Información del curso

Lenguajes de Programación — Tarea 0

Eduardo Acuña Yeomans*

8 de agosto de 2022

1 Racket

En este curso utilizaremos Racket. Ingresa a su página oficial y descarga la última versión (al menos 8.5) para la plataforma de la computadora que utilizarás. Revisa la guía de Racket para aprender los aspectos básicos del lenguaje, estudia con cuidado las secciones:

- Welcome to Racket
- Racket Essentials
- Built-In Datatypes
- Expressions and Definitions
- Programmer-Defined Datatypes

Configura el editor de texto de tu preferencia para poder programar en Racket cómodamente. El editor con más soporte para el lenguaje es DrRacket, incluido en la instalación de Racket. Editores libres de propósito general recomendables que han sobrevivido los tiempos (casi medio siglo) son GNU Emacs y Vi (ver Vim). Puedes encontrar un resumen de su soporte para Racket aquí y acá. Otros editores menos ancestrales y más nuevos (y posiblemente más restrictivos para ti) también tienen soporte para Racket (ver VS Code y Sublime Text).

2 Git

Crea un repositorio de git con rama principal `main`, este repositorio va a contener todas las tareas que realizarás a lo largo del curso.

Hospeda tu repositorio de forma pública en algún servicio como Codeberg, sourcehut, GitLab o GitHub. Debes enviar la URL de tu repositorio a la dirección eduardo.acuna@unison.mx antes del 12 de Agosto del 2022.

Tu repositorio debe contener un directorio por cada tarea: `tarea-00`, `tarea-01`, `tarea-02`, etc. Tu trabajo en cada tarea debe no debe modificar los directorios de otras tareas. Pero eres libre de crear otros directorios y archivos en la raíz de tu repositorio. Antes de iniciar una tarea, crea una rama a partir de `main`, trabaja en tu solución y antes de la fecha límite de la tarea, actualiza tu rama `main` para reflejar tus cambios. Únicamente la rama `main` será revisada y evaluada.

*eduardo.acuna@unison.mx

3 TEX

A lo largo del curso deberás plasmar tus ideas en prosa en lugar de código. Instala alguna distribución de TEX (ver TEX Live y MiKTeX) y si no tienes experiencia preparando documentos con este programa o algún descendiente como LATEX o ConTeXt, aprende los aspectos básicos.

Deberás agregar a tu repositorio los archivos de TEX así como el documento final en formato PDF o PostScript.

El objetivo es que no tengas limitaciones para escribir un documento con TEX. Puedes evitar escribir TEX manualmente utilizando programas que generen TEX por ti (ya sean de tu autoría o de otros) siempre y cuando incluyas el documento original y las instrucciones para obtener el documento en TEX.

4 Sobre evaluación en el curso

En ocasiones, calificar tareas puede ser un proceso subjetivo y propenso a sesgos. El objetivo es minimizar estos dos factores y no perjudicar negativamente la evaluación de tus tareas. Sin embargo, se consideran algunos criterios que conllevan a la asignación de **0** en la calificación:

1. No se entregó antes de la fecha límite
2. El código no compila
3. El código no contiene pruebas
4. La prosa no es inteligible

En algunas ocasiones (y a discreción del docente), se calificarán tareas con a lo más dos días de retraso. No cuentes con esto, aún si no te fue posible terminar a tiempo tu tarea, deberás terminarla y actualizar tu repositorio de git.

Queda prohibido hacer trampa, en particular copiar total o parcialmente soluciones de otros estudiantes (o del internet) así como dejarse copiar total o parcialmente. Se asignará una calificación de **0** en estos casos. Excepciones a esto se contemplan únicamente si se admite la falla con el docente dentro de dos días de la fecha límite.

En la primer parte del curso es importante que te acostumbres al ritmo y estilo de las clases y las tareas, es por ello que se realizan las siguientes consideraciones:

- Debes entregar todas las tareas y deben ser correctas, pero la calidad y estilo de tus soluciones no cuenta para tu calificación.
- Puedes discutir abiertamente los problemas con quien sea.
- Las tareas con a lo más dos días de retraso serán aceptadas sin una penalización fuerte en la calificación.

En la segunda parte del curso, puedes elegir trabajar con solo una persona más, en este caso deberás especificarlo en la entrega de tus tareas. También se realizan las siguientes consideraciones:

- La calidad y estilo de tus soluciones se considera un factor importante de tu calificación.
- Puedes discutir problemas con otras personas, pero la solución tiene que ser tuya o de tu pareja de trabajo.
- Las tareas deben ser entregadas antes de la fecha límite.

En la tercera y última parte del curso, todas las tareas las deberás trabajar por tu cuenta y no serán calificados trabajos entregados después de la fecha límite. A lo largo del curso se indicará en clase a partir de qué día comienza el segundo y tercer periodo.

5 Problemas

Resuelve los siguientes problemas en un archivo llamado `problemas.rkt`. Asegúrate que tu código sea correcto escribiendo un archivo de pruebas `pruebas.rkt`. Se proporcionan dos archivos como plantillas, sin embargo, las pruebas proporcionadas son insuficientes para una solución satisfactoria de la tarea.

1. Define una variable `pi` como `3.14`
2. Define una función para calcular el área del círculo dado su radio:

```
(test-case "area-circle"
  (check-eqv? (area-circle 5) 78.5))
```

3. Define una función que regrese el área y la circunferencia del círculo como una lista dado su radio:

```
(test-case "circle-properties"
  (check-within (circle-properties 5) '(78.5 31.4) 0.001))
```

4. Define un procedimiento que tome una lista con el largo y ancho de un rectángulo y que regrese una lista de su área y perímetro:

```
(test-case "rectangle-properties"
  (check-equal? (rectangle-properties '(2 4)) '(8 12)))
```

5. Define una función `find-needle` que encuentra el índice de `'needle` dada una lista de longitud 3 que contiene los símbolos `'needle` o `'hay`:

```
(test-case "find-needle"
  (check-eqv? (find-needle '(hay needle hay)) 1)
  (check-eqv? (find-needle '(hay hay hay)) -1))
```

6. Define un procedimiento que tome un número y regrese su valor absoluto:

```
(test-case "abs"
  (check-eqv? (abs 3) 3)
  (check-eqv? (abs -2) 2))
```

7. Usa la función de Racket `map` para incrementar los elementos de una lista en uno. Por ejemplo, dado `'(1 2 3)` regresa `'(2 3 4)`

```
(test-case "inclis1"
  (check-equal? (inclis1 '(1 2 3)) '(2 3 4)))
```

8. Define una función `even?` que al pasarla a `map` junto con una lista de números regrese una lista de `#t` o `#f`:

```
;; 8.
(define (even? x)
  ...)
```

```
(test-case "even?"
  (check-equal? (map even? '(1 2 3 4 5 6)) '(#f #t #f #t #f #t)))
```

9. Considera la siguiente función `add` y define `another-add` cuya condición de paro sea que el primer argumento es cero en lugar del segundo:

```
(define add
  (lambda (n m)
    (cond
      [(zero? m) n]
      [else (add1 (add n (sub1 m)))])))
```

```
(test-case "another-add"
  (check-eqv? (another-add 10 5) 15))
```

10. Revisa la guía de estilo de Racket y modifica tu código en caso de ser necesario.