + c}{a + \frac{b}{c}}$$

$$\frac{a + b + \frac{c}{d * a}}{a + b * \frac{c}{d}}$$

$$\frac{a + \frac{b}{c} + d}{a}$$

a = 2
b = 3
c = 4
d = 1

Notación prefija de Racket

En Racket, todas las expresiones tienen la forma: (<operador> <operando>*), es decir, que están siempre en notación prefija con pareamiento completo:

(* 2 3) -> equivale a (2 * 3)

(> 5 6)-> equivale a (5 > 6)

(+ 2 3 10)-> equivale a (2 + 3 + 10)

(+ 4 (* 3 2))-> equivale a (4 + 3 * 2)

Por ejemplo, la expresión infija 5a + 2bc^2 es: (+ (* 5 a) (* 2 b c c))

Ventajas de la Notación Prefija

- Procedimientos con número arbitrario de argumentos

  >(+ 21 35 12 7)

  75

- Facilidad de anidamiento

  >(+ (* 3 5) (- 10 6))

  19

Ventajas de la Notación Prefija

- Para facilitar la lectura,

$$(+\ (*\ 3\ (+\ (*\ 2\ 4)\ (+\ 3\ 5)))\ (+\ (-\ 10\ 7)\ 6))$$

se escribirá:

$$\left.\begin{array}{l}(+\ (*\ \ 3\\ \qquad (+\ \ (*\ \ 2\ \ \ 4)\\ \qquad\quad\ (+\ \ 3\ \ \ 5)))\\ \quad (+\ \ (-\ \ 10\ \ 7)\\ \qquad\ 6))\end{array}\right\}\ \text{Pretty printing}$$

Notaciones en Expresiones Aritméticas

Las expresiones aritméticas pueden ser representadas en 3 notaciones diferentes, infija, prefija y posfija. Cada una de estas notaciones se obtiene al recorrer el árbol en inorden, preorden y posorden respectivamente.

Notación Infija (Recorrido Inorden)

$$4 + 3 * 8$$

Notación Prefija (Recorrido Preorden)

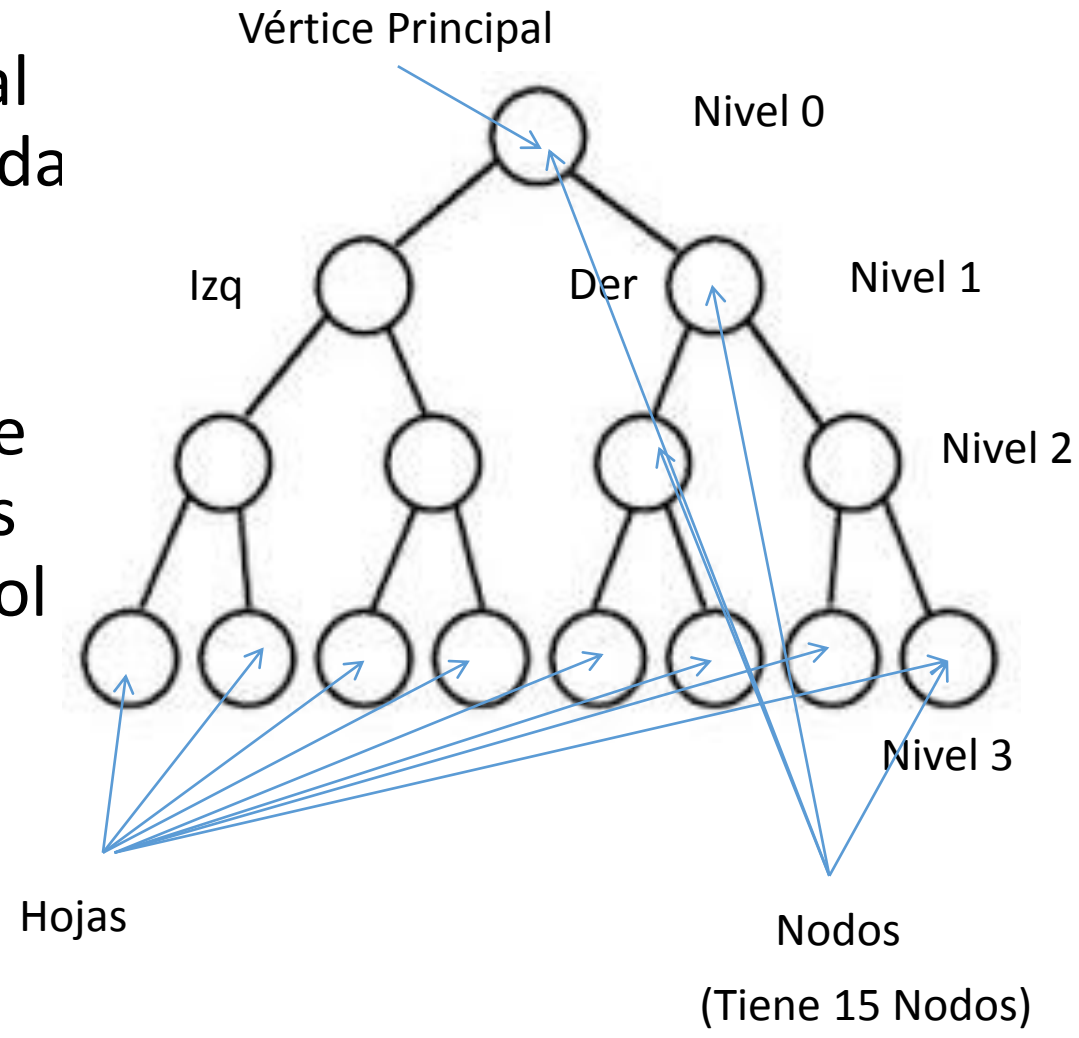
$$+ 4 * 3 8$$

Notación Posfija (Recorrido Posorden)

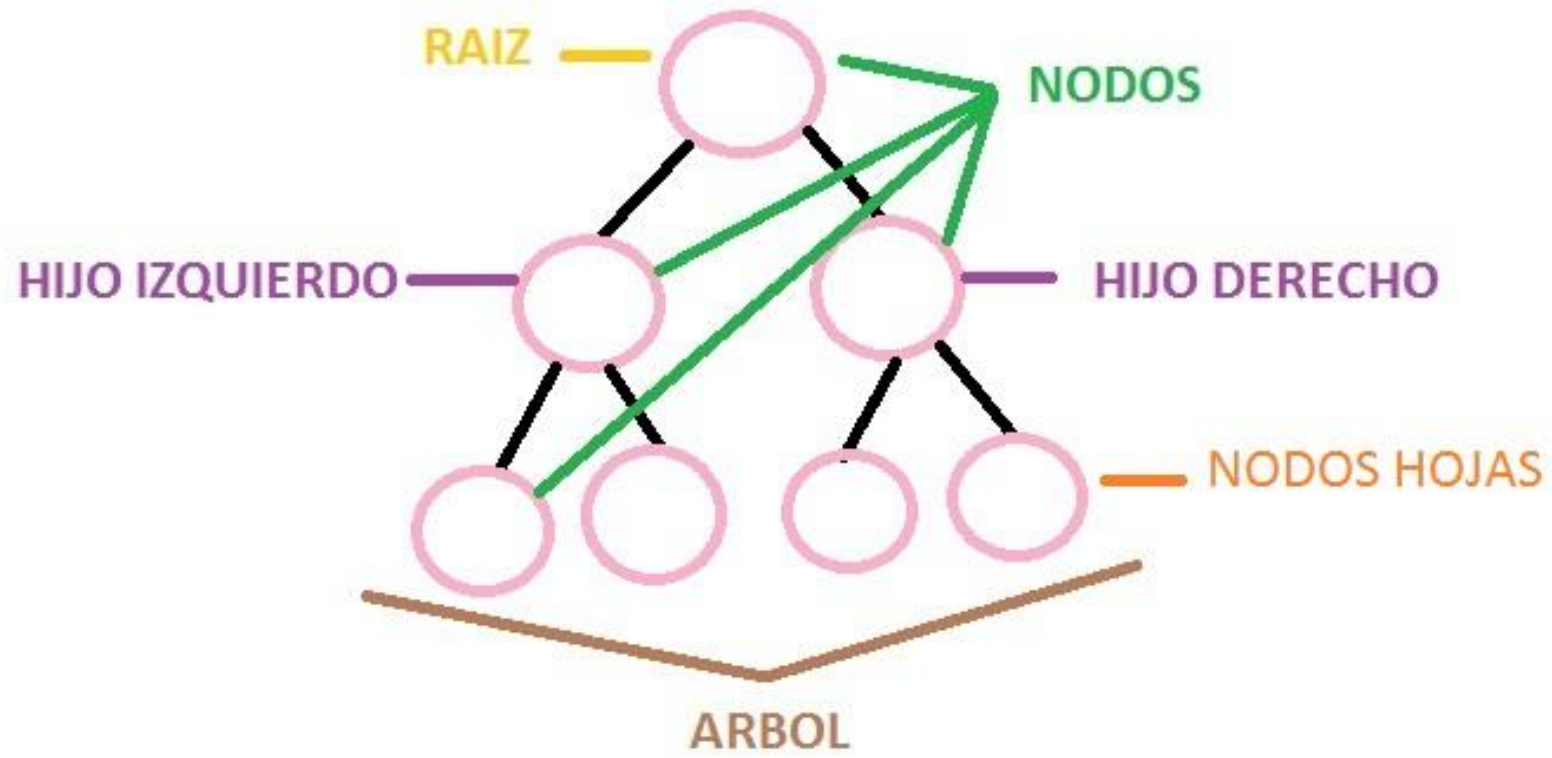
4 3 8 * +

Arboles Binarios

- El árbol Binario tiene un vértice principal del cual se desprenden 2 ramas (izquierda y derecha)
- Si la rama izquierda y la rama derecha también son 2 arboles binarios el vértice principal se llama Raíz y cada una de las ramas se les llama árbol izquierdo y árbol derecho. Cuando no se desprenden de ninguna otra rama se le conoce como hoja.



Arboles Binarios



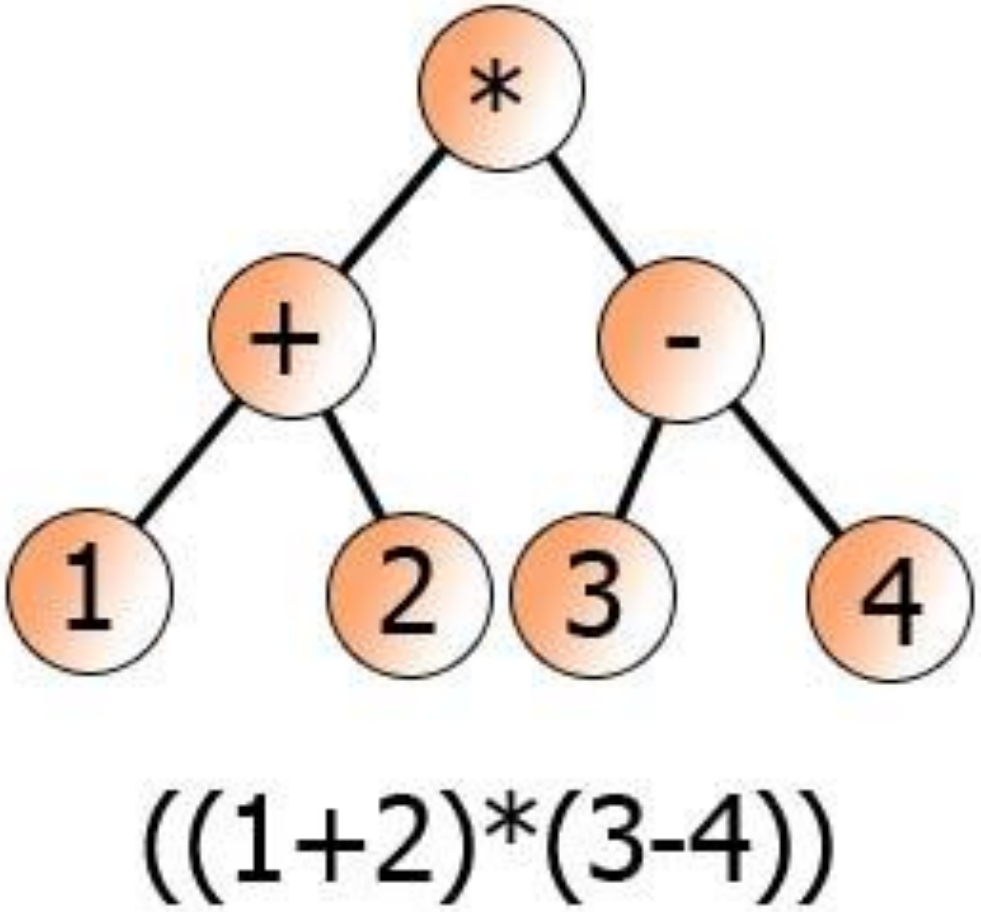
Ejemplo

El árbol genealógico es un árbol binario.

- Cada nodo tiene dos hijos
- Es significativo el orden de los subárboles.

Árbol de Sintaxis

- simplemente un **árbol de sintaxis**, es una representación de árbol de la estructura sintáctica abstracta (simplificada) del código fuente escrito en cierto lenguaje de programación



Volver válida cada expresión, de acuerdo a la sintaxis del lenguaje de expresiones válidas planteada en clase. Y a partir de allí construir el árbol de sintaxis de la expresión numérica correspondiente e indicar cual es la expresión en prefijo y postfijo de las siguientes expresiones:

1. 7
2. - a
3. 3 + 8
4. 11 / 4
5. 2 + (4 * 7)
6. a – b / c
7. 6 * 4 + 5
8. (3 + 9) – (2 * 6)
9. (((r / p)))
10. (((6 + (8 *2)) – ((9 - 2) / 3)) + ((8 / 5) * (7 + 6)))
11. ((k + ((s * ((h / (i - j)) - n)) * w)) – (((a – (d + e)) * ((q * p) / y)) / (r + x)))

Para las siguientes expresiones en prefijo, construya el árbol de sintaxis y la expresión en infijo.

1. (- 7 5)
2. (* (+ 4 7) (- 8 5))
3. (+ g (* (- r d) (/ m (+ (* n v) b))))
4. (/ (* a (/ (- f (+ r q)) (* (/ y w) (- (+ x k) j)))) p)

Manejo de Identificadores

- Identificador: identifica una variable cuyo valor es un objeto.
- Operador utilizado: *define*.

  >(define peso 75)

  peso

  Nota: En LISP toda función devuelve un valor

  >peso

  75

  >(* 2 peso)

  150

Manejo de Identificadores

- En general:

(define <ident> <expresión>)

Manejo de Identificadores

- Más ejemplos:

  >(define pi 3.14159)

  pi

  >(define radio 10)

  radio

  >(define circunferencia (* 2 *pi radio*))

  circunferencia

  >circunferencia

  62.8318

Tipos de Datos Simples

DrRacket utiliza diferentes tipos de objetos :

- Enteros: por ejemplo, 3 , 0 , -12 . La Aritmética de enteros utiliza precisión arbitraria .
- Fracciones: por ejemplo, 2/3 , -5/6 .
- Números de punto flotante : por ejemplo, 2.5 , -0.00303
- Los números complejos : por ejemplo 0.0+1.0i representa $i = \sqrt{-1}$,

  y $x+y\mathtt{i}$ Representa $x + i\,y$

Otros tipos de Datos

- Cadenas (strings): "abc " , " Abc "
- Caracteres Characters: #\A , #\b , #\c
- Funciones : +, - , *, / , car, cdr
- Booleanos (Valores de verdadero o falso) : #t , #f

Funciones Numericas

DrRacket tiene tantas funciones incorporadas al igual que las calculadoras de bolsillo :

- Square root: `(sqrt n)` $\Rightarrow \sqrt{x}$, e.g., `(sqrt 9)` $\Rightarrow$ `3`
- Exponents: `(expt x y)` $\Rightarrow x^y$, e.g., `(expt 2 3)` $\Rightarrow$ `8`
- Exponentials: `(exp x)` $\Rightarrow e^x$, e.g., `(exp 2)` $\Rightarrow$ `7.38905609893065`
- Natural Logarithm: `(log x)` $\Rightarrow \log_e x$, e.g.,
  `(log 2)` $\Rightarrow$ `0.6931471805599453`
- Trig functions: (argument must be in radians)
  - `(sin x)` $\Rightarrow \sin x$
  - `(cos x)` $\Rightarrow \cos x$
  - `(tan x)` $\Rightarrow \tan x$
  - `(asin x)` $\Rightarrow \arcsin x$
  - `(acos x)` $\Rightarrow \arccos x$
  - `(atan x)` $\Rightarrow \arctan x$
  - `(atan x y)` $\Rightarrow \arctan(x/y)$

Para la Funciones trigonométricas el argumento debe estar en radianes

Definiendo Nuevas Funciones

Es fácil de definir nuevas funciones en DrRacket .

```
(define (add1 x)
        (+ x 1))
```

define es una palabra clave especial que indica que estamos definiendo una nueva función. x es un variable ficticia que representa un solo argumento . Una vez add1 ha sido evaluada (o ejecutado) , que puede ser utilizado :

```
(add1 7) ⇒ 8
(add1 -0.5) ⇒ 0.5
(add1 (add1 7)) ⇒ 9
```

Definiendo Nuevas Funciones

define también se puede utilizar para definir nuevos símbolos o variables:

```
(define my-name "Robert")
(define golden-mean (/ (+ (sqrt 5) 1) 2))
```

Ejemplo de Funciones

```
;FUNCION QUE CALCULA EL AREA DE UN TRAPECIO

;Calcular el area de un trapecio
;area_trapecio  bm  b  h    =>  #
;donde  bm es la base mayor, b es la base menos y h es la altura
;ejemplo ( area_trapecio 18  8  6 ) => 78
;cuerpo del programa

 ( define (area_trapecio bm b h)
    ( / ( * ( + bm  b ) h ) 2 ) )
;prueba

 ( area_trapecio  18  8  6  )
```

```
Welcome to DrRacket, version 6.1.1 [3m].
Language: Beginning Student; memory limit: 128 MB.
78
>
```

Ejemplo de Funciones

```
;calcular el area de un triangulo
; area_triangulo  b  h   =>  #
;donde b es la base del triangulo y h es la altura del mismo
;ejemplo ( area_triangulo 2 4 )  => 4
;cuerpo del programa
  ( define ( area_triangulo b  h  )
     ( / ( *  b  h ) 2 ))
; prueba

  ( area_triangulo 2 4 )
```

```
Welcome to DrRacket, version 6.1.1 [3m].
Language: Beginning Student; memory limit: 128 MB.
4
>
```