# Compiladores 2024-2
# Facultad de Ciencias UNAM
# Práctica 1: Hola Racket.

Lourdes del Carmen Gonzáles Huesca  Juan Alfonso Garduño Solís
Fausto David Hernández Jasso

31 de enero
**Fecha de Entrega:** 9 de febrero

Los siguientes son ejercicios de clásicos programación y nos servirán para empezar a trabajar con Racket, resuelvelos teniendo en cuenta que serán revisados con pruebas unitarias así que **respeta el nombre especificado en las instrucciones**.

1. Programa tu propia versión de las siguientes funciones de orden superior en Racket:

   - *(0.5 puntos)*

     ```
     (my-map f l) -> lista
     f : funcion/procedimiento
     l : lista
     ```

   - *(0.5 puntos)*

     ```
     (my-filter p l) -> lista
     p : predicadol
     l : lista
     ```

   - *(0.5 puntos)*

     ```
     (my-foldr f i l) -> b
     f : funcion/procedimiento (f: a b -> b)
     i : b
     l : lista de a
     ```

   - *(0.5 puntos)*

     ```
     (my-foldl f i l) -> b
     f : funcion/procedimiento (f: b a -> b)
     i : b
     l : lista de a
     ```

     *Ten en cuenta que `foldl` en racket tiene un comportamiento diferente al de `foldl` en haskell.*

2. Uno de los métodos de cifrado de mensaje más simples y antigüos conocidos es el cifrado de caesar, este se basa en reemplazar cada letra del mensaje por una letra desplazada $n$ posiciones en el alfabeto en el que se está trabajando, por ejemplo:

   $$\text{abcdz} \xrightarrow[n=1]{caesar} \text{bcdea}$$

   - *(1 punto)* Implementa una función `caesar-encoder` que reciba una cadena `s` y un número entero `n` para devolver el `s` cifrada con `n`.
   - *(1 punto)* Implementa una función `caesar-decoder` que reciba lo mismo que la anterior pero ahora devuelva la cadena `s` descifrada con `n`.

   Estas funciones deben considerar como alfabeto el abecedario sin la letra eñe y respetar las mayúsculas. En caso de que el mensaje tenga un carácter que no pertenezca al abecedario se debe ignorar.

**Por ejemplo:**

```
> (caesar-encoder "EsTe Es Un MeNsAjE !#$%&/()" 3)
> "HvWh Hv Xq PhQvDmH !#$%&/()"
> (caesar-decoder "HvWh Hv Xq PhQvDmH !#$%&/()" 3)
> "EsTe Es Un MeNsAjE !#$%&/()"
```

3. *(2 puntos)* Programa una función `get-two` que dada una lista de números enteros `l` y un objetivo `o` que también es entero, regresa una lista de *dos* índices de `l` tal que la suma de los elementos en esos índices sea igual a `o`.
   Los indices de la lista resultante deben ser diferentes, es decir, si hacemos

   ```
   > (get-two '(3 2) 6)
   ```

   '(0 0) no puede ser una respuesta válida.
   **Ejemplos:**

   ```
   > (get-two '(1 9 7 2 3) 4)
   > '(0 4)
   > (get-two '(9 9) 18)
   > '(0 1)
   > (get-two '(-7 9 8 2 3 1 7) 0)
   > '(0 6)
   > (get-two '(-7 9 8 2 3 1 7) -6)
   > '(0 5)
   ```

   Para resolver este ejercicio debes implementar el algoritmo que usa *tablas hash*, utiliza las siguientes funciones:

   ```
   (make-hash)          ; Para crear una tabla
   (hash-has-key? t k)  ; Para saber si t contiene la clave k
   (hash-ref t k)       ; Para obtener el el valor asociado a k en la tabla t
   (hash-set! t k v)    ; Para agregar el valor v asociado a k en la tabla t
   ```

4. Un palíndromo es una palabra o frase que se lee igual al derecho que al revés. Por otra parte, un anagrama es una palabra creada reordenando los elementos que conforman otra palabra diferente, por ejemplo `taco` es anagrama de `cató`.
   *A pesar de que no cumple con la definicion para este ejercicio vamos a decir que una palabra es anagrama de si misma.*

   - *(1 punto)* Escribe una función `palindromo` que sea capaz de recibir números, cadenas y listas para contestar si lo que recibe es un palíndromo o no.
     **Ejemplos:**

     ```
     > (palindromo "Anita Lava la TiNa")
     > #t
     > (palindromo "áéíea")
     > #t
     > (palindromo 1001001)
     > #t
     > (palindromo '("Hola" "Como" "Estas" "Como" "Hola"))
     > #t
     ```

   - *(1 punto)* Escribe una función `anagramas-de` que reciba una cadena `s` y una lista de cadenas `l` para filtrar las cadenas de `l` que sean anagramas de `s`.

     ```
     > (anagramas "TaCo" '("aCató" "actÚ" "cAtó" "Saco"))
     > #t
     ```

   Como puedes ver en lo ejemplos estas funciones **no** deben hacer distinción entre mayúsuclas y minúsculas, así como las vocales acentuadas.

5. *(2 puntos)* La siguiente es conocida como la conjetura de Collatz:

   *Sea **n** un número entero positivo, si se le aplican los dos siguientes pasos repetidamente, eventualmente llegará a 1:*

   *Si **n** es par divídelo entre dos, en otro caso multiplicalo por 3 y sumale 1.*

   Por ejemplo para 12:

| Paso | resultado |
|---|---|
| 0 | 12 |
| 1 | 6 |
| 2 | 3 |
| 3 | 10 |
| 4 | 5 |
| 5 | 16 |
| 6 | 8 |
| 7 | 4 |
| 8 | 2 |
| 9 | 1 |

   Para este ejercicio debes escribir dos funciones:

   - *(1 punto)* `pasos-collatz` que recibe un número entero positivo y responde con el número de pasos que se deben dar para convertirlo en 1.
   - *(1 punto)* `lista-collatz` que recibe un número entero positivo y responde con una lista con las transformaciones que va sufriendo n hasta convertirse en 1.

   **Por ejemplo:**

```
> (pasos-collatz 12)
> 9
> (lista-collatz 12)
> '(12 6 3 10 5 16 8 4 2 1)
```

**Hasta dos puntos puntos extra.** Utiliza alguno de los operadores de plegado que definiste en el primer ejercicio para para implementar las siguientes funciones.
***Hint:** Utiliza lambdas.*

1. `(my-length lst) -> entero`
   `lst : lista`

2. `(my-reverse lst) -> lista`
   `lst : lista`

3. `(my-append lst1 lst2) -> lista`
   `lst1, lst2 : lista`

4. `(my-concatenate lst-de-lst) -> lista`
   `lst-de-lst: lista de listas.`

## Notas

- Asegurate de que cuando preguntes por una duda que requiera revisar código tu repositorio esté actualizado.
- Para dudas rápidas puedes encontrarme en Telegram.
- Esta práctica es individual, sin embargo las siguientes deberán ser en equipos de a lo más 4 personas y **no estarán permitidos los equipos de una sola persona.**