1. Primera sesión: *Presentación / Primitivas de Racket*

1.1. Objetivos

- ⋆ Presentar los lineamientos del curso.
- ⋆ Instalar DrRacket.
- ⋆ Dar una breve introducción a las primitivas del lenguaje de programación Racket.

1.2. Introducción

1.2.1. Descripción

El laboratorio de Lenguajes de Programación es un complemento a la clase del profesor, con el objetivo de apoyar a los alumnos a adquirir experiencia práctica en el diseño e implementación de distintos intérpretes para entender qué hace un lenguaje de programación.

El lenguaje de programación que utilizaremos a lo largo del curso es Racket en su variante `#plai`. Este lenguaje nos permite definir nuevos tipos de datos a partir de gramáticas, sin perder la expresividad matemática que éstas nos dan. Además nos permite realizar una caza de patrones para operar con dichas gramáticas y así poder crear distintos intérpretes para entender lo que hace un lenguaje de programación en alto nivel, pero sin dejar de lado qué pasa a nivel del ensamblador.

En ocasiones se usarán otros lenguajes como Haskell, Prolog o Python, para ejemplificar algunos temas como evaluación perezosa, inferencia de tipos u orientación a objetos.

1.2.2. Evaluación

A lo largo del curso se realizarán 8 prácticas que tendrán un peso del 30 % sobre la calificación final. Cada práctica consistirá en resolver distintos problemas utilizando los conceptos vistos en el curso hasta ese momento. Cada práctica tendrá un periodo de dos semanas para resolverse. Todas las prácticas se realizarán en equipos de **máximo 3 integrantes**.

1.2.3. Formato de entrega

Las prácticas serán entregadas cumpliendo los siguientes lineamientos

1. Se debe entregar un directorio `<equipo>_PXX`, donde `equipo` es el nombre del equipo y `XX` es el número de práctica a dos dígitos. Dentro del directorio se debe incluir:
   - ⋆ Un archivo `readme.txt` con el nombre de los autores, comentarios adicionales, opiniones, críticas o ideas sobre la práctica.

- ⋆ Los archivos requeridos en la práctica. Debe enviarse código lo más limpio posible. Un código limpio:
  - ∗ No tiene líneas muy largas (80 caracteres máximo).
  - ∗ Tiene identación consistente (3 o 4 espacios).
  - ∗ No tiene código o comentarios que no son usados.
  - ∗ No tiene funciones muy complicadas.
  - ∗ No tiene contenido repetitivo (*copy-paste*).

2. El directorio se enviará en un archivo comprimido `tar` o `zip` con nombre `<equipo>.tar` donde `equipo` es el nombre del equipo.

3. El archivo se enviará a `manu+ldp@ciencias.unam.mx` con el asunto `"[LDP-PXX]"` donde `XX` es el número de la práctica a dos dígitos.

4. La hora límite de recepción de prácticas es a las 23:59:59 horas del día fijado como fecha de entrega.

5. Si se detecta una práctica plagiada total o parcialmente, **se obtendrá automáticamente una calificación de cero en la misma y no se aceptarán más prácticas**.

La calificación de las prácticas se enviará al correo electrónico del alumno pasada una semana de la entrega correspondiente. Sólo habrá una semana para aclaraciones.

1.2.4. Forma de trabajo

1. Introducción al tema mediante una explicación breve y demostración de código a los alumnos para aprender la sintaxis o formato del lenguaje.

2. Ejercicios breves para desarrollar en laboratorio, para que los alumnos experimenten y se puedan aclarar dudas del código.

3. Hacer una conclusión de lo revisado y en caso de ser necesario acompañar de otra ronda de breves ejercicios.

1.2.5. Tipos de datos primitivos

1. Booleanos.

   `#t #f` ← Formas canónicas.

   true false ← Azúcar sintáctica

2. Números.

   - ⋆ Exactos: Su valor es conocido al 100 %
     - ∗ Enteros `-1 7`

- ∗ Racionales `1/7 2/9`
- ∗ Complejos `1+7i 2+9i`

⋆ Inexactos: Valor desconocido, no siempre se sabe su valor exacto.

- ∗ Flotantes `5.5 +inf.0 -inf.0 5.25e8`
- ∗ Complejos con flotantes `2.0+3.7i`

Los números pueden ser tan grandes como uno quiera, no hay límite en el tamaño.
`9589329875083257803471361430785628371562108735602378502345702487`

3. Caracteres

   Racket usa la codificación Unicode.

   `#\a #\e #\i #\E #\u3123` ← Se puede usar el código de Unicode.

4. Cadenas

   Son agrupaciones de caracteres.

   `“Hola mundo” \” \n`

5. Símbolos

   Son valores atómicos. La comparación de símbolos es más barata que una comparación de cadenas. Usan la notación `qoute` (más adelante hablaremos de esto).

   `’manzana ’hola ’racket`

1.2.6. Funciones predefinidas

Racket es un lenguaje de notación prefija con paréntesis. El primer elemento después del paréntesis “(“ es el nombre de la función a aplicar, y lo que sigue hasta llegar al paréntesis “)” son los parámetros.

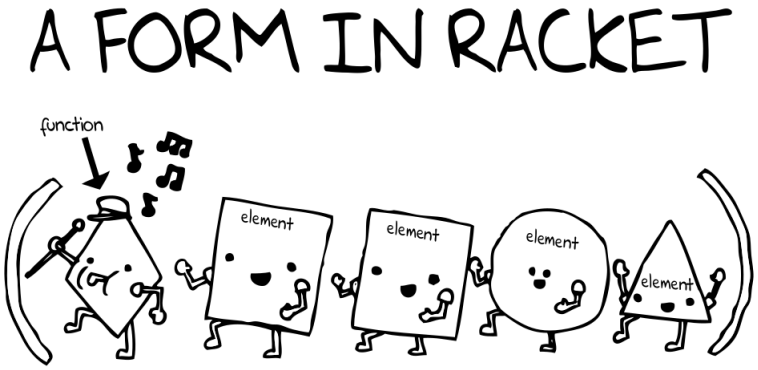

Imagen tomada de [1].

Ejemplos. Algunas funciones con tipos de datos primitivos:

```
> (+ 1 2)
```
En esta expresión primero se evalúa el lado izquierdo, luego el derecho y al final se evalúa la función.

```
> (* 2 4 5)
```

```
> (- 1 1/4)
```

```
> (+ 1 (- 3 4))
```
Evaluamos el `1`*, luego la expresión* `(- 3 4)` *y al final la suma.*

```
> (sqrt -1)
```

```
> (or (< 5 4) (equal? 1 (- 6 5)))
```
Idiom: Las funciones que regresan valores booleanos terminan con `?` *y son llamadas predicados.*

```
> (and (not (zero? 10)) (+ 1 2 3))
```
Todo lo que no es falso es verdadero. La función `and` *devuelve la evaluación de la última expresión verdadera.*

```
> (string-append ‘‘Ho’’ ‘‘la’’)
```
Idiom: Los nombres de funciones se escriben con minúsculas y cada palabra se separa con un guión.

```
> (string-length ‘‘hola mundo’’)
```

```
> (substring ‘‘Apple’’ 13)
```

```
> (string->symbol ‘‘Apple’’)
```
Idiom: Las expresiones que hacen convenciones de tipos se nombran poniendo un “`->`*” entre los tipos a convertir y son llamados conversores.*

```
> (display ‘‘hola’’)
```

```
> (+ 1 ‘‘Hola’’)
```
Racket es un lenguaje de tipo seguro. Al invocar una función con argumentos de tipo equivocado, se genera un error.

•

1.2.7. Condicionales

El típico `if`:

```
(if <condición> <then> <else>)
```

```
(if (> 4 10)
    "hola"
    "adiós")
```

`If` anidados:

```
(if (> 2 3)
    (display "hola")
    (if (> 6 5)
        (display "adiós")
        #f))
```

Es más legible usar `cond` para mútilples desiciones.

```
(cond
    [<condicion> <expresión>]+
    [else <expresión>]*)
```

El '+' significa uno o más.
El '*' significa cero o más.

```
(cond
    [(> 2 3) (display "hola")]
    [(> 6 5) (display "adiós")]
    [else #f])
```

1.2.8. Identificadores

Se pueden definir identificadores para que queden globalmente disponibles. Estas definiciones deben hacerse en el área de de definiciones del IDE.

Ejemplo. Uso de identificadores.

```
> (define MAX 100)

> (MAX)

> (< 25 MAX)

> (define x 10)

> (+ x 1)
```

•

También hay identificadores con alcance local:

```
(let ([<id> <expresión>]+)
    <cuerpo>)
```

Ejemplo. Uso de identificadores con alcance local.

```
> (let ([x 3] [y 2]) (+ x y))

> (define x 10)

> (define y 12)

> x

> y

> (let ([a 3] [b (+ a a)]) b)
```
No se permite usar un identificador en el cálculo de otro.

```
> (let ([a 3]) (let ([b (+ a a)]) b))
```
La solución es anidar.

```
> (let* ([a 3] [b (+ a a)]) b)
```
¡Tiene azúcar sintáctica!

•

1.2.9. Definición de funciones

Además de las funciones existentes, se pueden definir funciones propias:

```
(define (<nombre> <parametro>+)
    <cuerpo>)
```

Ejemplos. Algunos ejemplos de funciones.

```
> (define (doble x)
    (+ x x))
> (doble 2)

> (define (foo x)
    (if (< x 10)
        (diplay "menor")
        (display "mayor")))
> (foo 4)
> (foo 11)
```

```
> (define (suma a b)
    (+ a b))
> (suma 1 7)
> (suma 2 9)

> (define (selecciona x y)
    (if (< (random) 0.5)
        x
        y))
> (selecciona 1 7)
> (selecciona 1 7)
> (selecciona 1 7)
> (selecciona 2 9)
> (selecciona 2 9)
> (selecciona 2 9)
```

1.3. Ejercicios

1. Transforma las siguientes expresiones aritméticas en expresiones de Racket y pruébalas en la *ventana de interacciones*.

   a) $(4 \times 7) - (13 + 5)$

   b) $(3 \times (4 + (-5 - -3)))$

   c) $(2.5 \div (5 \times (1 \div 10)))$

   d) $5 \times ((537 \times (98.3 + (375 - (2.5 \times 153)))) + 255)$

2. Introduce localmente dos identificadores asociados a números y regresa el máximo.

3. En un curso de lenguajes de programación, el promedio de cada alumno se calcula mediante exámenes parciales, tareas y prácticas. Los exámenes tienen un valor del 40 % de la calificación final, las tareas y las prácticas 30 %. Escribe una función `(promedio e t p)`, donde e, es la calificación de exámenes, t el promedio de tareas y p el promedio de prácticas, que regresa el promedio total. Ejemplos:

```
>(promedio 10 10 6)
8.8
>(promedio 7 0 2)
3.4
>(promedio 10 0 0)
4.0
```

4. Escribe una función `(imc p e)` para calcular el índice de masa corporal de una persona. Fórmula:

$$IMC = \frac{peso}{estatura^2}$$

Ejemplos:

```
>(imc 65.0 1.75)
21.22
>(imc 63.0 1.56)
25.88
>(imc 68.0 1.80)
20.98
```

1.4. Referencias

[1] Felleisen, M., Barski C., *Realm of Racket*, No Starch Press, First Edition.

[2] Navas, Eduardo., *Programando con Racket 5*, Universidad Centroamericana "José Simeón Cañas", Primera edición, 2010.

[3] Tanter, Éric., *PrePLAI: Scheme y Programación Funcional*, 2011. Disponible en: http://users.dcc.uchile.cl/∼etanter/preplai/

[4] Flatt, Matthew., *The Racket Reference.*
Disponible en: https://docs.racket-lang.org/reference/index.html

[5] Grillmeyer, Oliver., *Exploring computer science with Scheme.*

[6] Springer, George., *Scheme and the art of programming.*