Conceptos básicos en programación y pensamiento computacional

Unidad 1

Apunte de cátedra
2do Cuatrimestre 2023

Pensamiento computacional (90)
Cátedra: Camejo

.UBAXXI

1. Introducción

Adelantemos el tiempo e imaginemos una situación probable en tu vida profesional. Vas a participar en un proyecto de sistemas. Ya sea que te toque un rol de especialista en el dominio como interlocutor entre los desarrolladores y el cliente, o que formes parte directa del equipo de desarrollo; es claro que tu tarea no será trivial. El trabajo de programación, de por sí, requiere un proceso importante de análisis para crear un modelo de solución válido. Y cuando la necesidad surge de parte de terceros, al proceso anterior se suman etapas de negociación donde se logra un consenso y definiciones importantes. En todos estos casos, compartir un lenguaje técnico facilita enormemente las cosas. Porque entender y compartir algunos términos y conceptos, con toda la carga de significado que conllevan, agiliza el diálogo y mejora la comprensión. Por eso, en todas las disciplinas se comparte un **Lenguaje Técnico** específico; y la Tecnología Informática, la Ingeniería de Software y la Programación de Sistemas en general, no son la excepción.

> Si un Arquitecto nuevo llega a la obra de construcción de una casa y le pregunta al Capataz o Maestro Mayor de Obras si los cimientos se hicieron con *zapatas* o *pilotes encadenados*, tanto uno como otro podrán continuar la conversación perfectamente, sin necesidad de explicarle, uno al otro, qué considera que es una zapata, pilote o cómo se definiría un sistema de encadenado… Comparten un lenguaje.

De modo que, antes de entrar en el desarrollo de los temas que nos ocupan en este curso haremos una breve introducción de ciertos conceptos, ideas y modelos que van a permitirnos establecer acuerdos y manejar un lenguaje común; incluso sobre términos y palabras que seguramente ya te son familiares, y aunque sea intuitivamente, comprendes bien. Sostendremos este ejercicio a lo largo de todo el curso porque las Ciencias de la Computación están en continuo cambio y renovación; y la Industria Informática colabora notablemente con el caos. Emplea caprichosamente nombres o términos, no siempre de acuerdo a su significado o peso específico, sino simplemente obedeciendo consignas de mercadotecnia (“marketing”). Es importante estar bien parados cuando el suelo se mueve. Entender qué significan los términos técnicos, por qué las ideas o modelos se llaman así y por qué se emplean es valioso para mantener el equilibrio.

Personalmente creo que gran parte de la visión críptica que tienen los legos (es decir, no especialistas) sobre el mundo de la programación y la computación en general, está sustentada en la falta de comprensión de los conceptos esenciales y el lenguaje técnico. Por eso, ***aprender a programar o pensar como un programador no sólo implica conocer ciertos modelos, herramientas y técnicas, también es importante aprender a hablar como ellos***.

1.1. La Computadora

Lo primero es lo primero… ¿Qué es una computadora?
Todos los programas que escribiremos serán ejecutados o corridos en una **computadora**. Así que deberíamos comenzar por ponernos de acuerdo en este concepto. ¿Qué es una computadora? Seguramente has usado o empleado varias a lo largo de tu vida y tienes una idea bastante formada. Pero necesitamos ser más que claros, debemos ser precisos:
Una computadora es un dispositivo físico de procesamiento de datos con propósito general.

Tal vez esta definición sea difícil de entender al principio así que vamos a ejemplificar.

Como hay computadoras por todas partes, es importante que aprendamos a separar la paja del trigo, sin marearnos en el intento.

> *El lavarropas de mi tía Jacinta, que lava fenomenal y se le puede pedir que haga un lavado con burbujas para las medias de fútbol de Juancito y un enjuague ultra light para el sweater de Lu; y que es capaz de controlar el agua que deja entrar de acuerdo al tipo de lavado y el peso del tambor, ¿es una computadora? El teléfono celular de mi tío Pipo, que es un modelo antiguo, de los de tapita; y le permite hacer llamadas, enviar SMSs y jugar a la viborita, ¿es una computadora? ¿Y el Mp4 que tiene guardado en un cajón mi hermana Ali?*

La respuesta a todas estas opciones es un **rotundo no**.
Es verdad que son dispositivos físicos de tratamiento de información (el lavarropas trata las manchas también), pero carecen de una propiedad esencial y que hace a la naturaleza de una computadora: **no son de propósito general**. No puedo conectar el lavarropas a internet y descargar un programa (**App**, de Application o programa o aplicación) que me deje calcular los intereses de una inversión financiera y otro que me permita llevar la fecha de vencimiento de mis pagos mensuales. A un teléfono viejo no le puedo descargar un juego diseñado por mí, para usarlo cuando estoy esperando en la cola.

La idea de una **computadora** es que es capaz de procesar datos y obtener nueva información o resultados. ¿Cuáles datos? ¿Qué resultados? Los que se le pidan. ¿Quién le pide eso a la Computadora? El o los programas que está ejecutando. ¿Qué programas se pueden correr en una computadora? Lo que la imaginación de la tecnología vigente permita. Para hacerlo más simple: Si tienes dudas acerca de si tu teléfono inteligente es una computadora; en realidad la pregunta que debes formularte es **si puedes instalar una nueva app** escrita por vos o descargada de internet, por ejemplo. Si la respuesta es sí, entonces ¡califica como tal!

1.2. Software y Hardware

Esto último nos conduce a un plano muy interesante, que habitualmente es mencionado, pero no se le otorga la importancia que tiene. **Las computadoras son sistemas que integran dos aspectos o planos contrapuestos e igualmente importantes y determinantes en su comportamiento**.

Toda computadora funciona con **Software** y **Hardware**. El **Software** es el conjunto de herramientas abstractas (programas). Lo llamamos la **componente lógica** de un modelo computacional. Por otro lado, el **Hardware** es el componente **físico** del dispositivo. Es decir que, el **Software** dirá qué hacer y el **Hardware** lo hará.

El mundo del **Software** (Ingeniería de Software y Ciencias Básicas de la Computación) es el mundo de las ideas, los modelos, el plano sin límites. De hecho, casi todas las ideas novedosas que se ven en programación ya fueron pensadas varias décadas atrás. Pero suele tomar muchos años que una idea ingeniosa llegue al plano real porque depende del grado de avance del **Hardware** y la tecnología con que es construido. Hoy en día la tecnología de **Hardware** dominante en computadoras es la electrónica, pero no incluiría jamás a ésta en la definición de una computadora. Las primeras computadoras no usaban esta tecnología.

Para que nos demos una idea cabal de la distancia entre estos dos escenarios (software y hardware), dos universos distintos, pensemos que la mayoría de las ideas y modelos empleados en la tan popular y mentada **Inteligencia Artificial** de hoy existen, y han sido publicadas, discutidas y compartidas en el mundo de la

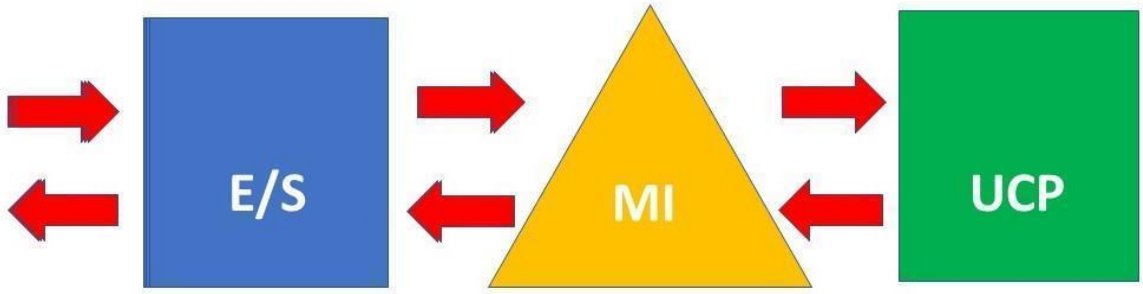
Ingeniería de Software desde mucho antes de la década del ‘80. Si no asomaron con fuerza a la superficie de la Industria Informática hasta hace unos años fue simplemente porque el estado del arte de la tecnología de hardware (y de algunas ramas de otras ciencias) no lo permitió. En la década del ‘80, la Internet era un juego de estudiantes en algún laboratorio perdido en California (no existía la world wide web), no se disponía (al menos con facilidad y a tan bajo costo) de los volúmenes de datos necesarios para emplear en los procesos de entrenamiento y se sabía bastante menos que ahora acerca de las neurociencias.

Pero sí, el concepto de cómo resolver problemas que requerían inteligencia y cómo lograr que los programas **aprendieran,** comenzaba a quedar bastante claro. De hecho, la famosa prueba de nivel de inteligencia de un programa que se llama **Test de Turing** fue diseñado por **Alan Turing** (sí, él, el genio de la película *Código Enigma*, matemático inglés que colaboró enormemente con las Ciencias de la Computación; sin dudas está en el Cuadro de Honor de los admirados por parte de la comunidad de programadores; y que vivió entre 1912 y 1954).

Los programadores elaboramos programas que serán ejecutados por una computadora, con su realidad y limitaciones de hardware. Usamos ideas ingeniosas, propias o ajenas, que sean viables, es decir, que nos permitan obtener resultados adecuados aplicándolas con el estado de avance tecnológico que disponemos. Es un trabajo creativo y pragmático y evoluciona constantemente de la mano de la evolución tecnológica en Hardware y Software. Además, es normal que ideas o modelos que habían sido abandonados o sustituídos por otros, con mejores resultados, refloten ante cambios de tecnología de hardware que los vuelven aptos e interesantes. Personalmente me gusta entender a la Ciencia Informática como una ciencia **vintage**.

1.3. El Modelo de Von Neumann y la Memoria Interna (MI)

Arquitectura de una Computadora: es el modelo que su diseño sigue con respecto a sus partes o **subsistemas** y la forma en que se interrelacionan. La Arquitectura prevalente hoy en día es **Von Neumann**. Tu computadora, celular, tablet, laptop, servidor de aplicaciones de tu compañía telefónica, y la inmensa mayoría, son **Von Neumann**.



Un modelo de arquitectura Von Neumann consta de tres componentes o subsistemas a través de los cuales fluye la información de una manera determinada. En una Arquitectura Von Neumann, se tienen tres partes:

1. La Información necesariamente ingresa a través del Subsistema de **Entrada - Salida** (**E/S).** La entrada sería nuestro mouse o teclado y la salida, nuestra pantalla.
2. Todo dato que se necesite para el procesamiento debe alojarse en **Memoria Interna** (**MI**), incluyendo los programas y los datos temporales que se generen.

3. La **Unidad Central de Proceso** (**UCP o CPU -Central Processing Unit-**) es quien realiza todas las transformaciones de datos, es decir, el proceso en sí. Los resultados se muestran a los usuarios a través del módulo **E/S**, pero éste los obtiene desde la **MI**, por ello, los resultados que genera **UCP** son guardados en **MI**.

Seguro que hay diferencias de componentes entre distintos equipos. Diferencias de procesador, diferencias en el ancho de buses, diferencias en la cantidad y calidad de componentes de memoria, etc. Pero todas están diseñadas, y funcionan de acuerdo al modelo de arquitectura que acabamos de describir.

Una consecuencia directa del empleo de esta arquitectura es que una instrucción debe estar cargada en **MI** para ser ejecutada. Y como no resulta cómodo escribir, uno por uno, los comandos o sentencias, antes de ser ejecutados, en la programación moderna se trabaja con el concepto de **Programa Almacenado**:

> **El Modelo de Funcionamiento de Programa Almacenado implica que para poder ser ejecutado un programa, debe ser cargado previamente en la memoria interna (MI).**

1.4. El Sistema Operativo

El **sistema operativo (SO)** es el programa encargado de administrar el equipo y sus recursos, que son constantemente disputados entre las solicitudes de todos los diferentes procesos o programas ejecutándose al mismo tiempo. Las computadoras modernas delegan esta gestión en el sistema operativo, siendo éste el encargado de administrar el equipo.

¿Una computadora puede funcionar sin el **SO**? ¡Claro! Pero ninguna computadora moderna lo intenta, no da buen resultado. Por ello, aunque no hace a la esencia del concepto de computadora, muchas personas toman como indicador de que un dispositivo está en esa categoría si trabaja con un **SO**. Algunos **SO** populares hoy en día son: Windows, Linux, IOS, Android.

1.5. Programa

Hemos mencionado varias veces la palabra **programa**. Una definición tradicional podría ser:

> **Un Programa de Computadora es un Algoritmo escrito en un Lenguaje de Programación.**

Pero resulta también interesante una definición matemática de este concepto. Podemos decir que la **Ecuación General de la Programación** es la siguiente:

$$P=A+D$$

Un **programa** (P) es la suma de **algoritmo** (A) y **datos** (D). Lo interesante de esta formulación es que podemos concluir rápida y genéricamente varias cosas, independientemente del programa del que se trate:

- Para escribir un **programa** es necesario escribir un **algoritmo** y diseñar una correcta organización de sus **datos**. Los dos términos de la ecuación son importantes, y como vemos, ninguno tiene una ponderación mayor en ella.

- Para mantener la misma calidad de P, si la calidad de D aumenta, A puede ser más simple. O, si mejoro A y D, forzosamente aumenta la calidad de P. Este concepto refleja mejor la idea de la programación moderna.

Por lo tanto, concluimos que: **Un buen programador no sólo mira el algoritmo, también debe ser muy inteligente en la selección del modelo y las herramientas para organizar los datos**. Por ello, en este curso comenzaremos aprendiendo a manejar las herramientas algorítmicas importantes, para pasar luego a hacer mayor foco en las herramientas de organización de datos.

Para explicar bien la ecuación deberíamos definir correctamente los términos.

1.5.1. Dato

El concepto de **dato** es tan trivial que resulta una de las cosas más difíciles del planeta definirlo. Partamos de esta definición:
"En el mundo de TI (tecnologías de la información), un dato es una representación simbólica ya sea numérica o alfabética, cuyo valor está listo para ser procesado por un ordenador y mostrarlo a un usuario en modo de información"

Tomada de *icorp.com.mx*

Seleccionamos esta definición porque incorpora el término **información**. En general, en software, diferenciamos **dato** de **información** considerando que **información es asignarle un significado al dato**. La diferencia no es trivial, pero es verdad que se desdibuja si usamos distintos niveles de abstracción. Por ello no somos muy ortodoxos en el empleo de estos términos, y solemos usarlos indistintamente.

1.5.2. Algoritmo

Un algoritmo es una serie finita de pasos precisos para alcanzar un objetivo.

Entendemos **serie**, como concepto matemático: **conjunto ordenado**. Un algoritmo debe constar de una cantidad **finita** de pasos, de lo contrario sería imposible alcanzar el objetivo. Recordemos que siempre el fin de un algoritmo es un objetivo bien específico.

En términos generales, se puede decir que se elabora un algoritmo para solucionar un problema. Los pasos deben ser absolutamente **precisos** y claros para quién debe llevar a cabo el algoritmo. Es decir, el grado de precisión que se requiere depende de para quién está dirigido.

Un ejemplo típico de algoritmo es una *receta de cocina*. Una serie finita de pasos precisos para obtener un plato determinado. Los pasos importan, el orden importa (si queremos hornear un bizcochuelo, primero se enmanteca y enharina un molde, se vierte la preparación y se envía al horno, cuando está cocido se desmolda. No da lo mismo que verter el contenido, enviar al horno, desmoldar y luego enharinar primero y enmantecar después). Como dijimos, el grado de precisión también cuenta, y depende de quién deba ejecutar el algoritmo.

Una receta de Lemon Pie:

1. *Preparar 200 gr de masa brisee*

2. *Fonzar un molde de 28 cm diámetro*
3. *Marcar 20 minutos en horno templado con peso*
4. *Retirar peso y terminar cocción*
5. *Rellenar con ½ kg de lemon curd*
6. *Cubrir con un merengue italiano de 4 claras*

¿Seremos capaces de ejecutar esta receta y llegar a buen puerto? Dependerá de nuestro conocimiento previo de cocina. O de la búsqueda exitosa de explicaciones en internet. También dependerá de quién va a cocinar este delicioso postre: ¿es un cocinero inexperto, o amateur? Dependiendo el caso, deberá emplear una receta (algoritmo) con un nivel de precisión diferente.

Para un programador, escribir un algoritmo es una tarea habitual, que se vuelve bastante intuitiva con la práctica. Sin embargo, para un principiante puede ser un desafío mayúsculo. A continuación detallaremos los pasos que debemos seguir para escribir un algoritmo. Este ejercicio se volverá cada vez más inconsciente, a medida que evoluciones en la destreza.

1. **Análisis del problema**
2. **Primer esbozo de solución**
3. **División del problema en partes**
4. **Ensamble de las partes**

- Una manera práctica de realizar un **análisis de problema** es plantearse muchos ejemplos. De situaciones típicas, de no tan típicas, de lo que está permitido, de lo que no, de lo que quizás; y definir concretamente cuál debería ser la respuesta correcta en cada caso.
- **Primer esbozo de la solución** es tirar la primera idea que se nos pueda ocurrir (en algún momento hay que zambullirse). No importa si es la mejor o la más brillante de las ideas, el proceso de diseño y construcción de programas normalmente es cíclico e incremental. Muchas veces las mejores ideas, como en los negocios, no son las primeras; pero la inacción y el status quo en busca de la perfección no es una conducta típica de un programador (ni de un ingeniero en general).
- Clarificada la primera idea, o compitiendo con ella, es interesante aplicar técnicas de diseño moderno. Nada funciona mejor para aclarar el caos que emplear la regla *"Divide et Vinces"*. Siempre es positivo descomponer un problema complejo en varios más sencillos y focalizar en cada uno de ellos.
- Una vez desarrolladas las partes, debemos acoplarlas para que todo el conjunto resuelva el problema planteado (siempre miramos al objetivo).

Ahora… ¿De qué manera debe ser escrito el **algoritmo** para que lo comprenda la **computadora**? Para responder esto es necesario precisar **qué entiende una computadora**. Hoy una computadora identifica y discierne sólo dos eventos: **Presencia de Tensión** O **Ausencia de Tensión**. Estos estados son: **disjuntos** y **complementarios**. Esta es una descripción absolutamente completa y concreta del mundo físico (hardware) actual de la computación. Hoy los fabricantes de procesadores ya los diseñan con un conjunto de instrucciones muy básicas que pueden entender y ejecutar.

Los dos únicos eventos que puede diferenciar una computadora (tensión y no tensión) son el motivo por el cual toda la realidad digital se modela empleando un sistema matemático binario (de base dos). La unidad mínima de medida de la información es un **bit**, corresponde a un dígito binario (0 o 1). Y claro, tantos 0s y 1s es mucho. Así que se agrupan (como decenas, centenas, etc., en el sistema decimal, pero en potencias de 2). **Byte** es 8 bits, 1 **Kbyte** es 1024 bytes, 1 **MegaByte** es 1024 Kbytes.

Esto es sólo una forma compleja de decir que **la computadora almacena y procesa información en forma de unos y ceros,** que manejan distintas unidades al agruparse cada cierta cantidad. Y cualquier algoritmo o instrucción que se le dé a la computadora debe estar codificado de manera tal que se pueda representar en términos de estos dos estados: uno y cero, o tensión y no tensión.

Si los programas son desarrollados por humanos (algunos son desarrollados por otros programas) y ejecutados por una computadora, tenemos que resolver la comunicación entre ellos (humanos y computadoras) para lograr que finalmente el algoritmo escrito por el programador sea entendido y aplicado por la máquina. No queda otra que utilizar un **lenguaje** que permita lograr una comunicación efectiva.

1.6. Lenguaje de Programación

¿Qué es un **lenguaje**? **Un lenguaje es un protocolo de comunicación.**

¿Y qué es un **protocolo**?: **Un protocolo es un conjunto de normas consensuadas.**

Así que, si logramos que un **Lenguaje** que pueda ser comprendido, tanto por el humano, como por la máquina, obtendremos lo que buscamos: una **comunicación efectiva**.

Al principio, el lenguaje elegido era el que la máquina conocía, en el mejor de los casos, todo se expresaba con una colección de órdenes (**primitivas**) conocidas por el procesador y que llamamos **Lenguaje Assembler**. Pero eso se hacía difícil. Había que intentar salvar la diferencia entre las dos puntas de la comunicación en un punto medio, idealmente más cerca del programador. Se hizo evidente la necesidad de emplear otra clase de lenguaje. ¿Cuál podría ser? ¿Cómo debería ser?

Un Lenguaje de Programación está dentro del grupo de los **Lenguajes Formales.** Estos son lenguajes que tienen un conjunto acotado y definido de elementos válidos (**tokens** o **palabras**) y reglas de construcción específicas de **sentencias** u **oraciones** válidas (**reglas de sintaxis**). Ejemplos de esta clase de lenguajes son los idiomas o Lenguas Vivas, el Lenguaje Matemático, el de Partituras Musicales, etc. Un lenguaje no formal sería aquel que tiene una fuerte polisemia, es decir que sus elementos pueden tener múltiples significados (como los cuadros, las películas, las composiciones musicales o los emojis; son todos de interpretación abierta).

Partiendo al medio la distancia entre los dos entes a comunicar (programador y computadora), se diseñan y emplean **lenguajes formales** con una **cantidad muy limitada de tokens válidos y reglas muy claras de generación** de los mismos y una **cantidad bien definida de reglas de sintaxis** para armar sentencias, acompañados de una serie concreta de reglas de interpretación de expresiones confusas (**reglas de desambiguación**). La evolución o mutación (cambios, agregados o eliminaciones de elementos y formas válidas) de estos lenguajes está controlada y supervisada por organizaciones con más éxito que la Real Academia Española. Si ese lenguaje está diseñado para escribir algoritmos que se limiten al procesamiento de datos, puede ser un **Lenguaje de Programación**.

Pero no sólo alcanza con tener un lenguaje, porque si escribo un programa con cualquier tipo de lenguaje formal la computadora podría de todas formas no entender nada, no reconocer ninguna orden o primitiva (lo mismo que si una persona me pide un vaso de agua en chino; es un lenguaje pero yo quizás no puedo traducirlo sin las herramientas adecuadas de traducción).

Será necesario traducirle a la computadora esas órdenes en código máquina (assembler). Un **Lenguaje** diseñado con todas las condiciones que hemos definido (formal, sin ambigüedades, acotado) facilita la traducción automática empleando **programas traductores**.

Estos programas tomarán sentencias y, sin tener en cuenta cualquier ambigüedad, ejecutarán lo pedido en las instrucciones. Pensemos que el mundo de la computadora es simple y sin grises: si o no. Es difícil charlar con una máquina y comentarle: - *El programa estalló* y esperar que no interprete que hubo destrucción masiva del equipo.

Un buen ejemplo de cómo una computadora interpreta nuestras instrucciones sin pensar al respecto, sin tener sentido común y sin ambigüedades, es este video: https://www.youtube.com/watch?v=cDA3_5982h8.
O el chiste de que la mamá de Juanito le dice: "andá al mercado. Trae un litro de leche, y si hay papas, traé 4", entonces Juanito vuelve con 4 litros de leche; porque había papas. La computadora es extremadamente literal al momento de leer nuestras instrucciones.

Usar un **Lenguaje de Programación**, en lugar de expresarnos con lenguaje coloquial (un lenguaje informal, familiar, con el que nos comunicaríamos con nuestros amigos y amigas, por ejemplo), hará nuestro trabajo más complicado, ya que tiene **reglas estrictas que respetar y no admite ninguna clase de errores o sobreentendidos**.

Aunque cada vez los lenguajes de programación se aproximan más a los lenguajes coloquiales, en el sentido de que podemos decir más con menos sentencias, los lenguajes de programación siguen siendo protocolos de comunicación y no lenguajes coloquiales. Y aunque el nivel de complejidad de las instrucciones que empleamos es cada vez mayor (complejidad porque resuelven cosas más complejas), sigue siendo mejor que volver a utilizar Assembler.

1.6.1. Traductores

Un lenguaje de programación con las características que definimos previamente (formal, sin ambigüedades, acotado), se identifica como un **Lenguaje de Alto Nivel.** Un lenguaje como assembler, más cercano al código máquina, se identifica como **Lenguaje de Bajo Nivel**.

La traducción de un programa escrito en un lenguaje de alto nivel a su versión en assembler, es realizada por otro lenguaje especializado. Existen dos tipos de programas traductores: **Intérpretes** o **Compiladores**.

Con la técnica de **compilación**, se lee y posteriormente traduce completamente un programa y, sólo entonces, puede ser ejecutado. Un traductor **intérprete** traduce sentencia a sentencia, a medida que se solicita su ejecución. En este curso emplearemos un lenguaje de programación (**Python**) que emplea técnica de traducción por intérprete. Los programas escritos en lenguaje de alto nivel se denominan **programas fuente** porque desde allí parten los traductores para generar una versión ejecutable.

Un **programa fuente** es un archivo de instrucciones. Esas instrucciones, aunque respetan la sintaxis del lenguaje de programación en que fue escrito, no dejan de ser texto sencillo, sin controles de formateo o alguna otra cosa especial. Por eso podemos editar un **programa fuente** con cualquier editor, por muy básico que sea. Una vez escrito y guardado, levantamos el programa fuente desde un programa intérprete para que gestione la traducción y ejecución del mismo.

Para mayor comodidad del programador, así como tenemos aplicaciones para editar documentos de texto que nos permiten hacer algunos trabajos con imágenes, tablas de datos, etc., podemos trabajar durante todo el **ciclo de vida de un programa** (escribimos, probamos, corregimos, volvemos a probar, y así hasta que la versión nos conforme y podamos liberarla) con herramientas integradas en una única app que cubre todas las necesidades de trabajo que puedan surgir. Abrimos esa herramienta y desde allí cubrimos todos los momentos y necesidades de la actividad de programación. Las aplicaciones que permiten hacer esto se denominan **Ambientes Centrados en Lenguajes** o **Entornos de Desarrollo**. Para desarrollar la práctica de este curso existen muchos ambientes disponibles, todos de acceso gratuito, y podrás elegir el que más se ajuste a tu gusto y comodidad.

Tal como hemos planteado, el abismo entre los programas diseñados por el programador y la realidad de quien tiene que ejecutarlos (la máquina), es cada vez mayor. El gap existe y es real, pero no se sortea por arte de magia. En el Mundo de la Informática Moderna la posibilidad de solución de problemas de todo tipo y complejidad se apoya sobre una gran cantidad de capas de software (**layers**) que se montan sobre la capacidad mínima de comprensión de una computadora y que van resolviendo problemas específicos y brindando servicio a las capas superiores. Administradores de dispositivos, transmisores de información, administradores de recursos propios, organizadores de tareas, sistemas operativos, graficadores, traductores, compresores, ambientes de diseño, etc., van resolviendo problemas puntuales y encargándose de determinadas tareas para que las aplicaciones de niveles más altos puedan abordar la resolución de problemas cada vez más complejos y abstractos, despreocupándose de detalles o problemas recurrentes.

2. Lenguaje Python

En este curso emplearemos como herramienta de programación el Lenguaje **Python**. Un lenguaje de alto nivel, interpretable, **multi paradigma**, **multi plataforma**. Es fácil explicar por qué es el elegido. Tanto como para el DT de la selección argentina argumentar por qué convoca a Messi. En la actualidad es uno de los lenguajes más estudiados y su uso para la construcción de aplicaciones variadas sigue creciendo.

Decir que es **multi paradigma** es equivalente a decir que se pueden construir programas con distintos enfoques o modelos de resolución de problemas usando el mismo lenguaje. Y claro, ya que estamos, también podemos mezclar **paradigmas** en un mismo programa. Por más maravilloso que sea, es un gran dolor de cabeza para los docentes o entrenadores que debemos cuidar mucho a nuestros aprendices para que no naufraguen. Algo así como comprar una manzana en la **Triple Frontera**. Puedes hacerlo en **Foz do Iguaçu**, **Puerto Iguazú** o **Ciudad del Este**. Incluso puedes comprar en cada una de esas ciudades. O comprar en una y comerla en otra. ¡¡Da igual! Pero no puedes pedirla de la misma forma, ni pagarla con cualquier moneda en todos los lados. Lo que importa es que sepas dónde estás parado, decidas qué te conviene, y lo hagas adecuadamente.

También es **multiplataforma**, es decir que un programa en Python puede ejecutarse en distintos **SO**. Esto es apenas un poco más frecuente que lo anterior, sobre todo si el lenguaje es interpretable (no compilable).

Es **abierto**, es decir que podemos ver el código de las herramientas que provee; como consecuencia directa, es gratis. Además de una Organización (Python Software Foundation) que monitorea su evolución y consistencia, existe una vasta comunidad de usuarios. Ellos colaboran en la resolución de problemas, en el apoyo al aprendizaje y la extensión y mejora del mismo, sobre todo con la provisión de **librerías** o **módulos**, que es la

manera más simple de extender un paquete de herramientas. En la actualidad (diciembre 2022) la cantidad de **bibliotecas** disponibles superan largamente las 7000.

Permite el desarrollo **rápido de aplicaciones** (**RAD**) y esto es invaluable para iniciarnos en programación. No necesitamos escribir cosas extras y de difícil comprensión para confeccionar un programa simple. Podemos ir directo al grano.

Tiene una excelente administración de memoria, lo que lo convierte en competidor seguro en aplicaciones que manipulan grandes volúmenes de datos (como en analítica de datos, inteligencia artificial, etc.).

Y lo mejor de todo: **Es un lenguaje diseñado por programadores que piensan en la salud de programadores**. Saben qué necesitamos, qué nos complica la vida innecesariamente. Intentan volver más fácil nuestro trabajo.

2.1. Hola, Mundo!

¿Cómo escribimos un programa en Python que muestre por pantalla la frase **Hola Mundo**? (*Tradicionalmente el primer programa que debe escribir todo programador).*

Pensemos, el programa debe mostrar algo, no necesita ninguna información de afuera, sabe exactamente lo que debe mostrar y es una información constante; por lo tanto, deberá usar la función **print**() para mostrar por pantalla.

Programa	**Salida en pantalla**
`print('Hola Mundo')`	`Hola Mundo`

Observemos que en el **print**() colocamos la frase dentro de los paréntesis. **print**() es una función y dentro de los paréntesis debemos colocar los **parámetros** de funcionamiento. En el caso de **print**(), qué queremos que muestre por pantalla.

La frase **Hola Mundo** la escribimos entre comillas simples. Siempre que queramos que se muestre texto constante (como en este caso, un cartel de salida) debe avisársele a Python que es un texto encerrándolo entre comillas simples o dobles (una u otra, no mezcladas).

Sin embargo no veremos esas comillas en la salida, sólo veremos el texto deseado.

Vamos a escribir un programa que muestre la **edad** de **Juan**, que es **30**:

```
print('La edad de Juan es 30')
```

La salida en pantalla será:

```
La edad de Juan es 30
```

Otra forma de hacerlo es así:

```
oracion = 'La edad de Juan es 30'
```

```
print(oracion)
```

La salida en pantalla será:

```
La edad de Juan es 30
```

¡Mismo resultado! Con esto ya podés comenzar con la guía de la Sesión 1.

A partir de la próxima clase comenzaremos a ensayar nuestros primeros programas un poco más complejos.