Taller 2

August 31, 2016

El presente taller evalua el proceso de aprendizaje alrededor de los temas de:

- Control de versiones: Git.
- Algebra computacional: Sympy.

Tenga en cuenta, que aunque el objetivo evaluativo esta en estos dos temas, se hace necesario del uso de habilidades obtenidades en los temas anteriores.

Este taller se trabajará en equipos, los cuales deben implementar un pequeño sistema de algebra computacional con soporte de operaciones aritmeticas y otras propiedades para funciones polinomicas en una variable.

1 Condiciones de entrega

Tenga presente cada uno de los siguientes aspectos para su entrega.

1.1 Plazo máximo de entrega y forma de entrega

La entrega se realiza como un repositorio público de github, y este se debe registrar en el archivo de `historial_talleres.md` del repositorio del curso. Para realizar este registro debe realizar una solicitud de PR en la cual solo se realizo la modificación de dicho archivo acorde a las indicaciones que encontrará en él.

El último `commit` que tendrá efecto es el último antes de la media noche del 20 de septiembre de 2016.

1.2 Equipo

Se deben organizar 2 o 3 equipos con 3 a 5 integrantes, de donde se elige uno que alojará en su usuario github el repositorio de la implementación. Todos los integrantes deben participar directamente con el repositorio y ser responsables de al menos una caracteristica de la implementación o del desarrollo de la documentación.

1.3 Repositorio

Elegido el usuario que tendrá el alojamiento del repositorio, deben crear el repositorio público con el nombre que deseen dar a su implementación.

El repositorio debe contener:

- Archivo de licencia: Especifique un tipo de licencia libre para su implementación.

- Archivo `README.md`: Debe contener una descripción de su implementación, forma general de instalación y uso, así como asociar la finalidad como actividad del curso de herramientas del pregrado de computación científica, enlazando a la página oficial del pregrado. En este mismo archivo se debe mencionar a los integrantes enlazando a sus cuentas de github y asociando sus responsabilidades principales en la implementación.

- Implementación en python: Puede usar python 2 o 3 según sea de su agrado. En caso de usar python 2 utilice las opciones de compatibilidad con python 3. Su desarrollo puede ser tan modular como lo desee o requiera.

- Documentación en *notebook*: Debe documentarse adecuadamente con sus casos de prueba que no solo explicarán el funcionamiento sino que serán prueba del correcto funcionamiento de la implementación. La documentación debe recordar su forma de uso (explicado en `README.md`). Los casos de prueba deben contrastarse con los resultados de `sympy` y de la evaluación numérica con `math` (según corresponda).

- Archivos de texto plano: Si llega a requerir de archivos de texto plano adicionales para configuración o almacenamiento de reglas, guardelos con extensión igual al nombre de su repositorio github (nombre de su implementación).

2 Mini-CAS

2.1 Sintaxis

La utilidad CAS a implementar debe usar la sintaxis matematica de python. La precedencia/orden de los operadores puede ser tomada de python o basada en scheme para las expresiones matemáticas, y estas deberán ser incluidas como cadena de caracteres.

```
expresion = '2 * x**3 + (x - a)**2 / 2'
```

o

```
expresion = '+ * 2 ** x 3 / ** - x a 2 2'
```

Por conveniencia, si usa la sintaxis de scheme, es conveniente usar agrupación para notar más claramente la aplicación de los operadores aunque no es necesario dada la definición de la aplicación de los operadores (unarios y binarios).

```
expresion = '(+ (* 2 (** x 3)) (/ (** (- x a) 2) 2))'
```

Por la indicación del uso de la sintaxis python se reemplazo `expt` por su equivalente python `**`.

2.2 Implementación

2.2.1 Representación de datos

Su implementación debe permitir la representación de:

- Valores numéricos reales: La representación de los reales se divide en las siguientes categorías que debe permitir.
  - Racionales: Representación convencional en formato decimal (forma extensiva) y con separador decimal anglosajón.
    * Enteros: El tipo de dato entero debe ser de longitud arbitraria. Equivalente al entero largo de python.
    * Fraccionarios: Representación como el cociente de dos enteros.
  - Irracionales: Representación como real elevado a una potencia real.

- Variables: Todo caracter individual corresponde a una variable. Por defecto, las caracteristicas que se implementarán deben usar la variable `x` como referencia.

- Operadores: `+`, `-`, `*`, `/`, `**`.

- Agrupación por parentesis: No se requiere dar soporte a agrupaciones dentro de agrupaciones.

- Expresiones algebraicas/polinomicas: Se forman por la unión de los elementos anteriores.

A partir de ahora, las menciones de los objetos representados se refieren a las definiciones dadas en esta sección.

2.2.2 Aritmetica de enteros

Su implementación debe soportar con representación exacta las operaciones entre enteros acorde a las siguientes reglas.

- `+`, `-`, `*`, `**`: Deben generar adecuadamente los valores enteros de longitud arbitraria que representen exactamente la aplicación del operador.

- `/`: Debe generar el valor entero o racional de longitud arbitraria requerida para representar exactamente la aplicación del operador, solo si este puede ser expresado extactamente con máximo 10 digitos. En caso contrario debe expresarse como fraccionario.

2.2.3 Aritmetica de fraccionarios

Se debe dar soporte a los 5 operadores indicados acorde a las reglas aritmeticas de los fraccionarios. Los resultados deben tener en cuenta el siguiente comportamiento por defecto.

- Una fracción debe estar constituida por números enteros en su numerador y denominador.

- Una fracción debe estar en su forma reducida/simplificada.

- La operación de un fraccionario y con un entero genera un fraccionario.

- Una fracción en la cual su denominador sea 1 se simplifica a un entero.

- Una fracción en la cual su numerador sea 0 se simplifica al entero nulo.

2.2.4 Aritmetica de racionales

Su implementación debe soportar con representación exacta las operaciones entre racionales acorde a las siguientes reglas.

- Ceros finales del racional se eliminan.

- La operación de un racional y un fraccionario queda a criterio del equipo su representación.

- $+$, $-$, $*$: Deben generar adecuadamente los valores racionales de longitud arbitraria que representen exactamente la aplicación del operador.

- $/$: Debe generar el valor entero o racional de longitud arbitraria requerida para representar exactamente la aplicación del operador, solo si este puede ser expresado extactamente con máximo 10 digitos. En caso contrario debe expresarse como fraccionario.

- $**$: Debe generar el valor entero o racional de longitud arbitraria requerida para representar exactamente la aplicación del operador acorde a los siguientes casos.
  - Si su segundo operando es 0.5 o $1/2$, y:
    - Si su primer operando es un número entero positivo: Genera el número entero si su raíz es exacta en los enteros. En caso contrario debe dejar indicada la potencia.

    - Si su primer operando es un racional positivo menor que 1: Genera el número racional que representa exactamente la aplicación del operador, solo si este puede ser expresado exactamente con máximo 10 digitos. En caso contrario debe dejar indicada la potencia.

  - Si su segundo operando es entero: Genera el número racional que representa exactamente la aplicación del operador, solo si este puede ser expresado exactamente con máximo 10 digitos. En caso contrario debe dejar indicada la potencia.

2.2.5 Aritmetica de irracionales

Su implementación debe soportar con representación exacta las operaciones entre racionales acorde a las siguientes reglas.

- Toda operación que involucre un irracional, debe dejar indicada la operación con el irracional.

2.2.6 Algebra de polinomios

Su implementación debe soportar las siguientes operaciones y funciones:

- Operaciones de variables: Debe cumplir con las reglas algebraicas de los 5 operadores.

  - Las operaciones algebraicas que no se pueden simplicar se deben dejar indicadas.

- `+`, `-`, `*`: Polinomio-Polinomio. Genera un polinomio simplificado con representación exacta con base a los comportamientos definidos anteriormente. La agrupación de terminos se realiza respecto a la variable `x`.

- `/`: Polinomio-Constante (numérica o algebraica). Genera un polinomio simplificado con representación exacta con base a los comportamientos definidos anteriormente. La agrupación de terminos se realiza respecto a la variable `x`.

- `**`: Polinomio-Entero positivo. Genera un polinomio simplificado con representación exacta con base a los comportamientos definidos anteriormente. La agrupación de terminos se realiza respecto a la variable `x`.

2.2.7 Funciones

Para la interacción del usuario se deben definir mínimo las siguientes funciones.

- `ispoly`: Evalua si la expresión ingresada es un polinomio en la variable `x`. Es valido como polinomio el grado 0.

- `isnumeric`: Evalua si la expresión ingresada es numerica.

- `ispolynum`: Evalua si la expresión ingresada es un polinomio en la variable `x` con coeficientes numéricos.

- `simplify`: Dada una expresión algebraica aplica los operadores indicados acorde a los comportamientos definidos anteriormente.

- `subs`: Permite substituir una variable por una expresión algebraica. Por defecto, tras la sustitución debe aplicar `simplify`.

- `roots`: Si la expresión es un polinomio de segundo o primer grado, debe indicar un arreglo con las raíces exactas. Si la expresión es un coeficiente debe devolver un arreglo nulo. Si el polinomio es de grado mayor a dos debe devolver el tipo de dato nulo.

- `eval`:
  - Si la expresión es numérica, evalua invocando el modulo `math`.
  - Si la expresión es un polinomio con coeficientes numéricos, recibe un segundo argumento numérico y evalua el polinomio en el punto usando el modulo `math`.

El equipo define la forma de creación de expresiones que operan expresiones ya existentes.

3 Evaluación

3.1 Componente individual

(0,5 unidades) Cada integrante del equipo debe ser responsable de al menos una caracteristica evaluable (ver lista de evaluación grupal). Deberá tener al menos un `commit` semanal asociado a dicha responsabilidad, y uno asociado a una caracteristica que no lo corresponda (no puede ser un comentario ni corrección de estilo).

3.2 Componente grupal

- Archivo `README.md`: 0,2 unidades.
- Archivo `LICENSE`: 0,1 unidades.
- Notebook con documentación (incluidos casos de prueba): 0,5 unidades.
- Implementación:
  - Funciones `isxxx`: 0,3 unidades.
  - Aritmetica de enteros: 0,6 unidades.
  - Aritmetica de fraccionarios: 0,5 unidades.
  - Aritmetica de racionales: 0,6 unidades.
  - Aritmetica de irracionales: 0,5 unidades.
  - Algebra de polinomios: 0,6 unidades.
  - Función `roots`: 0,2 unidades.
  - Función `subs`: 0,2 unidades.
  - Función `eval`: 0,2 unidades.