

INTRODUCCIÓN A LA PROGRAMACIÓN

Semana 1

ING. MARTIN POLIOTTO

Docente a cargo del modulo

Julio 2020



DIPLOMATURA EN

**NUEVAS
TECNOLOGÍAS**

SECRETARÍA DE
EXTENSIÓN
UNIVERSITARIA

SEU

UTN
Facultad Regional Córdoba

Ministerio de
PROMOCIÓN DEL EMPLEO
Y DE LA ECONOMÍA FAMILIAR

Ministerio de
CIENCIA Y
TECNOLOGÍA

CBA
ENTRE TODOS

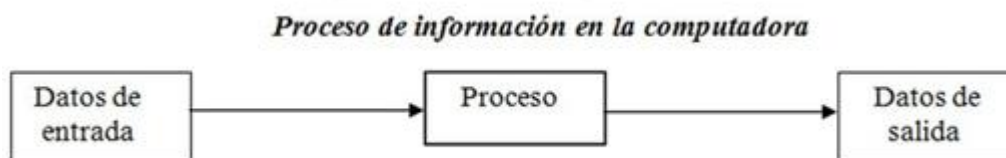
Oficina de
Montevideo
Organización
de las Naciones Unidas
para la Educación,
la Ciencia y la Cultura

Semana 01

1. Primeros conceptos

1.1 ¿Qué es una Computadora?

Una computadora es un dispositivo electrónico utilizado para procesar información y obtener resultados. Podemos pensar en un dispositivo que, a partir de ciertos datos de entrada (**input**), realiza una serie de **operaciones (Procesos)** y obtiene cierta información de interés para un usuario (**output**), tal como se muestra en el siguiente esquema:



Los componentes físicos que constituyen la computadora, junto con los dispositivos que realizan las tareas de entrada y salida, se conocen con el término de **hardware**. El conjunto de instrucciones que hacen funcionar a la computadora y se encuentra almacenado en su memoria se denomina **programa**. Al conjunto de todos los programas escritos para una computadora (por una persona a la que llamamos **programador**) se conoce con el término: **software**.



1.2 Breve reseña histórica

La computadora como concepto tecnológico ha sido desde el primer momento de su aparición un invento que no ha parado de mostrar innovaciones y mejoras, al punto que su evolución ha tomado por sorpresa a técnicos, científicos, académicos y empresarios de todas las épocas desde la década del 40 en el siglo XX, provocando muchas frases desafortunadas y errores notables en cuanto a las predicciones efectuadas respecto del futuro del invento.

Las primeras computadoras que pueden llamarse “operativas” aparecieron recién en la década del 40 del siglo XX, y antes de eso sólo existían en concepto, o en forma de máquinas de propósito único que pueden considerarse antecedentes válidos de las computadoras, pero no computadoras en sí mismas. Probablemente la primera persona que propuso el concepto de máquina de cálculo capaz de ser programada para procesar lotes de distintas combinaciones de instrucciones, fue el inglés *Charles Babbage* en 1821, luego ayudado por su asistente *Ada Byron*, también inglesa. Ambos dedicaron gran parte de sus vidas al diseño de esa máquina: Babbage diseñó la máquina en sí misma, sobre la base de tecnología de ruedas dentadas, y Byron aportó los elementos relativos a su programación, definiendo conceptos que más de un siglo después se convertirían en cotidianos para todos los programadores del mundo (como el uso de instrucciones repetitivas y subrutinas, que serán oportunamente presentadas en estas notas) La máquina de Babbage y Byron (que Babbage designó como “*Analytical Engine*” o “motor analítico”) era conceptualmente una computadora, pero como suele suceder en casos como estos, Babbage y Byron estaban demasiado adelantados a su época: nunca consiguieron fondos ni apoyo ni comprensión para fabricarla, y ambos murieron sin llegar a verla construida.

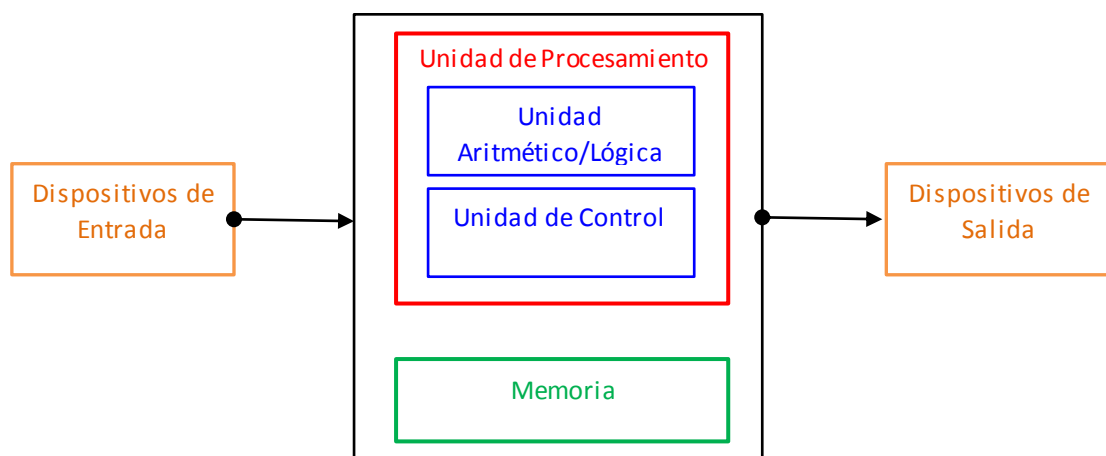
En 1890, en ocasión del censo de población de los Estados Unidos de Norteamérica, el ingeniero estadounidense *Herman Hollerith* diseñó una máquina para procesar los datos del censo en forma más veloz, la cual tomaba los datos desde una serie de tarjetas perforadas (que ya habían sido sugeridas por Babbage para su Analytical Engine). Puede decirse que la máquina de Hollerith fue el primer antecedente que *efectivamente llegó a construirse* de las computadoras modernas.

En 1943, con la Segunda Guerra Mundial en su punto álgido, el matemático inglés *Alan Turing* estuvo al frente del equipo de criptógrafos y matemáticos ingleses que enfrentaron el problema de romper el sistema de encriptación de mensajes generado por la máquina *Enigma* usada por los alemanes. Para esto, el equipo de Turing trabajó en secreto para desarrollar una máquina descryptadora que se llamó *Bombe*, gracias a la cual finalmente lograron romper el código *Enigma*. Sin embargo, cuando los alemanes cambiaron el código *Enigma* por el código *Lorenz SZ42*, Turing recurrió a *Tommy Flowers* para que diseñe y construya una nueva máquina, a la cual llamaron *Colossus*.

La primera máquina que podemos llamar computadora en el sentido moderno, apareció en 1944 de la mano del ingeniero estadounidense *Howard Aiken* y su equipo de colaboradores de la firma IBM. La máquina se llamó *Mark I* y fue construida para la Universidad de Harvard, aunque quedó obsoleta casi en paralelo con su desarrollo, pues usaba conmutadores mecánicos controlados en forma electrónica, mientras que al mismo tiempo otros investigadores ya estaban desarrollando la tecnología de tubos al vacío que derivaría en computadoras totalmente electrónicas. De hecho, en 1945 apareció la *Electronic Numerical Integrator and Calculator*, más conocida como *ENIAC*, que fue la primera computadora electrónica de gran escala.

La ENIAC fue diseñada por los ingenieros estadounidenses *John Mauchly* y *Presper Eckert* para la Universidad de Pensilvania.

En el mismo proyecto ENIAC trabajó *John von Neumann*, quien realizó aportes para el diseño de un nuevo tipo de máquina con programas almacenados designada como *EDVAC (Electronic Discrete Variable and Calculator)*. El diseño de *von Neumann* incluía una *unidad de procesamiento* dividida en una *unidad aritmético-lógica* y una *unidad de control*, además de contar con una *unidad de memoria* para almacenar tanto al programa como a sus datos, y también permitía *periféricos de entrada y salida* (ver **¡Error! No se encuentra el origen de la referencia.**). Este esquema con el tiempo se convirtió prácticamente en un estándar, al que se conoció como el *Modelo de von Neumann* o también *Arquitectura von Neumann* y muchas computadoras diseñadas desde entonces se basaron en el.



A finales de la década de 1950 y principios de la de 1960 los tubos al vacío fueron reemplazados por los transistores, lo que permitió abaratar costos y aumentar potencia. La aparición de los circuitos integrados hacia finales de la década del 60 permitió reunir muchos transistores en un solo circuito, abaratando aún más el proceso de

fabricación, y permitiendo reducir cada vez más el tamaño (y no sólo el costo) del computador. Esto abrió el camino que llevó a nuestros días: a finales de la década de 1970 aparecieron las primeras computadoras personales (*Apple II* e *IBM PC*), cuyo costo y tamaño las hacían accesibles para el público promedio... y el resultado está a la vista.

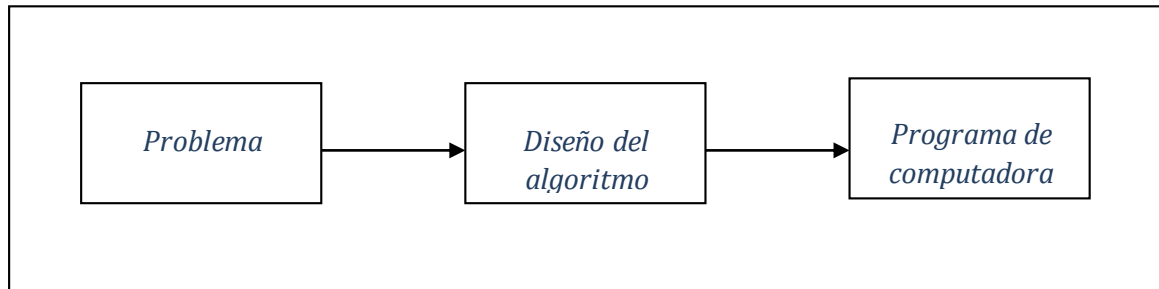
Sin embargo, el hecho que las computadoras hayan ocupado el sitio que ocupan en tan breve plazo, y que evolucionen en la forma vertiginosa que lo hacen, no es en realidad una sorpresa: simplemente se debe al hecho de su propia *aplicabilidad general*. Las computadoras aparecieron para facilitar tareas de cálculo en ámbitos académicos, científicos y militares, pero muy pronto se vio que podrían usarse también en centenares de otras actividades. Invertir en mejorar el invento valía la pena, y el invento mismo ayudaría a descubrir y definir nuevas áreas de aplicación.

1.3 Concepto de algoritmo

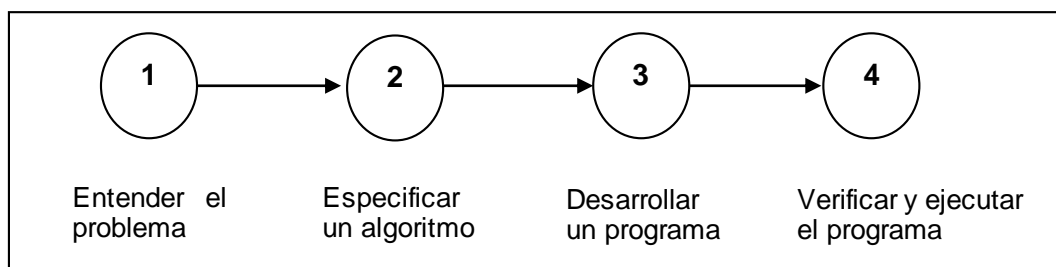
Todo programador de computadoras es antes que nada una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático. Un concepto que aparece de inmediato cuando nos referimos a una forma de resolver un problema es precisamente el de **algoritmo**.

En general, el conjunto finito de pasos para resolver un problema recibe el nombre de *algoritmo*.

Si un algoritmo se plantea y escribe de forma que pueda ser cargado en una computadora, entonces tenemos un *programa*.



En definitiva, para que un computador pueda ser usado para resolver un problema, los pasos a seguir se ven en el siguiente esquema:



Todo algoritmo debe cumplir las siguientes características fundamentales:

- Debe ser **preciso** e indicar el orden de realización de cada paso
- Debe estado **definido**. Si se sigue 2 veces el mismo algoritmo se deben obtener los mismos resultados (para el mismo conjunto de entradas)
- Debe ser **finito**. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, se debe tener un conjunto finito de pasos.

Podemos ejemplificar el concepto anterior mediante el siguiente ejemplo:

Un cliente ejecuta un pedido a una fábrica. La fábrica examina en su banco de datos la ficha del cliente; si el cliente es solvente entonces la empresa acepta el pedido: en caso contrario, rechazará el pedido.

El algoritmo correspondiente incluiría los siguientes pasos:

1. Inicio
2. Tomar pedido del cliente
3. Examinar la ficha del cliente
4. Si el cliente es solvente, entonces aceptar el pedido; caso contrario rechazarlo.
5. Fin.

Podemos tomar como segundo ejemplo un problema algo más complicado: *“conocer si un número es primo o no”*. Un número es primo si solo puede dividirse por sí mismo y por la unidad (dicho de otra manera, no tiene más divisores que él mismo y la unidad). Por ejemplo 9, 8, 6, 4, 12, 20, etc..., no son primos, ya que son divisibles por números distintos de ellos mismo y de la unidad. Así, 9 es divisible por 3, 8 lo es por 2, etc. El algoritmo de resolución del problema pasa por dividir sucesivamente el número por 2, 3, 4, etc.

1. Inicio
2. Poner **X** igual a 2 ($X = 2$, siendo X una variable que representa a los divisores del número que se busca **N**)
3. Dividir N por X (**N/X**)
4. Si el resultado de **N/X** es entero, entonces **N NO es un número primo** y bifurcar al punto 7; caso contrario continuar con el proceso
5. Sumar 1 a X (**X <- X + 1**)
6. Si **X** es igual a N, entonces **N**, entonces **N** es un número primo; en caso contrario, bifurcar al punto 3.
7. Fin.

Si por ejemplo **N = 131**, ejecutar el algoritmo anterior resulta:

1. Inicio
2. $X = 2$
3. Dividir 131/2. Como el resultado no es entero, se continúa el proceso.
4. $X = X + 1$, es decir $X = 3$
5. Como X no es 131, entonces se bifurca al paso 3.

Si continuamos con el proceso llegaremos a que 131 es un número primo.

1.4 Los lenguajes de programación

Para que un algoritmo sea efectivamente interpretado por un computador (es decir que el proceso descripto sea realizado por su procesador) este debe ser escrito en un **lenguaje de programación**. Un programa se escribe en un lenguaje de programación y las operaciones que conducen a expresar un algoritmo en forma de programa se llama *Programación*.

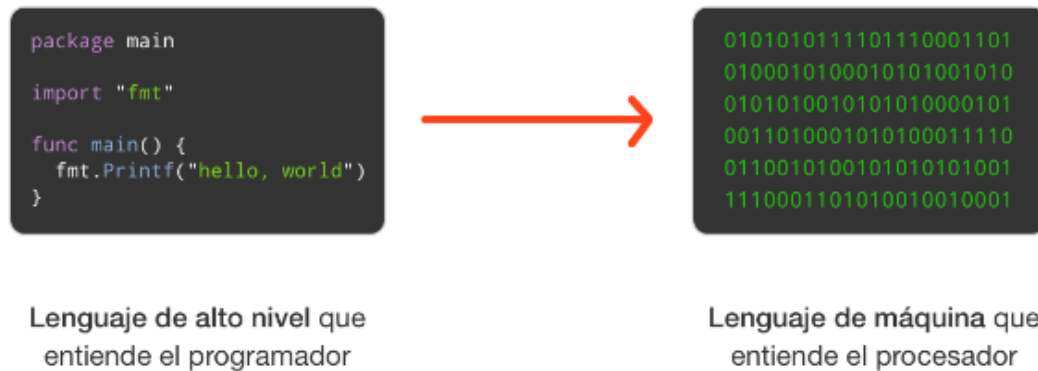
Los principales tipos de lenguajes son:

- Lenguaje de máquina: instrucciones binarias (cadenas de 0 y 1) que son interpretadas por la computadora (operaciones y direcciones de memoria escritas en binario)
- Lenguaje de Bajo nivel: son más sencillos de entender que el código de máquina pero, al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel más conocido es el *ensamblador* (*Assembly Language*). Este se reduce a una serie de instrucciones básicas que permiten manipular direcciones de memoria y ejecutar ciertas operaciones de cálculo a gran velocidad.
- Lenguaje de Alto nivel: son los utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas en un modo mucho más fácil que los lenguajes de máquina y ensambladores. Ejemplos de estos son: C/C++, Java o el mismo **Python**.

En general mientras más cerca del hardware estemos programando se dice que el lenguaje utilizado es de más bajo nivel. En cambio mientras más nos alejemos del hardware de la máquina y nos centramos en resolver problemas para un usuario independientemente del equipo, entonces utilizamos lenguajes de alto nivel.

1.5 Programas compilados vs interpretados

Nuestro código de programa escrito en un lenguaje de programación, comúnmente llamado **programa fuente** (o source) necesita ser traducido por programas traductores conocidos como compiladores o intérpretes. Dependiendo si necesitamos compilar o interpretar nuestro programa



frente es que hablamos de lenguajes de programación de un tipo u otro.

1.5.1 Lenguaje Interpretado

El uso de los lenguajes interpretados ha venido en crecimiento y cuyos máximos representantes son los lenguajes usados para el desarrollo web entre estos **Ruby**, **Python**, **PHP** (se interpreta del lado del servidor), **JavaScript** y otros como **Perl** o **Smalltalk**. Básicamente un lenguaje interpretado es aquel en el cual sus instrucciones o más bien el código fuente, escrito por el programador en un lenguaje de alto nivel, es traducido por el intérprete a un lenguaje entendible para la máquina paso a paso, instrucción por instrucción. El proceso se repite cada vez que se ejecuta el programa el código en cuestión.

Los lenguajes interpretados permiten el tipado dinámico de datos, es decir, no es necesario inicializar una variable con determinado tipo de dato sino que esta puede cambiar su tipo en condición al dato que almacene entre otras características más. También tienen por ventaja una gran independencia de la plataforma donde se ejecutan de ahí que los tres primeros mencionados arriba sean multiplataforma comparándolos con

algunos lenguajes compilados como Visual Basic, y los programas escritos en lenguajes interpretados son más livianos. La principal desventaja de estos lenguajes es el tiempo que necesitan para ser interpretados. Al tener que ser traducido a lenguaje máquina con cada ejecución, este proceso es más lento que en los lenguajes compilados, sin embargo, algunos lenguajes poseen una máquina virtual que hace una traducción a lenguaje intermedio con lo cual el traducirlo a lenguaje de bajo nivel toma menos tiempo.

1.5.2 Lenguaje Compilado

Un lenguaje compilado es aquel cuyo código fuente, escrito en un lenguaje de alto nivel, es traducido por un compilador a un archivo ejecutable entendible para la máquina en determinada plataforma. Con ese archivo se puede ejecutar el programa cuantas veces sea necesario sin tener que repetir el proceso por lo que el tiempo de espera entre ejecución y ejecución es ínfimo. Dentro de los lenguajes de programación que son compilados tenemos la familia **C** que incluye a **C++, Objective C, C#** y también otros como **Fortran, Pascal, Haskell o Visual Basic**.

En general, un lenguaje compilado está optimizado para el momento de la ejecución, aunque esto signifique una carga adicional para el programador. Por otro lado, un lenguaje interpretado está optimizado para hacerle la vida más fácil al programador, aunque eso signifique una carga adicional para la máquina.

1.6 La resolución de problemas con computadora

El proceso de resolución de un problema por computadora conduce a la escritura de un programa y su ejecución en la misma. Aunque el proceso de diseñar programas es, esencialmente un proceso creativo, se puede considerar una serie de fases o pasos comunes, que generalmente deben seguir todos los programadores.

Las fases de resolución de un problema con computadora son:

- Análisis del problema
- Diseño del algoritmo
- Codificación
- Compilación y ejecución
- Prueba
- Depuración
- Mantenimiento
- Documentación

Constituyen el ciclo de vida del software y su función es:







- **Análisis:** el problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por la persona que encarga el software.
- **Diseño:** una vez analizado el problema, se diseña una solución que conducirá a un algoritmo que resuelva el problema
- **Codificación:** la solución se escribe en la sintaxis del lenguaje de alto nivel y se obtiene un programa
- **Ejecución, Prueba y Depuración:** el programa se ejecuta, se comprueba rigurosamente y se eliminan los errores (o bugs) que puedan aparecer.
- **Mantenimiento:** el programa se actualiza y modifica, cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.
- **Documentación:** escritura de las diferentes fases del ciclo de vida del software, esencialmente el análisis, diseño y codificación, unidos a manuales de usuario y de referencia así como normas para el mantenimiento.

1.7 Herramientas de programación

Las dos herramientas más utilizadas comúnmente para diseñar algoritmos son: diagramas de flujo y pseudocódigos.

1.7.1 Diagramas de flujo

Un **diagrama de flujo** es una representación gráfica de un algoritmo. Los símbolos utilizados han sido estandarizados por el Instituto Norteamericano de Normalización (**ANSI**), y los más utilizados se muestran en la siguiente tabla:

| Símbolo | Significado |
|---|---|
|  | Representa el inicio y fin de un programa |
|  | Entrada/Salida de datos |
|  | Proceso: cualquier tipo de operación |
|  | Decisión: indica operaciones lógicas o de comparación de datos |
|  | Indicador de dirección o línea de flujo (sentido de ejecución de las operaciones). |
|  | Conectores: en la misma página o entre páginas diferentes |

1.7.2 Pseudocódigo

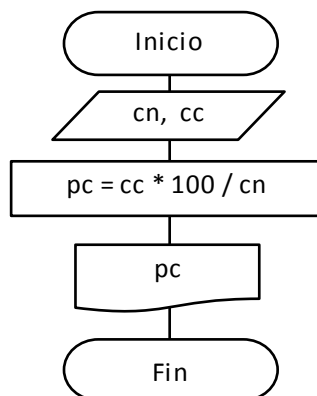
Es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan tanto la

escritura como la lectura de programas. En esencia, el pseudocódigo se puede definir como un *lenguaje de especificaciones de algoritmos*.

Podemos ejemplificar las herramientas de programación con el siguiente problema:

Problema: Se conoce la cantidad total de personas que padecen cierta enfermedad en todo el país, y también se sabe cuántas de esas personas viven en una ciudad determinada. Se desea saber qué porcentaje representan estas últimas sobre el total de enfermos del país.

Diagrama de flujo y pseudocódigo:



1. Cargar en *cn* la cantidad de enfermos del país
2. Cargar en *cc* la cantidad de enfermos de la ciudad
3. Calcular y asignar en *pc* el porcentaje:
 $pc = cc * 100 / cn$
4. Mostrar el porcentaje *pc*

Notar que el símbolo  representa una salida impresa.

1.7.3 PSeInt

PSeInt es una herramienta para asistir a un estudiante en sus primeros pasos en programación (Descargable desde

<http://pseint.sourceforge.net/>). Mediante un simple e intuitivo pseudo-lenguaje en español (complementado con un editor de diagramas de flujo), le permite centrar su atención en los conceptos fundamentales de la algoritmia computacional, minimizando las dificultades propias de un lenguaje y proporcionando un entorno de trabajo con numerosas ayudas y recursos didácticos. Algunas de las características principales son:

- Permite generar y editar el diagrama de flujo del algoritmo
- Puede interpretar (ejecutar) los algoritmos escritos
- Permite convertir el algoritmo de pseudocódigo a código numerosos lenguajes de programación, dentro de los cuales se encuentra Java.
- Es multiplataforma
- Es totalmente libre y gratuito

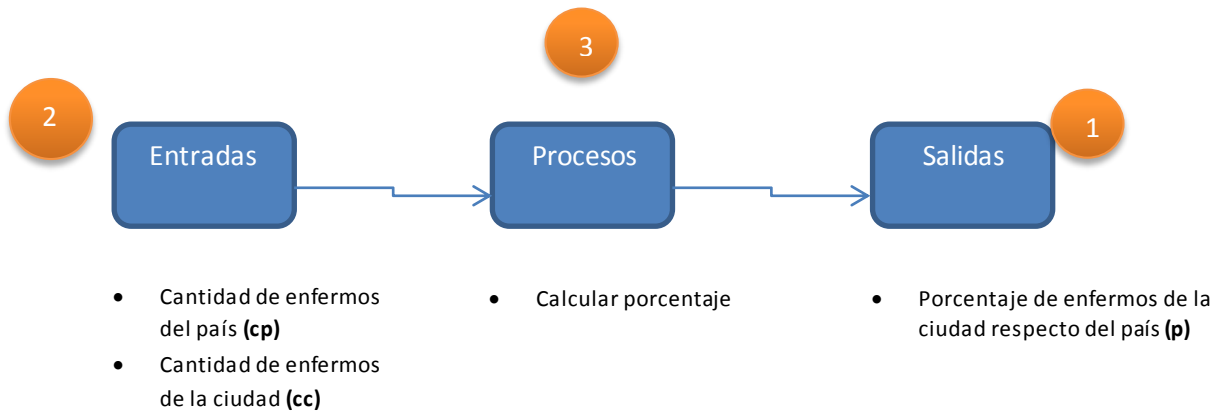
1.8 Método paso a paso

Pensado de una manera sistémica, podemos resolver problemas siguiendo los siguientes pasos:

- **1º Paso** Lectura inicial y análisis a priori de la situación planteada en el problema.
- **2º Paso** Análisis detallado del problema, se delimita el problema y divide en subproblemas más simples para su comprensión (Diseño Top Down o descendente). En este punto primero se analizan las salidas, luego las entradas necesarias para lograr los resultados esperados, y por último se describen los procesos requeridos (lógica de resolución) para transformar las entradas en salidas.

- **3° Paso** Graficar el Diagrama de Flujo o Algoritmo, que es la representación gráfica de la resolución del problema planteado, a través de la implementación del diseño Top Down (de lo general a lo particular), previamente analizado en la etapa anterior.
- **4° Paso** Comprobar que el Algoritmo funciona para una muestra pequeña y representativa de la población, lo cual implica seleccionar aquellas entradas de datos que pueden presentarse; mediante el uso de la herramienta Prueba de Escritorio (matriz $n \times n$ variables que facilita la ratificación o rectificación de la lógica implementada) y es una actividad áulica.
- **5° Paso** Codificar el Algoritmo a un lenguaje natural denominado Pseudocódigo, como una instancia previa y de consolidación de la lógica empleada, tendiente a traducir cada acción del Diagrama de Flujo planteado a instrucciones. (Actividad Áulica).
- **6° Paso** Codificar el Algoritmo en un lenguaje de programación y ejecutarlo, llevando el Algoritmo y Pseudocódigo realizado en los pasos anteriores; verificando que el mismo arroja los mismos resultados que la prueba de escritorio realizada en forma manual.

Si tomamos el problema anterior, de nuestro análisis resultaría:



Luego graficamos la solución en un diagrama de flujo (algoritmo), y finalmente podemos escribir y probar la solución en un lenguaje de programación.

2. Comenzando con Java

2.1 Breve historia

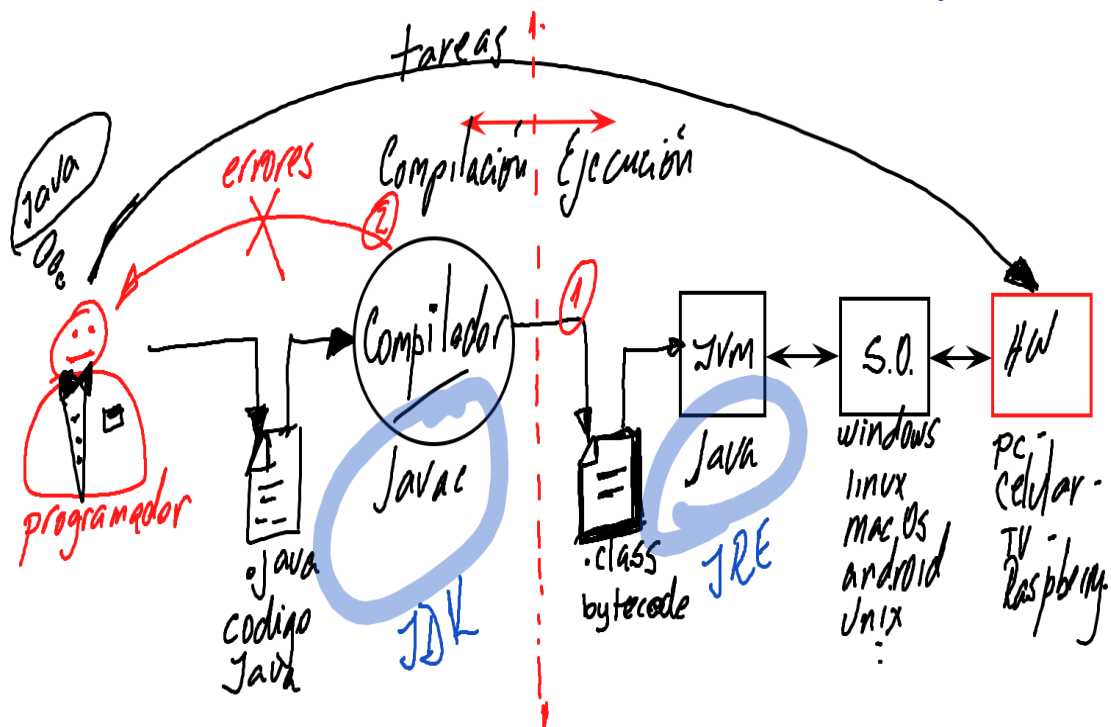
Hacia 1991, luego de la aparición del popular C++, un equipo de ingenieros de la firma Sun Microsystems, liderado por James Gosling y Patrick Naughton, comenzó a trabajar en el diseño de un lenguaje de programación que pudiera usarse para programar dispositivos electrodomésticos. Ese lenguaje debía tener la capacidad de adaptarse a distintos tipos de dispositivos y procesadores, siendo capaz de generar programas que pudieran correr en cualquier tipo de dispositivo que soportara al lenguaje. Se dice que algunos miembros del equipo de trabajo de Gosling y Naughton estaban tomando un café discutiendo sobre el nuevo nombre del lenguaje, y alguien notó que estaban tomando "café de Java". Sin más, se propuso el nombre Java para el nuevo lenguaje, y eso explica también por qué el logotipo del lenguaje es una taza de café humeante. Por los motivos ya indicados, el lenguaje Java se diseñó para poder generar programas que sean portables de una plataforma a otra. Vale decir: un programa desarrollado en un tipo de computadora con cierto tipo de sistema operativo, debería poder llevarse a otro tipo de computadora con un sistema operativo diferente, sin tener que cambiar el programa y sin tener que volver a compilar. La idea era buena y el lenguaje tenía elementos que lo hacían muy flexible y poderoso.

Por otro lado en 1991, la Internet fue liberada para su uso comercial y eso provocó un cambio profundo en las estrategias de comercialización, transmisión de datos, comunicaciones y (por supuesto) en el diseño y desarrollo de sistemas informáticos. La

Internet ofrece numerosos servicios, entre los cuales se encuentra la World Wide Web (o simplemente, la web) a través de la cual todas las computadoras enlazadas a la red pueden acceder a información gratuita sobre prácticamente cualquier tema que haya sido publicado. Con Java se podían crear pequeños programas llamados *applets*, los cuales podían incluirse dentro de una página web y descargarse en la computadora del usuario. Mediante applets, un programador web adquiriría mucha más flexibilidad en la programación y muchos más elementos gráficos que con HTML, y Java se relanzó como lenguaje adquiriendo de inmediato un éxito rotundo.

2.2 Entorno de trabajo

Para comenzar a escribir programas lo primero que se necesita es conocer cómo es el entorno de trabajo Java, es decir, qué herramientas son necesarias para nuestros desarrollos. La siguiente imagen pretende sintetizar este entorno.



Algunas consideraciones:

- El programador escribe, respetando la sintaxis del lenguaje, el código en un archivo con **extensión .java** (generalmente llamada código fuente)
- Luego, el código pasa por un proceso de compilación, donde principalmente se analiza que la sintaxis es correcta y de ser así se genera un **archivo .class** (conocido como **bytecode**)
- Si se presenta algún error durante el proceso anterior, necesitamos editar el archivo fuente y volver a compilar.
- Para este primer proceso es necesario tener instalado un **compilador**, que es una de las herramientas incluidas en el **JDK** (Java Development Kit). Este es distribuido gratuitamente por Oracle en:

<https://www.oracle.com/technetwork/es/java/javase/downloads/index.html>

Luego el archivo .class es interpretado por la pieza fundamental del entorno, la **Java Virtual Machine (JVM)**. Este programa es el

responsable de traducir el código escrito a código interpretado por equipo en particular (Hardware) con un Sistema Operativo (SO) determinado. Notar que no solo es posible ejecutar código Java en una pc de escritorio o notebook, sino que también puede ejecutar sobre un teléfono celular o Smart-tv, entre otros.

- Al entorno necesario para ejecutar programas Java, se lo conoce como **JRE (Java Runtime Enviroment)**

Para facilitar todas las tareas descriptas, los programadores cuentan con un programa especial que integra todo este trabajo: el **IDE (Entorno de Desarrollo Integrado)**. Uno de los más populares es *NetBeans* cuya distribución puede descargarse desde el sitio: <https://netbeans.org/>

2.3 Características principales

Java es un lenguaje:

- Orientado a Objetos: todo lo que hay son objetos.
- Compilado e Interpretado: (la JVM es más que un intérprete, pero mantiene el mismo concepto): De este modo se consigue la independencia de la máquina: el código compilado se ejecuta en máquinas virtuales que sí son dependientes de la plataforma. Para cada sistema operativo distintos, ya sea Microsoft Windows, Linux, OS X, existe una máquina virtual específica que permite que el mismo programa Java pueda funcionar sin necesidad de ser recompilado.
- Independiente de la Plataforma: Debido a que existen máquinas virtuales para diversas plataformas de

hardware, el mismo código Java puede funcionar prácticamente en cualquier dispositivo para el que exista una JVM.

Adicionalmente podemos mencionar:

- Gestiona la memoria de manera segura: solo se pide memoria para los objetos, luego un proceso **recolector de basura** que se encarga de liberarla cuando éstos ya no están siendo utilizados.
- Soporta multi-hilos
- Distribuido: es posible programar tanto en aplicaciones cliente como en procesos servidores.

Bibliografía

- [1] Luis Joyanes Aguilar, Programación en C++. Algoritmos, estructura de datos y objetos, Madrid: Mc Graw Hill, 2000.
- [2] Mgtr. Ing. Valerio Frittelli, Material de clases adaptado para Programa de Formación Plan 111 Mil en UTN-FRC, 2017
- [3] cesc1989, " Lenguajes de programación: Compilados vs Interpretados", 2012. [Online]. Available: <https://otroespacioblog.wordpress.com/2012/09/02/lenguajes-de-programacion-compilados-vs-interpretados/>