Algoritmos y Estructuras de Datos I

Segundo cuatrimestre de 2025

Departamento de Computación - FCEyN - UBA

Solucionando problemas con una computadora

IP - AED I: Temario de la clase

- Modalidad de la materia, Régimen de aprobación
- Introducción
  - Objetivos de la materia.
  - Noción de Problema, especificación, algoritmo, programa
  - Relaciones entre conceptos.
- Especificación
  - Qué vs Cómo.
  - Lenguaje naturales vs Lenguajes formales.
  - Noción de contrato.
  - Estructura de una especificación (precondiciones y postcondiciones).
  - Parámetros y tipos de datos.
- Algoritmos y programas
  - Lenguajes de programación.
  - Código fuente, compiladores, intérpretes.
  - Entorno de desarrollo integrado (IDE).
  - Paradigmas de Programación.
  - Buenas prácticas de programación.
  - Sistemas de control de versiones (Git).

IP - AED I: Antes de empezar

- Sobre la modalidad de la materia:
  - La materia es presencial. Las clases son en aulas y laboratorios.
  - Clases Teóricas (jueves, TM 9:30 a 13:30 hs, TT de 13:30 a 17:30 TN 17:30 a 21:30 hs)
  - Clases Laboratorio (lunes y miércoles, TM 11:00 a 13:30 hs, TT 14:30 a 17:00 hs, TN 19:30 a 22 hs)
  - Cada Turno tiene un JTP a cargo (ver Campus).
- Turnos
  - Revisen sus mails porque es la vía principal de comunicación (a través del Campus).
  - Tienen que tener acceso al Campus.
  - No se pueden cambiar de turno.
  - Casos particulares, con justificaciones, certificados, etc... avisen cuanto antes
    - Por mail: marianogonzalez@dc.uba.ar (asistente académico)
    - Por campus
    - Al JTP responsable del turno.

IP - AED I: Régimen de aprobación

- Evaluaciones:
  - Parcial individual de programación en Haskell en computadora (nota numérica, se aprueba con 6).
  - Parcial individual de programación en Python en computadora (nota numérica, se aprueba con 6).
  - Un TP grupal de programación en Python + Testing (Aprobado/Desaprobado)
  - Cada instancia de evaluación tiene una instancia de recuperación.
- Criterio de aprobación de la materia:
  - TP aprobado y notas en ambos parciales mayor o igual que 8: promoción directa, queda el promedio de notas.
  - TP aprobado y notas en ambos parciales mayor o igual que 7: final oral (coloquio).
  - La instancia de coloquio, sólo es válida hasta las mesas de finales de Diciembre 2025. Luego de esas fechas, se deberá dar final escrito.
  - TP aprobado y notas en ambos parciales mayor o igual que 6: final escrito (tienen 8 cuatrimestres para rendir el final, no es recomendable dejarlo 'colgado").
  - Se pueden presentar a recuperatorio para "levantar" nota, pero tener en cuenta que la nota del recuperatorio pisa la nota del parcial.

IP - AED I: Grupos

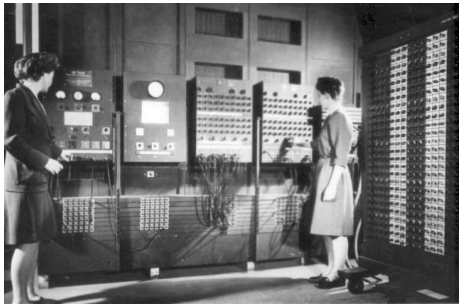
- Tienen que armar grupos para hacer el TP
  - Grupos de estudiantes (la cantidad le informarán en los Laboratorios)
  - Todos los integrantes deben estar en el mismo turno
  - ¿Cómo consigo compañeros de grupo?
    - Hablando entre ustedes
- El TP se entregan a través de Git (sistema de control de versiones)
  - Hay un Taller para aprender Git.
  - Al menos un integrante del grupo debería hacerlo!

Introducción a la Programación - AED I

Objetivo: Aprender a programar en lenguajes funcionales y en lenguajes imperativos.

- Especificar problemas.
  - Describirlos de manera tal que podemos construir y probar una solución
- Pensar algoritmos para resolver los problemas.
  - En esta materia nos concentramos en problemas para tratamiento de secuencias principalmente.
- Empezar a Razonar acerca de estos algoritmos y programas.
  - Veremos cómo hacer testing para verificar los algoritmos.
  - Veremos cómo contar la cantidad de operaciones que demanda un algoritmo.
- Escribir programas que implementen los algoritmos que pensamos.
  - Vamos a usar dos lenguajes de programación bien distintos para esto.

¿Qué es una computadora?



- Una **computadora** es una máquina electrónica que procesa datos automáticamente de acuerdo con un programa almacenado en memoria.
  - Es una máquina electrónica.
  - Su función es procesar datos.
  - El procesamiento se realiza en forma automática.
  - El procesamiento se realiza siguiendo un programa.
  - Este programa está almacenado en una memoria interna.

¿Qué es un algoritmo?

- Un **algoritmo** es la descripción de los pasos precisos para resolver un problema a partir de datos de entrada adecuados.
  1. Es la descripción de los pasos a realizar.
  2. Especifica una sucesión de instrucciones primitivas.
  3. El objetivo es resolver un problema.
  4. Un algoritmo típicamente trabaja a partir de datos de entrada.

Ejemplo: Un Algoritmo

- **Problema:** Encontrar todos los números primos menores que un número natural dado n
- **Algoritmo:** Criba de Eratóstenes (276 AC - 194 AC)
  Escriba todos los números naturales desde 2 hasta a n
  Para $i \in \mathbb{Z}$ desde 2 hasta $\lfloor \sqrt{n} \rfloor$
  Si i no ha sido tachado, entonces
  Para $j \in \mathbb{Z}$ desde i hasta $\lfloor \frac{n}{i} \rfloor$ haga lo siguiente:
  Si no ha sido tachado, tachar el número $i \times j$
- **Resultado:** Los números que no han sido tachados son los números primos menores a n

¿Qué es un programa?

- Un **programa** es la descripción de un algoritmo en un lenguaje de programación.
  1. Corresponde a la implementación concreta del algoritmo para ser ejecutado en una computadora.
  2. Se describe en un lenguaje de programación.

Ejemplo: Un Programa (en Haskell)

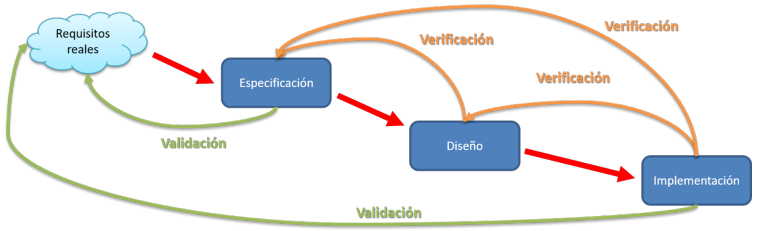
Implementación de la Criba de Eratóstenes en Haskell

```
eratostenes :: Int → [Int]
eratostenes n = eratostenesAux [2..n] (truncate(sqrt(fromIntegral n))) n

eratostenesAux :: [Int] → Int → Int→ [Int]
eratostenesAux lista 1 n = lista
eratostenesAux lista i n | elem i lista = eratostenes_aux (eliminar lista i i (div n i)) (i−1)
                         | otherwise = eratostenes_aux lista (i−1) n

eliminar :: [Int] → Int → Int → Int → [Int]
eliminar lista i j n | j > n = lista
                     | otherwise = eliminar (sacarElem lista (i∗j)) i (j+1) n

sacarElem :: [Int] → Int → [Int]
sacarElem [] elem = []
sacarElem (x:xs) elem | x == elem = xs
                      | otherwise = x:(sacarElem xs elem)
```


Especificación, algoritmo, programa

1. **Especificación:** descripción del problema a resolver.
   - ¿Qué problema tenemos?
   - Habitualmente, dada en lenguaje formal.
   - Es un contrato que da las propiedades de los datos de entrada y las propiedades de la solución.
2. **Algoritmo:** descripción de la solución escrita para humanos.
   - ¿Cómo resolvemos el problema?
   - Puede existir sin una computadora.
3. **Programa:** descripción de la solución para ser ejecutada en una computadora.
   - También, ¿cómo resolvemos el problema?
   - Pero descripto en un lenguaje de programación.
   - Requiere una computadora para ejecutarse.

Problema, especificación, algoritmo, programa



Dado un problema a resolver (de la vida real), queremos:

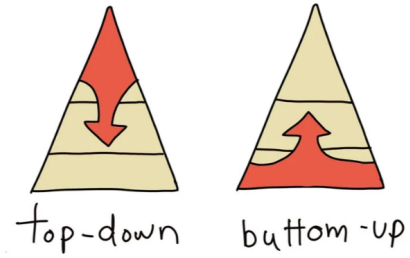
- Poder describir de una manera clara y unívoca (especificación)
  - Esta descripción debería poder ser validada contra el problema real
- Poder diseñar una solución acorde a dicha especificación
  - Este diseño debería poder ser verificado con respecto a la especificación
- Poder implementar un programa acorde a dicho diseño
  - Este programa debería poder ser verificado con respecto a su especificación y su diseño
  - Este programa debería ser la solución al problema planteado

También hablaremos de cómo encarar problemas...

O partir el problema en problemas más chicos...

Los conceptos de modularización y encapsulamiento siempre estarán relacionados con los principios de diseño de software. La estrategia se puede resumir en:

- Descomponer un problema grande en problemas más pequeños.
- Componerlos y obtener la solución al problema original.
- Estrategias *Top Down* versus *Bottom Up*.



Diferenciaremos el QUÉ del CÓMO

- Dado un problema, será importante describirlo sin ambigüedades.
- Una buena descripción no debería condicionarse con sus posibles soluciones.
- Saber que dado un problema, hay muchas formas de describirlo y a su vez, muchas formas de solucionar... y todas pueden ser válidas!

Especificación de problemas

- Una especificación es un contrato que define qué se debe resolver y qué propiedades debe tener la solución.
  - Define el qué y no el cómo.
- La especificación de un problema incluye un conjunto de parámetros: datos de entrada cuyos valores serán conocidos recién al ejecutar el programa.
- Además de cumplir un rol "contractual", la especificación del problema es insumo para las actividades de ...
  - Testing,
  - Verificación formal de correctitud.
  - Derivación formal (construir un programa a partir de la especificación).

Lenguaje naturales y lenguajes formales

- Lenguajes naturales
  - Idiomas (castellano)
  - Mucho poder expresivo (modos verbales –potencial, imperativo–, tiempos verbales –pasado, presente, futuro—, metáforas, etc.)
  - Con un plus (conocimiento del contexto, suposiciones, etc)
  - Pueden traer inconvenientes para especificar problemas porque pueden ser ambiguos
  - No tienen un cálculo formal para transformar expresiones válidas en otras expresiones más sencillas o equivalentes.
- Lenguajes formales
  - Sintaxis sencilla
  - Limitan lo que se puede expresar
  - Explicitan las suposiciones
  - Relación formal entre lo escrito (sintaxis) y su significado (semántica)
  - Tienen cálculo para transformar expresiones válidas en otras válidas

Lenguajes formales. Ejemplos:

Aritmética: es un lenguaje formal para los números y sus operaciones. Tiene un cálculo asociado.

Lógicas: proposicional, de primer órden, modales, etc.

Lenguajes de especificación: sirven para describir formalmente un problema.

Contratos

- Una especificación es un contrato entre el programador de una función y el usuario de esa función.
- **Ejemplo**: calcular la raíz cuadrada de un número real.
- ¿Cómo es la especificación (informalmente, por ahora) de este problema?
- Para hacer el cálculo, el programa debe recibir un número no negativo.
  - Obligación del usuario: no puede proveer números negativos.
  - Derecho del programador: puede suponer que el argumento recibido no es negativo.
- El resultado va a ser la raíz cuadrada del número recibido.
  - Obligación del programador: debe calcular la raíz, siempre y cuando haya recibido un número no negativo
  - Derecho del usuario: puede suponer que el resultado va a ser correcto

Partes de una especificación (contrato)

1. Encabezado
   - Nombre del problema que se va a especificar.
   - Datos de entrada, parámetros o argumentos.
   - Salida (resultado esperado)
2. Precondiciones o cláusulas "requiere"
   - Condición sobre los argumentos, que el programador da por cierta.
   - Especifica lo que requiere la función para hacer su tarea.
   - Por ejemplo: "el valor de entrada es un real no negativo"
3. Postcondiciones o cláusulas "asegura"
   - Condiciones sobre el resultado, que deben ser cumplidas por el programador siempre y cuando el usuario haya cumplido las precondiciones.
   - Especifica lo que la función asegura que se va a cumplir después de llamarla (si se cumplía la precondición).
   - Por ejemplo: "la salida es la raíz cuadrada del valor de entrada"

Parámetros y tipos de datos

- La especificación de un problema incluye un conjunto de parámetros: datos de entrada cuyos valores serán conocidos recién al ejecutar el programa.
- Cada parámetro tiene un tipo de datos.
  - **Tipo de datos:** Conjunto de valores provisto de ciertas operaciones para trabajar con estos valores.
- Ejemplo 1: parámetros de tipo *fecha*
  - valores: ternas de números enteros
  - operaciones: comparación, obtener el año, ...
- Ejemplo 2: parámetros de tipo *dinero*
  - valores: números reales con dos decimales
  - operaciones: suma, resta, ...

¿Por qué escribir la especificación del problema?

- Nos ayuda a entender mejor el problema
- Nos ayuda a construir el programa
  - Derivación (Automática) de Programas
- Nos ayuda a prevenir errores en el programa
  - Testing
  - Verificación (Automática) de Programas

Algoritmos y programas

- El primer paso será especificar un problema
- Luego, el objetivo será escribir un algoritmo que cumpla esa especificación
  - Secuencia de pasos que pueden llevarse a cabo mecánicamente
- Puede haber varios algoritmos que cumplan una misma especificación

- Una vez que se tiene el algoritmo, se escribirá el programa
  - Expresión formal de un algoritmo
  - Lenguajes de programación
    - Sintaxis definida
    - Semántica definida
    - Qué hace una computadora cuando recibe ese programa
    - Qué especificaciones cumple
    - Ejemplos: Haskell, Python, C, C++, C#, Java, Smalltalk, Prolog.

Lenguajes de programación

- En palabras simples, es el conjunto de instrucciones a través del cual los humanos interactúan con las computadoras.

- Permiten escribir programas que son ejecutados por computadoras.

Lenguajes de programación

No es tema de la materia... pero demos algún contexto por si se ponen a googlear...

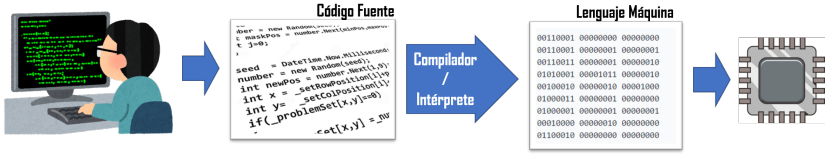
- **Lenguaje Máquina**: son lenguajes que están expresados en lenguajes directamente inteligibles por la máquina, siendo sus instrucciones cadenas de 0 y 1.
- **Lenguaje de Bajo Nivel**: son lenguajes que dependen de una máquina (procesador) en particular (el más famoso probablemente sea Assembler)
- **Lenguaje de Alto Nivel** (en la materia usaremos algunos de estos): fueron diseñados para que las personas puedan escribir y entender más fácilmente los programas que escriben.

```
Push   a        ; a → pila
Push   b        ; b → pila
Load   (c),R1   ; c → R1
Mult   (S),R1   ; b*c → R1
Store  R1,R2    ; R1 → R2
Add    (S),R1   ; a+b*c → R1
Store  R1,(x)   ; R1 → x
Add    #3,R2    ; 3+b*c → R2
Store  R2,(y)   ; R2 → y
```

Código fuente, compiladores, intérpretes...

No es tema de la materia... pero demos algún contexto por si se ponen a googlear...

- **Código Fuente**: es el programa escrito en un lenguaje de programación según sus reglas sintácticas y semánticas.
- **Compiladores e Intérpretes**: son programas *traductores* que toman un código fuente y generan otro programa en otro lenguaje, por lo general, lenguaje de máquina



IDE (Integrated Development Environment)

Para escribir y ejecutar un programa alcanza con tener:

- Un editor de texto para escribir programas (Ejemplos: notepad, notepad++, gedit, etc)
- Un compilador o intérprete (según el lenguaje a utilizar), para procesar y ejecutar el programa

Pero un mundo mejor es posible...

IDE (Integrated Development Environment)

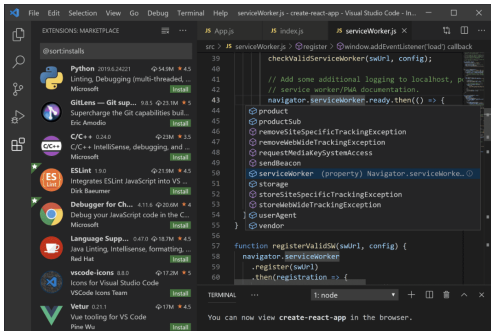
Ventajas de utilizar algún IDE

- Un editor está orientado a editar archivos mientras que un IDE está orientado a trabajar con proyectos, que tienen un conjunto de archivos.
- Integran un editor con otras herramientas útiles para los desarrolladores:
  - Permiten hacer destacado (*highlighting*) de determinadas palabras y símbolos dependiendo del lenguaje de programación.
  - Son capaces de verificar la sintaxis de los programas escritos (*linters*)
  - Generar vistas previas (*previews*) de cierto tipo de archivos (ej, archivos HTML para desarrollos web)
  - Suelen tener herramientas integradas (por ejemplo, el Android Studio tiene emuladores integrados)
  - Se pueden especializar según cada lenguaje particular
  - Permiten hacer depuración o *debugging*!

IDE (Integrated Development Environment)

Algunos IDEs:

- Visual Studio (https://visualstudio.microsoft.com/es/)
- Eclipse (https://www.eclipse.org/)
- IntelliJ IDEA (https://www.jetbrains.com/es-es/idea/)
- Visual Code o Visual Studio Code (https://code.visualstudio.com/)
  - Es un editor que se "convierte" en IDE mediante *extensions*.
  - Lo utilizaremos para programar en Haskell y Python.



Paradigmas

Existen diversos paradigmas de programación. Comunmente se los divide en dos grandes grupos:

- Programación Declarativa
  - Son lenguajes donde el programador le indicará a la máquina lo que quiere hacer y el resultado que desea, pero no necesariamente el cómo hacerlo.
- Programación Imperativa
  - Son lenguajes en los que el programador debe precisarle a la máquina de forma exacta el proceso que quiere realizar.

Paradigmas

Cada grupo, se especializa según diferentes características

- Programación Declarativa: describe un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución.
  - Paradigma Lógico: los programas están construídos únicamente por expresiones lógicas (es decir, son Verdaderas o Falsas).
  - Paradigma Funcional: está basado en el modelo matemático de composición funcional. El resultado de un cálculo es la entrada del siguente, y así sucesivamente hasta que una composición produce el valor deseado.

Paradigmas

Cada grupo, se especializa según diferentes características

- Programación Imperativa: describe la programación como una secuencia de instrucciones o comandos que cambian el estado de un programa.
  - Paradigma Estructurado: los programas se dividen en bloques (procedimientos y funciones), que pueden o no comunicarse entre sí. Existen estructuras de control, que dirigen el flujo de ejecución: IF, GO TO, Ciclos, etc.
  - Paradigma Orientado a Objetos: se basa en la idea de encapsular estado y comportamiento en objetos. Los objetos son entidades que se comunican entre sí por medio de mensajes.

Paradigmas

Cada grupo, se especializa según diferentes características

- Programación Declarativa
  - Paradigma Lógico: PROLOG
  - Paradigma Funcional: LISP, GOFER, HASKELL.
- Programación Imperativa
  - Paradigma Estructurado: PASCAL, C, FORTRAN, FOX, COBOL
  - Paradigma Orientado a Objetos: SMALLTALK
- Lenguajes multiparadigma: lenguajes que soportan más de un paradigma de programación.
  - JAVA, PYTHON, .NET, PHP

Paradigmas

Dado dos números, determinar si el segundo es el doble que el primero...

- Prolog:

```
% La siguiente regla es verdadera si X es el doble que Y
es_doble(X, Y) :-
    X is 2*Y.
```

- Haskell:

```
esDoble :: Integer -> Integer -> Bool
esDoble x y = x == 2*y -- Verificamos si x es igual al doble de y
```

- Python:

```
def esDoble(x: int, y: int) -> bool:
    if(x == 2*y):
        return True
    else:
        return False
```


Paradigmas

En la materia resolveremos (programaremos) problemas utilizando estos dos paradigmas:

- Paradigma Funcional
  - Utilizaremos Haskell
- Paradigma Imperativo
  - Utilizaremos Python

Resolviendo problemas con una computadora

Durante el cuatrimestre, además de resolver problemas, veremos algunos aspectos sobre cómo resolverlos:

- Hablaremos de buenas prácticas
  - Utilizar nombres declarativos
  - Modularizar problemas
  - Uso de comentarios
  - y más...
- ¿De qué se trata esto?... veamos un adelanto

Utilizar nombres declarativos

- Usar nombres que revelen la intención de los elementos nombrados. El nombre de una variable/función debería decir todo lo que hay que saber sobre la variable/función

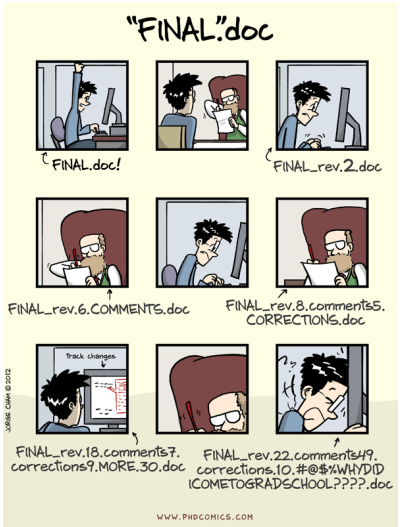
  1. Los nombres deben referirse a conceptos del dominio del problema.
  2. Una excepción suelen ser las variables con scopes pequeños. Es habitual usar **i**, **j** y **k** para las variables de control de los ciclos.
  3. Si es complicado decidirse por un nombre o un nombre no parece natural, quizás es porque esa variable o función no representa un concepto claro del problema a resolver.
  4. Usar nombres pronunciables! No es buena idea tener una variable llamada **cdcptdc** para representar la "cantidad de cuentas por tipo de cliente".

Utilizar nombres declarativos

Ambos programas son el mismo... ¿Cuál se lee más claro?

```
int x = 0;
vector<double> y;
...
for(int i=0;i<=4;i=i+1) {
 x = x + y[i];
}

int totalAdeudado = 0;
vector<double> deudas;
...
for(int i=0;i<=conceptos;i=i+1) {
 totalAdeudado = totalAdeudado + deudas[i];
}
```


Control de versiones



Sistemas de Control de Versiones (CVS)

- Permiten organizar el trabajo en equipo.
- Guarda un historial de versiones de los distintos archivos que se usaron.
- Facilitan la gestión del código fuente de un proyecto y la colaboración entre programadores.
- Existen distintas aplicaciones: svn, cvs, hg, git

Git

- Creado por Linus Torvalds, conocido por iniciar y mantener el desarrollo del kernel (núcleo) de Linux.
- Sistema de control de versiones distribuido, orientado a repositorios y con énfasis en la eficiencia.
  - Se tiene un servidor que permite el intercambio de los repositorios entre los usuarios.
  - Cada usuario tiene una copia local del repositorio completo.
- Acciones básicas: clone, checkout, add, remove, commit, push, pull, status...

Hagan el Taller de Git (y si pueden el de Latex)

¿Preguntas?

¡Nos vemos el jueves que viene!