Unidad 1: Conceptos Básicos de Programación

Introducción a la computación, algoritmos y pruebas de escritorio

Helder Octavio Fernández Guzmán

Agosto 2025

UMSS — introducción a la Programación

Motivación

- Entender **hardware** y **software** es el primer paso para **programar con sentido**: saber qué se ejecuta y cómo interactúa con el equipo.
- Los **algoritmos** son el corazón de la resolución de problemas: pensar paso a paso y verificar antes de codificar.
- Conectar la **historia de la computadora** con la programación ayuda a comprender por qué los lenguajes y paradigmas han evolucionado.
- Enfoque **ABP + VPL**: aprender resolviendo problemas reales y validando con prácticas en laboratorio virtual.

Objetivos del tema

- Distinguir entre **hardware** y **software**, identificando sus componentes y roles.
- Definir **algoritmo** y reconocer sus propiedades (claridad, finitud, orden).
- Aplicar **pruebas de escritorio** para validar algoritmos simples antes de programarlos.
- Relacionar la **evolución histórica** (video) con cambios en hardware, software y lenguajes.
- Preparar el terreno para prácticas en **VPL** usando problemas guiados por **ABP**.

Objetivos de aprendizaje

- Comprender qué es **hardware** y **software**.
- Reconocer la importancia de los **algoritmos**.
- Aplicar **pruebas de escritorio** en problemas simples.
- Relacionar la **evolución histórica** con la programación.

Video introductorio

Historia de la computadora (generaciones)

reproducción (10 minutos) y discusión breve.

Enlace: https://www.youtube.com/watch?v=a8Q2xpI7hbs

Preguntas guía

- ¿Qué cambios notaron entre generaciones?
- ¿Cómo se relacionan hardware y software en cada etapa?

Línea del tiempo del hardware (materiales)

Evolución de los materiales en las computadoras

- **1ª Generación (1940s–1950s):** Tubos de vacío — grandes, frágiles, alto consumo.
- **2ª Generación (1950s–1960s):** Transistores — pequeños, confiables, menor consumo.
- **3ª Generación (1960s–1970s):** Circuitos integrados — mayor velocidad y menor costo.
- **4ª Generación (1970s–1980s):** Microprocesadores — CPU completa en un chip, auge de la PC.
- **5ª Generación (1980s–hoy):** ULSI, microelectrónica y nanotecnología — miles de millones de transistores, IA y dispositivos móviles.

Línea del tiempo del software / SO

Evolución de los Sistemas Operativos

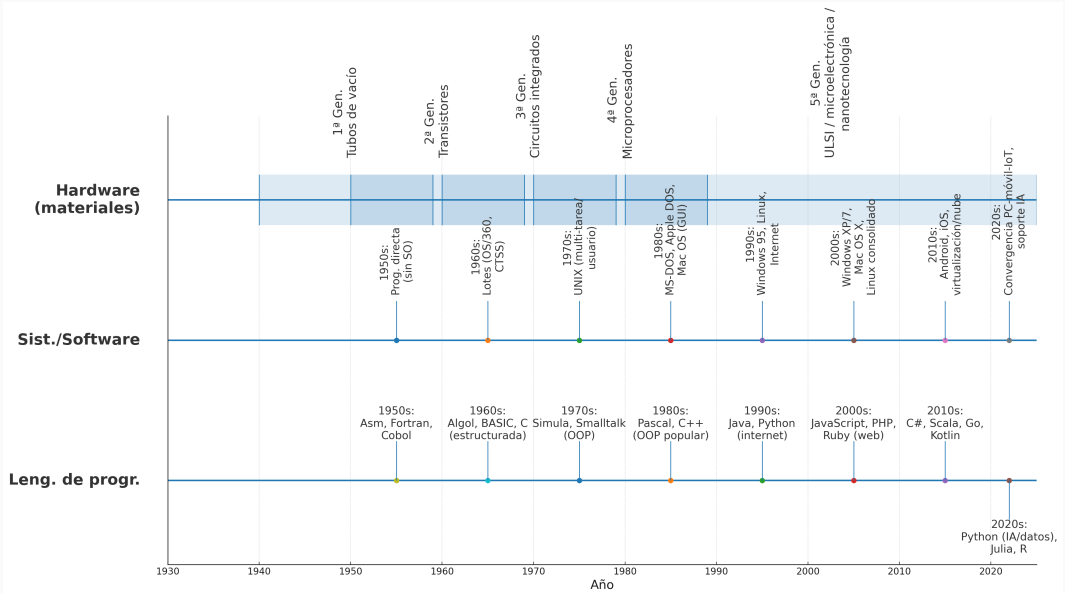
- **1950s:** Programación directa en máquina, sin SO.
- **1960s:** Primeros SO por lotes (IBM OS/360, CTSS).
- **1970s:** UNIX — multitarea, multiusuario, base de sistemas modernos.
- **1980s:** MS-DOS, Apple DOS, Mac OS con interfaz gráfica.
- **1990s:** Windows 95, Linux, auge de Internet.
- **2000s:** Windows XP/7, Mac OS X, Linux consolidado.
- **2010s:** Android, iOS, virtualización y nube.
- **2020s:** Convergencia PC–móvil–IoT, soporte para IA.

Línea del tiempo (en programación)

Evolución de lenguajes y paradigmas

- **1950s**: Ensamblador, Fortran, Cobol.
- **1960s**: Algol, BASIC, C (bases de la programación estructurada).
- **1970s**: Simula, Smalltalk (*orientación a objetos*).
- **1980s**: Pascal, C++ (OOP se populariza).
- **1990s**: Java, Python (internet y multipropósito).
- **2000s**: JavaScript, PHP, Ruby (web a gran escala).
- **2010s**: C#, Scala, Go, Kotlin (multiparadigma).
- **2020s**: Python (IA/datos), Julia, R (computación científica).

Panorama histórico integrado



Hardware: lo tangible

Componentes

- CPU (procesamiento)
- Memoria (principal/secundaria)
- Dispositivos de entrada (teclado, mouse)
- Dispositivos de salida (monitor, impresora)

Ideas clave

- El hardware ejecuta instrucciones del software.
- Rendimiento depende de arquitectura y recursos.

Software: lo intangible

Tipos

- Sistemas operativos (Windows, Linux, macOS)
- Aplicaciones (ofimática, navegadores, IDEs)
- Utilitarios (antivirus, compresores)

Relación con hardware

- Capa de abstracción para el usuario
- Gestión de recursos (CPU, memoria, E/S)

Algoritmos

Definición: Secuencia de pasos *claros* y *finitos* para resolver un problema.

Ejemplo (número par/impar):

1. Leer un número entero n.
2. Calcular n mód 2.
3. Si el resto es 0, mostrar “Es par”; en caso contrario, “Es impar”.

Buenas prácticas:

- Claridad, orden, finitud.
- Datos de prueba representativos.

Prueba de escritorio (desk checking)

Idea: Simular manualmente el algoritmo con datos de prueba para verificar su corrección.

Entrada	**Paso 1 (leer)**	**Paso 2 (n mód 2)**	**Salida**
$n = 4$	$n \leftarrow 4$	$4 \text{ mód } 2 = 0$	"Es par"
$n = 7$	$n \leftarrow 7$	$7 \text{ mód } 2 = 1$	"Es impar"

Errores típicos a detectar:

- Condiciones mal definidas.
- Casos borde no considerados (0, negativos, muy grandes).

Algoritmo cotidiano (narrativo)

Problema: Un extranjero necesita llegar en transporte público desde **Tiquipaya** hasta la **UMSS** (esquina **Oquendo** y **Jordán**).

Algoritmo cotidiano (narrativo)

Solucion propuesta:

1. Salir a la carretera principal en Tiquipaya.
2. Esperar un minibús con letrero "*Cercado / Centro*".
3. Preparar **cambio exacto** (aprox. Bs 3,50).
4. Subir al minibús y pagar al conductor.
5. Preguntar: *"¿Llega a la UMSS, Oquendo y Jordán?"*
6. Observar el recorrido y las calles señalizadas.
7. Identificar **Av. Oquendo** y el cruce con **Jordán**.
8. Tocar timbre / avisar para bajar en esa esquina.
9. Bajar con cuidado y verificar la señalización.
10. Caminar hasta la puerta principal de la UMSS.

Idea central: Un algoritmo efectivo es **claro**, **ordenado** y **finito**; omitir pasos puede "romper" el proceso.

Algoritmo cotidiano (pseudocódigo)

Pseudocódigo

```
Inicio
  Salir a carretera principal en Tiquipaya
  Esperar minibús con letrero ‘‘Cercado / Centro"
  Preparar Bs 3.50 en monedas
  Subir al minibús y pagar pasaje
  Preguntar si pasa por ‘‘Oquendo y Jordán"
  Mientras no se llegue a la esquina Oquendo-Jordán hacer
    Observar calles y letreros
  FinMientras
  Avisar para bajar en Oquendo y Jordán
  Bajar con cuidado
  Caminar hasta la puerta de la UMSS
Fin
```

Nota: Esta notación pseudocódigo es ilustrativa; los pasos son secuenciales y fáciles de verificar con una “prueba de escritorio”.

Discusión guiada (ABP)

Preguntas para el grupo

- Si **no** se prepara cambio exacto, ¿qué podría ocurrir? ¿Cómo lo prevenimos en el algoritmo?
- ¿Cómo ajustar el algoritmo si el minibús **no llega** a Oquendo y Jordán?
- ¿Qué **señales de validación** usarías para saber que estás cerca del destino?
- Propón una **versión alternativa** usando otra línea o combinando transporte.

Mini–reto: Intercambia tu algoritmo con un compañero y ejecuten una *"prueba de escritorio"* simulando el recorrido.
(Identifiquen pasos ambiguos o faltantes y mejoren el algoritmo.)

Resumen

- **Hardware** = parte física; **software** = parte lógica.
- **Algoritmo** = pasos ordenados y finitos.
- **Prueba de escritorio** valida el razonamiento antes de programar.
- La **historia** ofrece contexto para entender la evolución de ambos.

Tarea para la próxima clase

Tarea: Diseña un algoritmo cotidiano con condiciones y validaciones.

Elige **una** de las siguientes situaciones reales y descríbela en forma de algoritmo:

1. **Inscripción/matriculación a la carrera por primera vez** (revisar requisitos, llenar formularios, pago, verificación de cupos).
2. **Tramitar el carnet de identidad boliviano** (presentar documentos, toma de fotografía, pago en banco, retiro del carnet).
3. **Comparar y comprar un pasaje aéreo en línea (BOA vs otra línea)** (búsqueda de vuelos, comparación de precios/horarios, elegir opción, pagar).

Requisitos de la tarea:

- Algoritmo con al menos **10 pasos claros y ordenados**.
- Incluir **condiciones** y **casos alternativos** (ej: si falta un documento, si no hay cupo, si no hay disponibilidad de vuelo).