Introducción a la programación
Introducción al entorno Haskell

Taller de Álgebra I

Segundo cuatrimestre de 2013

Taller de Álgebra I

- Horarios:
  1. Martes de 09:00 a 12:00
  2. Miércoles de 17:30 a 20:30
  3. Viernes de 15:00 a 18:00

Taller de Álgebra I

- Horarios:
  1. Martes de 09:00 a 12:00
  2. Miércoles de 17:30 a 20:30
  3. Viernes de 15:00 a 18:00

- Profesores:
  1. Flavia Bonomo (`fbonomo@dc.uba.ar`)
  2. Javier Marenco (`jmarenco@dc.uba.ar`)
  3. Esteban Mocskos (`emocskos@dc.uba.ar`)

Taller de Álgebra I

- Horarios:
  1. Martes de 09:00 a 12:00
  2. Miércoles de 17:30 a 20:30
  3. Viernes de 15:00 a 18:00
- Profesores:
  1. Flavia Bonomo (`fbonomo@dc.uba.ar`)
  2. Javier Marenco (`jmarenco@dc.uba.ar`)
  3. Esteban Mocskos (`emocskos@dc.uba.ar`)
- Ayudantes:
  1. Pablo Brusco
  2. Federico Lebrón
  3. Mariano Rean
  4. Xavier Warnes

Taller de Álgebra I

- De qué se trata el taller de Álgebra I?
  1. Dar una introducción a la computación y a la programación, apta tanto para estudiantes de computación como para estudiantes de matemática.
  2. Implementar y probar los algoritmos que se ven en las clases teóricas y prácticas.
  3. Dar una introducción a la programación funcional.

Taller de Álgebra I

- De qué se trata el taller de Álgebra I?
  1. Dar una introducción a la computación y a la programación, apta tanto para estudiantes de computación como para estudiantes de matemática.
  2. Implementar y probar los algoritmos que se ven en las clases teóricas y prácticas.
  3. Dar una introducción a la programación funcional.
- Clases teórico-prácticas todas las semanas del cuatrimestre, dictadas en laboratorios de computación.
- Aproximadamente dos horas de clase y una hora de consultas por turno.

Régimen de cursada y evaluación

- Régimen de evaluación: Dos trabajos prácticos en grupos de dos personas.
  - Deben estar aprobados los dos trabajos prácticos para aprobar el taller.
  - Cada trabajo práctico tiene su recuperatorio, que consiste en entregar el trabajo corregido.
- No es necesario aprobar el taller de Álgebra I para anotarse en las materias correlativas.
- Sí es necesario aprobar el taller de Álgebra I para rendir el final de Álgebra I!

Qué es una computadora?

Qué es una computadora?

- Una **computadora** es una máquina que procesa información automáticamente de acuerdo con un programa almacenado.
  Claude Bellavoine, *¿Qué es una computadora?* Editorial El Ateneo, 1969.

Qué es una computadora?

- Una **computadora** es una máquina que procesa información automáticamente de acuerdo con un programa almacenado.
  Claude Bellavoine, *¿Qué es una computadora?* Editorial El Ateneo, 1969.

  1. Es una máquina.

Qué es una computadora?

- Una **computadora** es una máquina que procesa información automáticamente de acuerdo con un programa almacenado.
  Claude Bellavoine, *¿Qué es una computadora?* Editorial El Ateneo, 1969.

  1. Es una máquina.
  2. Su función es procesar información, y estos términos deben entenderse en sentido amplio.

Qué es una computadora?

- Una **computadora** es una máquina que procesa información automáticamente de acuerdo con un programa almacenado.
  Claude Bellavoine, *¿Qué es una computadora?* Editorial El Ateneo, 1969.

  1. Es una máquina.
  2. Su función es procesar información, y estos términos deben entenderse en sentido amplio.
  3. El procesamiento se realiza en forma automática.

Qué es una computadora?

- Una **computadora** es una máquina que procesa información automáticamente de acuerdo con un programa almacenado.
  Claude Bellavoine, *¿Qué es una computadora?* Editorial El Ateneo, 1969.

  1. Es una máquina.
  2. Su función es procesar información, y estos términos deben entenderse en sentido amplio.
  3. El procesamiento se realiza en forma automática.
  4. El procesamiento se realiza siguiendo un programa.

Qué es una computadora?

- Una **computadora** es una máquina que procesa información automáticamente de acuerdo con un programa almacenado.
  Claude Bellavoine, *¿Qué es una computadora?* Editorial El Ateneo, 1969.

  1. Es una máquina.
  2. Su función es procesar información, y estos términos deben entenderse en sentido amplio.
  3. El procesamiento se realiza en forma automática.
  4. El procesamiento se realiza siguiendo un programa.
  5. Este programa está almacenado en una memoria interna de la misma computadora.

Resolviendo problemas con una computadora

- La resolución de un problema usando una computadora requiere de al menos los siguientes pasos:
  1. Identificar el problema a resolver, idealmente dando una especificación formal del mismo.
  2. Pensar un algoritmo para resolver el problema.
  3. Implementar el algoritmo en un lenguaje de programación y en una plataforma determinada, obteniendo así un programa ejecutable.

Ejemplo

- **Problema:** Dados dos números naturales representados en notación decimal, encontrar la suma de los números.

Ejemplo

- **Problema:** Dados dos números naturales representados en notación decimal, encontrar la suma de los números.
- Podemos pensar varios algoritmos para resolver este problema.

Ejemplo

- **Problema:** Dados dos números naturales representados en notación decimal, encontrar la suma de los números.
- Podemos pensar varios algoritmos para resolver este problema.
  1. **Algoritmo escolar**: Sumo las unidades del primero a las del segundo, después las decenas, etc ("llevándome uno de acarreo" cuando hace falta)

Ejemplo

- **Problema:** Dados dos números naturales representados en notación decimal, encontrar la suma de los números.
- Podemos pensar varios algoritmos para resolver este problema.
  1. **Algoritmo escolar**: Sumo las unidades del primero a las del segundo, después las decenas, etc ("llevándome uno de acarreo" cuando hace falta)
  2. **Algoritmo sucesor**: Voy sumando 1 al primero y restando uno al segundo, hasta que el segundo llegue a 0.

Ejemplo

- **Problema:** Dados dos números naturales representados en notación decimal, encontrar la suma de los números.
- Podemos pensar varios algoritmos para resolver este problema.

1. **Algoritmo escolar**: Sumo las unidades del primero a las del segundo, después las decenas, etc ("llevándome uno de acarreo" cuando hace falta)
2. **Algoritmo sucesor**: Voy sumando 1 al primero y restando uno al segundo, hasta que el segundo llegue a 0.
3. **Algoritmo "rincón del vago"**: Entro a google.com, escribo el primer número, luego el signo "+", luego escribo el segundo y luego aprieto "Voy a tener suerte".

Ejemplo

- Los tres algoritmos difieren en los pasos primitivos usados para expresarlos:
  1. sumar números de un dígito cada uno, contemplar el acarreo, concatenar resultados, etc.
  2. sumar y restar 1, y comparar contra el 0.
  3. Usar teclado y mouse de una computadora conectada a Internet.
- Cuál algoritmo tiene más sentido/conviene usar?

Programas

- Un programa es la descripción de un algoritmo en un lenguaje de programación.
  1. Es una descripción precisa, de modo tal que pueda ser ejecutada por una computadora.
  2. Un lenguaje de programación tiene típicamente una sintaxis y una semántica bien definidas.
- Cuando se implementa un programa, las siguientes cuestiones son de interés:
  1. El programa es correcto?
  2. Implementa adecuadamente el algoritmo propuesto?
  3. Puede pasar que el programa no termine?
  4. Qué datos son válidos para ejecutar el programa?
  5. Cuánto va a tardar la ejecución?
  6. Está bien resuelto el problema original?

Etapas en el desarrollo de un programa

1. **Diseño**: Pensar la estructura del programa, dividir la solución en módulos, analizar la interacción entre estos módulos, etc.
2. **Codificación**: Escribir el código en un lenguaje de programación.
3. **Validación**: Determinar si el programa cumple con lo especificado.
4. **Mantenimiento**: Corregir errores y adaptar el programa a nuevos requerimientos.

0. Especificación

- El planteo inicial del problema puede ser vago y ambiguo. Al especificar damos una descripción clara y precisa, idealmente en un lenguaje formal.
- **Ejemplo:** "Calcular de edad de una persona" es una especificación vaga, porque:
  1. ¿Cuáles son los datos de entrada?
  2. ¿Qué forma van a tener? (días, años enteros, años fraccionarios, etc.)
  3. ¿Cómo recibiremos los datos?
  4. ¿Cómo debemos retornar el resultado?
- Al especificar formalizamos estos aspectos. Por ejemplo, "*Necesito una función que, dadas dos fechas en formato dd/mm/aaaa, calcule la cantidad de días que hay entre ellas, sin incluir la segunda*"

1. Diseño

- ¿Qué lenguaje de programación vamos a usar? ¿Sobre qué plataforma?
- ¿Varios programas o uno muy complejo?
  1. ¿Cómo dividirlo en partes, qué porción del problema resuelve cada una?
  2. ¿Distintas partes en distintas computadoras?
  3. ¿Cómo se especifica cada parte?
  4. ¿Cada programador recibe una sola parte? ¿Cómo se organiza el trabajo de desarrollo?
- ¿Existen programas ya hechos con los que debamos interactuar?
- ...

2. Codificación

- Esta etapa consiste en escribir un programa de acuerdo con el diseño de la etapa anterior.
- Asumiendo que están definidos los pasos primitivos, los combinamos para llegar al resultado buscado de manera efectiva.
- Debemos traducir el algoritmo (escrito o idea) para que una computadora lo entienda.
- En el taller de Álgebra I vamos a usar un lenguaje de programación funcional llamado **Haskell**.

Haskell: programación funcional

Haskell B. Curry (1900–1982)

- Existen muchos otros lenguajes de programación y otros paradigmas de lenguajes de programación, cada uno adaptado a diferentes situaciones.

3. Validación

- Estas actividades buscan asegurar que un programa cumple con la especificación.

3. Validación

- Estas actividades buscan asegurar que un programa cumple con la especificación.
- **Testing:** Probar el programa con muchos datos y ver que en todos los casos se comporte de manera adecuada.

3. Validación

- Estas actividades buscan asegurar que un programa cumple con la especificación.
- **Testing:** Probar el programa con muchos datos y ver que en todos los casos se comporte de manera adecuada.
  1. Para estar seguro de que anda bien, habría que probarlo con todos los datos posibles de entrada.
  2. Si hay un error, con algo de suerte se lo puede detectar.
  3. No es infalible (un programa puede pasar el testing pero puede haber errores).

3. Validación

- Estas actividades buscan asegurar que un programa cumple con la especificación.
- **Testing:** Probar el programa con muchos datos y ver que en todos los casos se comporte de manera adecuada.
  1. Para estar seguro de que anda bien, habría que probarlo con todos los datos posibles de entrada.
  2. Si hay un error, con algo de suerte se lo puede detectar.
  3. No es infalible (un programa puede pasar el testing pero puede haber errores).
- **Verificación formal:** Demostrar que el programa cumple con la especificación.

3. Validación

- Estas actividades buscan asegurar que un programa cumple con la especificación.
- **Testing:** Probar el programa con muchos datos y ver que en todos los casos se comporte de manera adecuada.
  1. Para estar seguro de que anda bien, habría que probarlo con todos los datos posibles de entrada.
  2. Si hay un error, con algo de suerte se lo puede detectar.
  3. No es infalible (un programa puede pasar el testing pero puede haber errores).
- **Verificación formal:** Demostrar que el programa cumple con la especificación.
  1. Requiere que el lenguaje de especificación sea formal y admita una semántica bien definida.
  2. Es infalible (si se demuestra la corrección, entonces el programa no tiene errores).
  3. En general requiere más tiempo que las actividades de testing.

4. Mantenimiento

- Tiempo después de implementado el programa, encontramos problemas o errores ...
  1. El programa no cumplía la especificación.
  2. La especificación no describía correctamente el problema
  3. Cometimos errores en la validación, o el testing no detectó los problemas.
- ... o cambian los requerimientos.
- El mantenimiento justifica las etapas anteriores. Si se hicieron bien, entonces las modificaciones serán más sencillas y de bajo impacto.

Lenguajes de programación

- ¿Cómo es un lenguaje de programación?
- ¿Cómo se le dice a una computadora lo que tiene que hacer?

Paradigmas de lenguajes de programación

- **Paradigma:** Definición del modo en el que se especifica el cómputo (que luego es implementado a través de programas).
  1. Representa una "toma de posición" ante la pregunta: ¿cómo se le dice a la computadora lo que tiene que hacer?.
  2. Todo lenguaje de programación pertenece a un paradigma.

Paradigmas de lenguajes de programación

- **Paradigma:** Definición del modo en el que se especifica el cómputo (que luego es implementado a través de programas).
  1. Representa una "toma de posición" ante la pregunta: ¿cómo se le dice a la computadora lo que tiene que hacer?.
  2. Todo lenguaje de programación pertenece a un paradigma.

- Estado del arte:
  1. Paradigma de programación imperativa: C, Basic, Ada, Clu

Paradigmas de lenguajes de programación

- **Paradigma:** Definición del modo en el que se especifica el cómputo (que luego es implementado a través de programas).
  1. Representa una "toma de posición" ante la pregunta: ¿cómo se le dice a la computadora lo que tiene que hacer?.
  2. Todo lenguaje de programación pertenece a un paradigma.

- Estado del arte:
  1. Paradigma de programación imperativa: C, Basic, Ada, Clu
  2. Paradigma de programación en objetos: Smalltalk

Paradigmas de lenguajes de programación

- **Paradigma:** Definición del modo en el que se especifica el cómputo (que luego es implementado a través de programas).
  1. Representa una "toma de posición" ante la pregunta: ¿cómo se le dice a la computadora lo que tiene que hacer?.
  2. Todo lenguaje de programación pertenece a un paradigma.

- Estado del arte:
  1. Paradigma de programación imperativa: C, Basic, Ada, Clu
  2. Paradigma de programación en objetos: Smalltalk
  3. Paradigma de programación orientada a objetos: C++, C#, Java

Paradigmas de lenguajes de programación

- **Paradigma:** Definición del modo en el que se especifica el cómputo (que luego es implementado a través de programas).
  1. Representa una "toma de posición" ante la pregunta: ¿cómo se le dice a la computadora lo que tiene que hacer?.
  2. Todo lenguaje de programación pertenece a un paradigma.

- Estado del arte:
  1. Paradigma de programación imperativa: C, Basic, Ada, Clu
  2. Paradigma de programación en objetos: Smalltalk
  3. Paradigma de programación orientada a objetos: C++, C#, Java
  4. Paradigma de programación funcional: LISP, F#, Haskell

Paradigmas de lenguajes de programación

- **Paradigma:** Definición del modo en el que se especifica el cómputo (que luego es implementado a través de programas).
  1. Representa una "toma de posición" ante la pregunta: ¿cómo se le dice a la computadora lo que tiene que hacer?.
  2. Todo lenguaje de programación pertenece a un paradigma.

- Estado del arte:
  1. Paradigma de programación imperativa: C, Basic, Ada, Clu
  2. Paradigma de programación en objetos: Smalltalk
  3. Paradigma de programación orientada a objetos: C++, C#, Java
  4. Paradigma de programación funcional: LISP, F#, Haskell
  5. Paradigma de programación en lógica: Prolog