CLASE_01_a >> Introducción y fundamentos principales

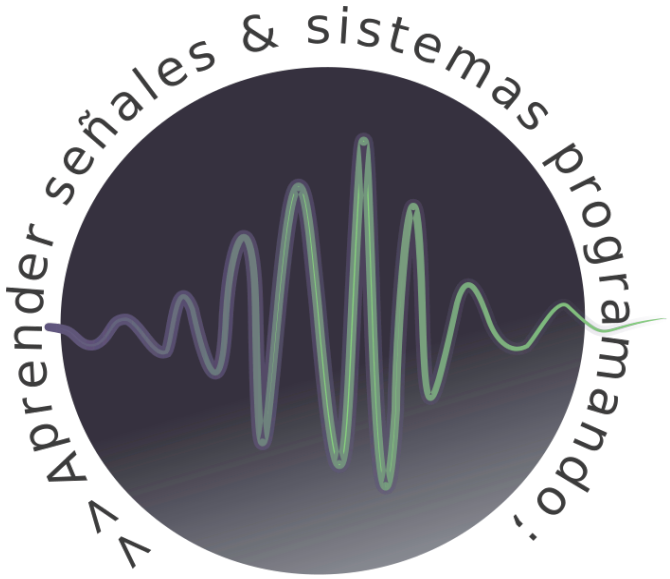


Tabla de contenidos

Conceptos básicos de programación

Algoritmos

Un algoritmo se define como una **descripción no ambigua y precisa de las acciones** que hay que realizar para resolver un problema bien definido en un tiempo finito. Sin embargo, es un método general para resolver todos los casos posibles del mismo problema y, por lo tanto, debe ser **independiente de los datos de entrada** de cualquier caso concreto. Un algoritmo se transforma en un **programa** a partir de que el mismo es codificado en un lenguaje que la computadora es capaz de interpretar [M. Jesús Marco Galindo, et al., 2010].

Paradigma de programación

Un paradigma de programación es una **propuesta tecnológica adoptada por una comunidad de programadores y desarrolladores**. Representa un enfoque particular o filosófico para diseñar soluciones. Los paradigmas difieren unos de otros, en los conceptos y la forma de abstraer los elementos involucrados en un problema, así como en los pasos que integran su solución del problema, en otras palabras, los cálculos. En general la mayoría son variantes de los dos tipos principales, imperativa y declarativa [M. Jesús Marco Galindo, et al., 2010]:

Básicamente, hay dos tipos de paradigmas de programación (en términos generales):

- **Programación imperativa:** los programas son **secuencias de instrucciones** que deben llevarse a cabo como una receta para resolver un problema determinador. Es la programación más utilizada y más antigua. Ejemplos puros de este paradigma serian: **C, BASIC, Pascal**.
- **Programación declarativa**: en contraposición a la programación imperativa, es un paradigma de programación que está basado en el desarrollo de programas **especificando o "declarando" un conjunto de condiciones,** proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica a la computadora qué es lo que se desea obtener o qué es lo que se está buscando. A su vez, este paradigma se dividen en **lenguajes funcionales** y **lenguajes lógicos**.

Hay muchos más tipos de paradigmas, pero la mayoría se desprenden de los mencionados anteriormente. Para conocer más del tema ir al siguiente link (Paradigma de programación - Wikipedia)

En esta asignatura vamos a trabajar principalmente con el paradigma de programación imperativa y en menor medida con la programación orientada a objetos, que se desprende de la anterior (Programación orientada a objetos - Wikipedia)

¿Cómo resolver un problema, en programación imperativa?

- **Definición del problema**: comprensión clara del problema. Entender profundamente cuál es el problema que se trata de resolver, incluyendo el contexto en el cual se usará. Una vez analizado el problema, asentar el análisis por escrito. En este sentido es importante en que enfoque se va a trabajar para lo cual en el diseño de software el **front-end** es la parte del software que interactúa con el o los usuarios y el **back-end** es la parte que procesa la entrada desde el front-end. La separación del sistema en front-ends y back-ends es un tipo de abstracción que ayuda a mantener las diferentes partes del sistema separadas. Por ejemplo, en un sonometro digital, front-end hace referencia a la visualización del entorno gráfico donde interactua el usuario y back-end al procesamiento de las señales. En sintetizadores del habla, el front-end se refiere a la parte del sistema que convierte la entrada del texto en una representación simbólico-fonética y el back-end convierte la representación fonética y simbólica en el sonido. Muchos programas tienen su concepto de diseño dividido en front-ends y back-ends, pero en la mayoría de los casos, el back-end está oculto del usuario final y solo pueden utilizarlo el cliente intermedio o el administrador que se encargará de gestionar el sistema de información. Sin embargo, muchos programas están escritos para servir de simple front-end para otros que ya existen.
- **Diseño del algoritmo:** a partir de la deficinicon del problema es preciso describir los valores de entrada (conocidos) y las salidas (incógnitas) que se requieran, incluyendo las unidades conforme describe los valores de entrada y salida. Trabajar con una **versión simplificada** del problema en un

lenguaje verbal de los pasos necesarios para resolver el problema. A continuacion se presentan dos herramientas para trabajar en este punto:

- **Diagrama de flujo**: enfoque **gráfico** para crear un agoritmo (Diagrama de flujo - Wikipedia).

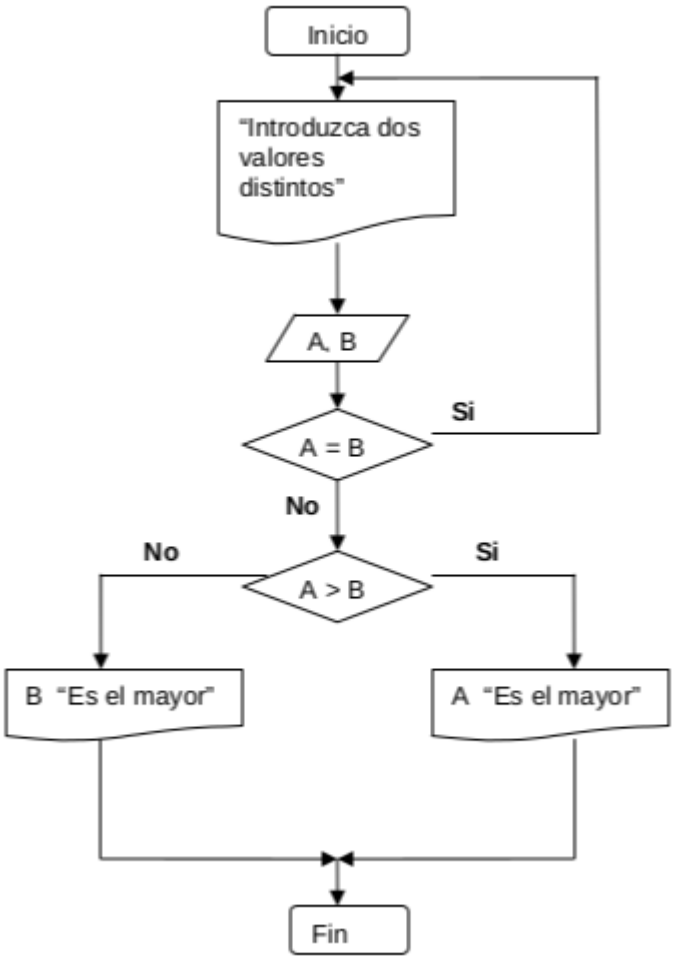
SImbología más importante:

Óvalo o Elipse: inicio y final del diagrama.

Rectangulo: representa la ejecución de uno o más procedimientos.

Rombo: representa la formulación de una pregunta o cuestión.

Trapezoide: cargar datos al sistema.



- **Pseudocódigo**: es una descripción **verbal** del algoritmo, compacta e informal (Pseudocódigo - Wikipedia).

1.Inicio

2.Inicializar variables: A = 0, B = 0

3.Solicitar la introducción de dos valores distintos

4.**Leer** los dos valores

5.Asignarlos a las variables A y B

6.**Si** A = B **Entonces** vuelve a 3 porque los valores deben ser distintos

7.Si A>B **Entonces**

Escribir A, "Es el mayor"

8.**De lo contrario**: **Escribir** B, "Es el mayor"

9.Fin_Si

10. Fin

Ventajas del pseudocódigo sobre los diagramas de flujo

Los pseudocódigos presentan los siguientes beneficios:

- Ocupan mucho menos espacio en el desarrollo del problema.
- Permite representar de forma fácil operaciones repetitivas complejas.
- Es más sencilla la tarea de pasar de pseudocódigo a un lenguaje de programación formal.
- Si se siguen las reglas de indentación se puede observar claramente los niveles en la estructura del programa.
- En los procesos de aprendizaje de los alumnos de programación, éstos están más cerca del paso siguiente (codificación en un lenguaje determinado, que los que se inician en esto con la modalidad Diagramas de Flujo).
- Mejora la claridad de la solución de un problema.
- **Implementación del programa:** el resultado del diseño es un algoritmo, pero para ejecutarlo se debe **traducir** el diseño realizado en un pseudo lenguaje o representación grafica **a un lenguaje de programación** que la computadora interprete. Esta etapa es sencilla, dado que sólo se trata de una traducción al lenguaje de interes de una manera bastante mecánica. Para lo cual es imprescindible conocer la **sintaxis** del lenguaje. Una buena opcion para realizar consultas respecto a la implementacion es Stackoverflow.

Stack orientado a Signal Processing

- **Pruebas (testing):** en la etapa de pruebas se trata de probar el programa resultante con diferentes datos de entrada.

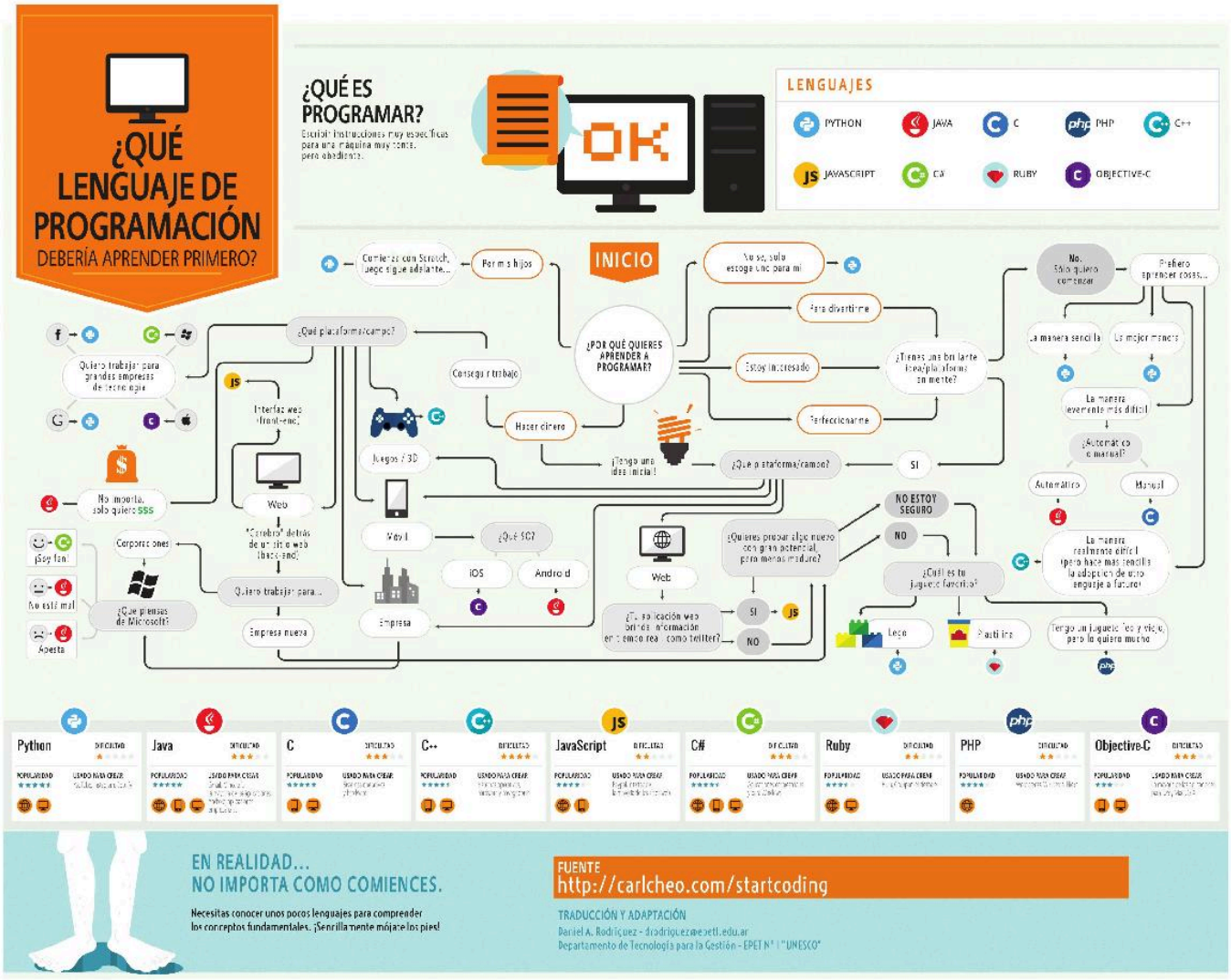
- **Mantenimiento y mejoras:** el mantenimiento del código y la implementación de mejoras no serian posible sin una correcta documentación. Herramientas excelentes para documentar y tener control sobre las versiones del código son: GitHub, Gitlab, Bitbucket, entre otras.

MATLAB

Para esta asignatura se va a trabajar con MATLAB

Caracteristicas principales del lenguajes

- Lenguaje M interpretado (Python, Octave, SciLab, otros).
- Manipulación de Matrices *(MA*Trix *LAB*oratory).
- Toolboxes.
- Graficas 2D y 3D.
- Interfaces de usuario.
- Combinación con otros lenguajes.
- Multiparadigma.
- Multiplataforma (Windows, GNU/Linux, MacOS)
- **Pago**.



Más información sobre el entorno de programación

Estrategia de aprendizaje para los alumnos de la asignatura

Sin conocimiento previo:

1. Seminario MATLAB UNTREF o curso "Introduction to MATLAB" MITOpenCourseWare [Yossi Farjoun, 2008]. Incluso MATLAB cuenta con una academia de aprendizaje virtual.
2. Clases practicas.
3. Guía de ejercicios.

Conocimientos básicos:

1. Clases practicas.
2. Guía de ejercicios.

En cualquiera de los casos consultar las referencias (respetando el orden) ante cualquier duda:

- Apunte de la asignatura "Introducción a MATLAB".

- Fundamentos de señales y sistemas usando la Web y Matlab [Kamen, Edward W., and Bonnie S. Heck, 2008].
- MATLAB para ingenieros [Holly Moore, 2007].
- Introduction to Audio Analysis: A MATLAB® Approach [Giannakopoulos, Theodoros, and Aggelos Pikrakis, 2014]

En mayo/Octubre, aproximadamente, se evaluará (con nota) las capacidades individuales en la plataforma.

Referencias

- Yossi Farjoun. *RES.18-002* ***Introduction to MATLAB****.* Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA. Spring 2008
- Kamen, Edward W., and Bonnie S. Heck. ***Fundamentos de señales y sistemas usando la Web y Matlab***. Pearson Prentice-Hall, 2008.
- Holly Moore. ***MATLAB para ingenieros***. Pearson, 2007.
- Giannakopoulos, Theodoros, and Aggelos Pikrakis. ***Introduction to Audio Analysis: A MATLAB® Approach***. Academic Press, 2014.
- Mat.caminos.upm.es. ***Lenguaje de programación - MateWiki***. Available at: https://mat.caminos.upm.es/wiki/Lenguaje_de_programaci%C3%B3n#Tipado_est.C3.A1tico_y_din.C3.A1mico. Universidad Politécnica de Madrid. 2017.
- M. Jesús Marco Galindo, Josep Vilaplana Pastó. **Introducción a la programación**. Universitat Oberta de Catalunya. pp 1-16, 2010.