Curso de especialización en Inteligencia Artificial y Big Data

UD01. LENGUAJES Y PLATAFORMAS DE PROGRAMACIÓN EN ENTORNOS DE INTELIGENCIA ARTIFICIAL

Carlos Sáenz Adán

1. Introducción

¿Cuántas acciones de las que has realizado hoy, crees que están relacionadas con la programación? Hagamos un repaso de los primeros instantes del día: te ha despertado la alarma de tu teléfono móvil o radio-despertador, has preparado el desayuno utilizando el microondas, mientras desayunabas has visto u oído las últimas noticias a través de tu receptor de televisión digital terrestre, te has vestido y puede que hayas utilizado el ascensor para bajar al portal y salir a la calle, etc. Quizá no es necesario que continuemos más para darnos cuenta de que casi todo lo que nos rodea, en alguna medida, está relacionado con la programación, los programas y el tratamiento de algún tipo de información.

El volumen de datos que actualmente manejamos y sus innumerables posibilidades de tratamiento constituyen un vasto territorio en el que los programadores tienen mucho que decir.

En esta primera unidad realizaremos un recorrido por los conceptos fundamentales de la programación de aplicaciones. Iniciaremos nuestro camino conociendo con qué vamos a trabajar, qué técnicas podemos emplear y qué es lo que pretendemos conseguir. Continuaremos con el análisis de las diferentes formas de programación existentes, identificaremos qué fases conforman el desarrollo de una aplicación software y avanzaremos detallando las características relevantes de cada uno de los lenguajes de programación disponibles, para posteriormente, realizar una visión general del lenguaje de programación Java. Finalmente, tendremos la oportunidad de conocer con qué herramientas podríamos desarrollar nuestros programas, escogiendo entre una de ellas para ponernos manos a la obra utilizando el lenguaje Java.

2. Programas y programación

Generalmente, la primera razón que mueve a una persona hacia el aprendizaje de la programación es utilizar el ordenador como herramienta para resolver problemas concretos. Como en la vida real, la búsqueda y obtención de una solución a un problema determinado, utilizando medios informáticos, se lleva a cabo siguiendo unos pasos fundamentales. En la siguiente tabla podemos ver estas analogías.

En la vida real…	En Programación…
Observación de la situación o problema.	Análisis del problema: requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle.
Pensamos en una o varias posibles soluciones.	Diseño o desarrollo de algoritmos: se establece una solución al problema sin entrar en detalles tecnológicos. Se aplican diferentes técnicas y principios para establecer de forma detallada los pasos a seguir para resolver el problema.

Aplicamos la solución que estimamos más adecuada.	Resolución del algoritmo elegido en la computadora: consiste en convertir el algoritmo en programa, ejecutarlo y comprobar que soluciona verdaderamente el problema.

¿Qué virtudes debería tener nuestra solución?

- Corrección y eficacia: si resuelve el problema adecuadamente.
- Eficiencia: si lo hace en un tiempo mínimo y con un uso óptimo de los recursos del sistema.

Para conseguirlo, cuando afrontemos la construcción de la solución deberemos tener en cuenta los siguientes conceptos:

1. Abstracción: se trata de realizar un análisis del problema para descomponerlo en problemas más pequeños y de menor complejidad, describiendo cada uno de ellos de manera precisa. Divide y vencerás, esta suele ser considerada una filosofía general para resolver problemas y de aquí que su nombre no sólo forme parte del vocabulario informático, sino que también se utiliza en muchos otros ámbitos.
2. Encapsulación: consiste en ocultar la información que manejan de los diferentes elementos que forman el sistema. La forma de manejar es información no debe influir en el resto de los elementos del sistema.
3. Modularidad: un proyecto software será dividido en módulos independientes, dependiendo de su tamaño, donde cada uno de ellos tendrá su función correspondiente. Los demás módulos del sistema podrán utilizar su funcionalidad sin necesidad de conocer cómo funciona internamente.

3. Los algoritmos y los lenguajes de programación

Se podría definir un algoritmo como una secuencia ordenada de pasos, descrita sin ambigüedades, que conducen a la solución de un problema dado.

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos. Piensa en una receta de cocina, ésta puede ser expresada en castellano, inglés o francés, podría ser cocinada en fogón o vitrocerámica, por un cocinero o más, etc. Pero independientemente de todas estas circunstancias, el plato se preparará siguiendo los mismos pasos.

La diferencia fundamental entre algoritmo y programa es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado lenguaje de programación para que puedan ser ejecutados en el ordenador y así obtener la solución.

Los lenguajes de programación son sólo un medio para expresar el algoritmo, es decir, establece una serie de normas sintácticas y semánticas para expresarlo. El diseño de los algoritmos será una tarea que necesitará de la creatividad del desarrollador y de los conocimientos de las técnicas de programación. Estilos distintos, de distintos programadores a la hora de obtener la solución del problema, darán lugar a algoritmos diferentes, igualmente válidos.

En esencia, todo problema se puede describir por medio de un algoritmo y las características fundamentales que éstos deben cumplir son:

- Debe ser preciso e indicar el orden de realización paso a paso.
- Debe estar definido, si se ejecuta dos o más veces con los mismos datos de entrada, debe obtener el mismo resultado cada vez. Además, debe dar una respuesta a cualquier dato de entrada.
- Debe ser finito, debe tener un número finito de paso

4. Etapas en el desarrollo de un programa

4.1. Análisis

Esta es la primera fase del proyecto. Es la fase de mayor importancia en el desarrollo del proyecto y todo lo demás dependerá de lo bien detallada que esté. También es la más complicada, ya que no está automatizada y depende en gran medida del analista que la realice.

Se especifican y analizan los requisitos funcionales y no funcionales del sistema.

- Funcionales: Qué funciones tendrá que realizar la aplicación. Qué respuesta dará la aplicación ante todas las entradas. Cómo se comportará la aplicación en situaciones inesperadas.
- No funcionales: Restricciones sobre los requisitos funcionales. Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

Lo fundamental es la buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas.

La culminación de esta fase es el documento ERS (Especificación de Requisitos Software). Donde el documento más importante es la relación de los requisitos funcionales y no funcionales del sistema, junto con la priorización de los mismos; indicando cuales son obligatorios, cuales opcionales, etc. Como ejemplo de requisitos funcionales para una aplicación de tiendas de cosmética habría que considerar:

- Si desean que la lectura de los productos se realice mediante códigos de barras.
- Si van a detallar las facturas de compra y de qué manera la desean.
- Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno.
- Si van a operar con tarjetas de crédito.
- Si desean un control del stock en almacén.
- Etc.

4.2. Diseño

Una vez identificados los requisitos es necesario componer la forma en que se solucionará el problema. En esta fase traduciremos los requisitos funcionales y no funcionales en una representación del software a desarrollar. Podemos encontrar 4 tipos de diseño dentro de esta fase:

Diseño de datos

Se encarga de transformar la información relativa al dominio del problema (obtenida durante el análisis) en estructuras de datos que se utilizan en la implementación. Es una tarea de abstracción, donde se definirán aquellas estructuras necesarias para abordar el desarrollo del programa.

Para realizar este diseño se pueden utilizar diagramas como entidad/relación (relacionadas con bases de datos) o diagramas de clases UML (de propósito más genérico), entre otros.

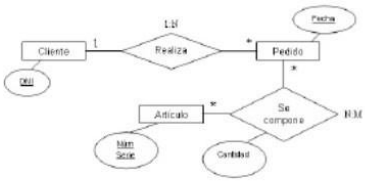


Diagrama entidad relación

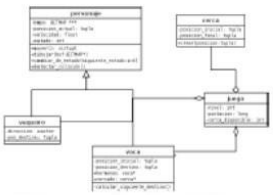
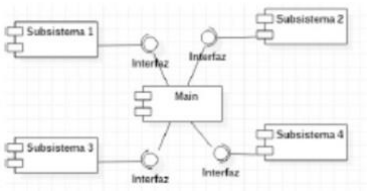


Diagrama de clases UML

Diseño arquitectónico

Se centra en la representación de la estructura de los componentes del software. Un componente puede ser: un módulo de la aplicación, la base de datos, los conectores entre capas, etc. Se utilizan diagramas que muestren la relación entre estos componentes, como por ejemplo los diagramas de componentes UML.



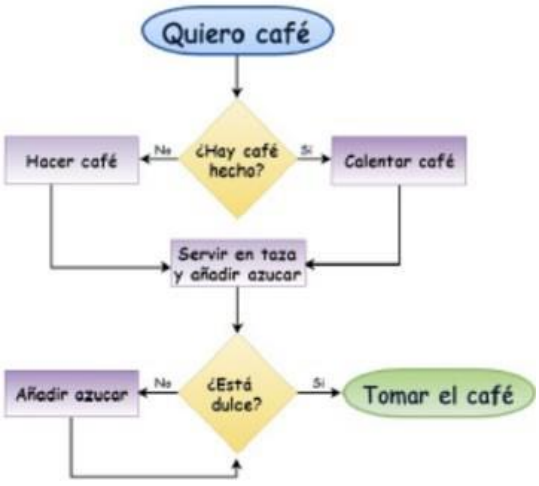
Diseño de interfaz

Describe cómo se comunica el software consigo mismo, con los sistemas que operancon él, y con las personas que lo utilizan. Para representar como pueden comunicarse las personas con el software se pueden utilizar prototipos. La siguiente imagen muestra un ejemplo de un prototipo realizado en papel, no es la única forma de realizarlos.

Diseño procedimental

Transforma los elementos estructurales anteriormente diseñados en una descripción procedimental. Es decir, describe a gran nivel el detalle de los algoritmos que implementarán los anteriores diseños. El resultado de este diseño es un nivel de diseño suficiente para que sirva de guía a la hora de generar el código fuente. Existen numerosas técnicas para realizar el diseño procedimental.

Diagramas de flujo: se utilizan una serie de símbolos interconectados que tienen un significado concreto.	Pseudocódigo: utiliza texto descriptivo para realizar el diseño del algoritmo.

	**algoritmo** Sumar
	
	variables
	  entero a, b, c
	
	inicio
	  escribir("Introduzca el primer número (entero): **")**
	  leer(a **)**
	  escribir("Introduzca el segundo número (entero): **")**
	  leer(b **)**
	  c ← a + b
	  escribir("La suma es: **",** c **)**
	fin

4.3. Implementación

En base a lo establecido en la fase de diseño, el programador codifica el programa utilizando un lenguaje de programación concreto. Para ello se basará en el diseño procedimental generado en la fase anterior. Como resultado se obtendrá el código fuente del programa.

Para la fase de implementación deberemos tener en cuenta los siguientes aspectos:

- Dependiendo del lenguaje de programación, podremos/tendremos que optar por un tipo de paradigma de programación (imperativo, lógico, funcional, ...)
- Se deben seguir unas guías de estilo concretas del lenguaje de programación o de la empresa (clean code).
- Depediendo del lenguaje de programación, podremos/tendremos que utilizar un entorno de desarrollo concreto.

Para saber más...

Código limpio (clean code). La definición de Clean Code puede tener múltiples acepciones. Aquí se incluye la traducción de una de ellas, formulada por Bjarne Stroustrup, el creador del lenguaje de programación C++:

Me gusta que mi código sea elegante y eficiente. La lógica del código debería ser sencilla para dificultar la depuración de errores, con un número mínimo de dependencias para facilitar el mantenimiento, con una gestión de errores completa de acuerdo a una estrategia bien definida, y con un rendimiento cercano al óptimo para evitar que otros desarrolladores generen código complejo como consecuencia del primero. El código limpio hace una cosa y la hace bien.

Piense en un proyecto en el que participan, concurrentemente, un equipo de 8 desarrolladores software. Resulta evidente asumir la existencia de una guía que sirva como referencia para escribir un código que sea consistente. De otro modo, cada ingeniero utilizaría, por ejemplo, una convención de nombrado diferente para su código. Esta situación dificultaría tanto la coordinación interna del equipo como el futuro mantenimiento del código.

Un ejemplo de guía de estilo para el desarrollo de código en C++ es la utilizada internamente por Google (ver mas información). Otro ejemplo representativo, para el caso del lenguaje de programación Python, es la guía de estilo PEP (Python Enhancement Proposal).

4.4. Pruebas

Tradicionalmente, esta fase está orientada a encontrar pruebas de calidad que se pueden realizar durante el proceso de desarrollo o con el producto entregado. Por regla general es altamente recomendable que las pruebas se vayan realizando durante el proceso de desarrollo.

En la práctica no se les da la importancia suficiente a las pruebas, lo habitual es disminuir plazo de pruebas para reducir plazos de entrega y costes. No obstante, esta es una muy mala práctica que a la larga supone todo lo contrario; incremento de costes y plazos.

Los principales motivos por los que suelen aparecer errores o fallos son:

- La falta de comunicación con el cliente, lo cual produce fallos de entendimiento entre lo que el cliente quiere y lo que el analista entiende.
- Gran complejidad del software desarrollado y poca reutilización.
- Se producen cambios de requisitos durante el proceso de desarrollo.
- Se reducen las pruebas con el objetivo de ganar tiempo. Esto produce que existan errores que no se detectan.
- No existe una documentación adecuada que permita el mantenimiento y reutilización de código.

De forma muy genérica, podemos decir que existen principalmente dos tipos de pruebas atendiendo a su objetivo y el momento en el que se realizan.

1. Verificación: consiste en comprobar los resultados de las distintas fases del desarrollo según unos parámetros de calidad que se definan. Responde a la pregunta de ¿El producto está correctamente construido?
2. Validación: consiste en comprobar los requisitos de software, lo que el usuario espera que hayamos hecho. Responde a la pregunta de ¿El producto construido es correcto? La validación comenzará una vez la verificación se ha completado.

4.5. Documentación

Todas las etapas del desarrollo deben quedar perfectamente documentadas. En esta etapa será necesario reunir todos los documentos generados y clasificarlos según el nivel técnico de sus descripciones. El conjunto de documentos relacionados con un proyecto software deben:

- actuar como un medio de comunicación entre los miembros del equipo.
- ser un repositorio de información del sistema para ser utilizado por el personal de mantenimiento.
- proporcionar información para ayudar a planificar la gestión del presupuesto y programar el proceso del desarrollo del software.
- indicar a los usuarios cómo utilizar y administrar el sistema.

Podemos diferenciar dos tipos principales de documentación, la relacionada con el proceso y la del producto.

- Documentación del proceso: registran el proceso de dearrollo y mantenimiento. Son documentos en los que se indican planes, estimaciones y horarios que se utilizan para predecir y controlar el proceso software. Contiene información sobre cómo usar los recursos durante el desarrollo, y

normas que se han de implementar en el proceso.

- Documentación del producto: registran información sobre el producto que está siendo desarrollado. Define dos tipos de documentación: la documentación del sistema que describe el producto desde un punto de vista técnico, orientado al desarrollo y mantenimiento del mismo; y la documentación del usuario que ofrece una descripción del producto orientada a los usuarios que utilizarán el sistema.

4.6. Implantación

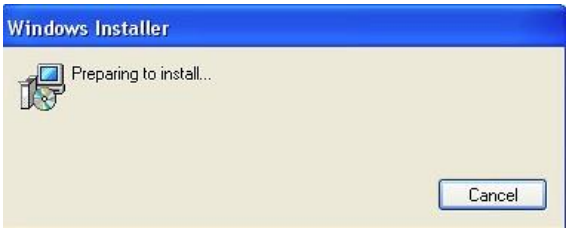
También llamado explotación. Una vez que se han realizado todas las pruebas y documentación de todas las etapas se pasa a la explotación del sistema. En esta etapa se lleva a cabo la instalación y puesta en marcha del producto software en el entorno de trabajo del cliente. En esta etapa se llevan a cabo las siguientes tareas:

- Se define la estrategia para la implementación del proceso. Se desarrolla un plan donde se establecen las normas para la realización de las actividades y tareas de este proceso. Se definen los procedimientos para recibir, registrar, solucionar, hacer un seguimiento de los problemas y para probar el producto software en el entorno de trabajo.

- Pruebas de operación. Para cada versión del producto, se llevarán a cabo pruebas de funcionamiento y tras satisfacerse los requisitos especificados, se libera el software para su uso operativo.

- Uso operacional del sistema. El sistema entrará en acción en el entorno privado de acuerdo con la documentación de usuario.

- Soporte al usuario. Se debe proporcionar asistencia y consultoría a los usuarios cuando la soliciten. Estas peticiones y las acciones subsecuentes se registran y supervisan.



4.7. Mantenimiento

Hay diferentes tipos de mantenimiento en función del motivo que lo origina:

- Mantenimiento correctivo: este se debe a que el cliente detecta errores en el software a pesar de las pruebas que se realizaron, y consiste en corregir dichos errores.
- Mantenimiento adaptativo: en muchos casos el software debe adaptarse a cambios del entorno externo porque, por ejemplo, se tiene un nuevo sistema operativo. Este tipo de mantenimiento se realiza frecuentemente por los rápidos avances en el ámbito de la informática.
- Mantenimiento perfectivo: es muy común que, con el paso del tiempo, el cliente solicite ampliaciones funcionales, es decir, desee que la aplicación incorpore nuevos requisitos funcionales. Asimismo, es posible que se deseen mejoras en requisitos no funcionales, por ejemplo, que el programa se ejecute más rápidamente o que se incremente la seguridad del sistema.

5. Los lenguajes de programación

Como hemos visto, en todo el proceso de resolución de un problema mediante la creación de software, después del análisis del problema y del diseño del algoritmo que pueda resolverlo, es necesario traducir éste a un lenguaje que exprese claramente cada uno de los pasos a seguir para su correcta ejecución. Este lenguaje recibe el nombre de lenguaje de programación.

Definimos lenguaje de programación como un conjunto de reglas sintácticas y semánticas, símbolos y palabras especiales establecidas para la construcción de programas. Es un lenguaje artificial, una construcción mental del ser humano para expresar programas. Además, cada lenguaje de programación, al igual que otro tipo de lenguajes, se basa en una gramática.

Gramática del lenguaje: Reglas aplicables al conjunto de símbolos y palabras especiales del lenguaje de programación para la construcción de sentencias correctas.

Además, cada gramática dispone de:

- Léxico: Es el conjunto finito de símbolos y palabras especiales, es el vocabulario del lenguaje.
- Sintaxis: Son las posibles combinaciones de los símbolos y palabras especiales. Está relacionada con la forma de los programas.
- Semántica: Es el significado de cada construcción del lenguaje, la acción que se llevará a cabo.

5.1. Clasificación de los lenguajes de programación

Los lenguajes de programación se pueden clasificar atendiendo a diferentes criterios como son el nivel de abstracción, el paradigma de programación, o la forma de ejecución. En concreto, se hará hincapié en la tercera forma, ya que resulta de relevancia en el campo de la IA.

Según el nivel de abstracción

Los lenguajes de programación han evolucionado a lo largo del tiempo para aumentar su rendimiento y

facilitar el trabajo a los programadores. Cada vez, existen lenguajes de programación que son más "legibles" y por lo tanto más fáciles de usar y entender. A continuación se muestran tres niveles de abstracción sobre los que se pueden clasificar los lenguajes de programación.

Bajo nivel de abstracción

- Sus instrucciones son complejas e ininteligibles. Se componen de combinaciones de unos y ceros.
- No necesita ser traducido, por lo tanto es el único lenguaje que entiende directamente el ordenador.
- Fue el primer lenguaje utilizado. En su momento, los expertos debían tener un dominio profundo del hardware para poder entender este lenguaje de programación.
- Difiere para cada procesador. Las instrucciones no son portadles de un equipo a otro.
- Salvo excepciones, no se suele utilizar este lenguaje para programar hoy en día.

Lenguaje de nivel medio

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

C000                    ORG     ROM+$0000 BEGIN MONITOR
C000 8E 00 70   START   LDS     #STACK

                        ****************************************
                        * FUNCTION: INITA - Initialize ACIA
                        * INPUT: none
                        * OUTPUT: none
                        * CALLS: none
                        * DESTROYS: acc A

0013                    RESETA  EQU     %00010011
0011                    CTLREG  EQU     %00010001

C003 86 13      INITA   LDA A   #RESETA   RESET ACIA
C005 B7 80 04           STA A   ACIA
C008 86 11              LDA A   #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A   ACIA

C00D 7E C0 F1           JMP     SIGNON    GO TO START OF MONITOR

                        ****************************************
                        * FUNCTION: INCH - Input character
                        * INPUT: none
                        * OUTPUT: char in acc A
                        * DESTROYS: acc A
                        * CALLS: none
                        * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04   INCH    LDA A   ACIA      GET STATUS
C013 47                 ASR A             SHIFT RDRF FLAG INTO CAR
C014 24 FA              BCC     INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A   ACIA+1    GET CHAR
C019 84 7F              AND A   #$7F      MASK PARITY
C01B 7E C0 79           JMP     OUTCH     ECHO & RTS

                        ****************************************
                        * FUNCTION: INHEX - INPUT HEX DIGIT
                        * INPUT: none
                        * OUTPUT: Digit in acc A
                        * CALLS: INCH
                        * DESTROYS: acc A
                        * Returns to monitor if not HEX input

C01E 8D F0      INHEX   BSR     INCH      GET A CHAR
C020 81 30              CMP A   #'0       ZERO
C022 2B 11              BMI     HEXERR    NOT HEX
C024 81 39              CMP A   #'9       NINE
C026 2F 0A              BLE     HEXRTS    GOOD HEX
C028 81 41              CMP A   #'A
C02A 2B 09              BMI     HEXERR    NOT HEX
C02C 81 46              CMP A   #'F
C02E 2E 05              BGT     HEXERR
C030 80 07              SUB A   #7        FIX A-F
C032 84 0F      HEXRTS  AND A   #$0F      CONVERT ASCII TO DIGIT
C034 39                 RTS

C035 7E C0 AF   HEXERR  JMP     CTRL      RETURN TO CONTROL LOOP
```

- Dada la dificultad y poca portabilidad del lenguaje máquina, el ensamblador lo sustituyó para facilitar la labor de programación.
- Sigue estando cercano al hardware, pero, en lugar de unos y ceros, se programó usando mnemotécnicos, que son instrucciones inteligibles por el programador que permiten comprender de una forma más sencilla qué hace el programa.
- Este lenguaje necesita compilarse y traducirse al lenguaje máquina para poder ejecutarse.
- Se trabaja con los registros del procesador y

  direcciones físicas. En lenguajes de nivel más alto, ya se utilizan variables y estructuras más sofisticadas.
- Es difícil de comprender y programar.
- Dada la dificultad y poca portabilidad del lenguaje máquina, el ensamblador lo sustituyó para facilitar la labor de programación.

Alto nivel de abstracción

- La mayoría de los lenguajes de programación actuales pertenecen a esta categoría (Java, Python, C#, C, C++, ...).
- Tienen una forma de programar más intuitiva y sencilla.
- Más cercano al lenguaje humano que al lenguaje máquina.
- Suelen tener librerías y funciones predeterminadas que solucionan algunos de los problemas que suelen presentarse al programador.
- En ocasiones, ofrecen frameworks para una programación más eficiente y rápida.
- Suelen trabajar con mucha abstracción y orientación a objetos. De esta manera, es más fácil la reutilización y la encapsulación de componentes.

```
import unidecode
import re

def eliminar_tildes_minuscula(texto):
    unidecode.unidecode(texto)
    return unidecode.unidecode(texto).lower()

def eliminar_hastags_puntuacion(texto):
    texto = texto.replace(", ", " ")
    return texto.replace("#", "")

texto = 'El #terremoto que sacudió el sábado al suroeste de #Haití dejó [...]'
texto = eliminar_tildes_minuscula(texto)
texto = eliminar_hastags_puntuacion(texto)

print(texto)
```

Según el paradigma de programación

El paradigma de programación define el estilo de programación que soporta un lenguaje de programación. En esta sección únicamente haremos referencia a los paradigmas de programación estructurada y orientada a objetos.

Programación estructurada

La programación estructurada se define como una técnica para escribir lenguajes de programación que permite sólo el uso de tres tipos de sentencias o estructuras de control:

- Sentencias secuenciales.
- Sentencias selectivas (condicionales).
- Sentencias repetitivas (iteraciones o bucles).

Los lenguajes de programación que se basan en la programación estructurada reciben el nombre de

lenguajes de programación estructurados.

El mayor inconveniente de este paradigma de programación es que el programa se concentra en un único bloque, el cual se hace complejo de mantener a medida que se añade funcionalidad. Además, no permite la reutilización de código, ya que va "todo en uno". Es por ello que la programación estructurada se sustituyó por la programación modular, donde los programas se codifican por módulos y bloques, permitiendo la reutilización de los mismos.

Programación orientada a objetos

Después de comprender que la programación estructurada no es útil cuando los programas se hacen muy largos, es necesaria otra técnica de programación que solucione este inconveniente. Nace así la Programación Orientada a Objetos (en adelante, POO). Los lenguajes de POO tratan a los programas no como un conjunto ordenado de instrucciones (tal como sucedía en la programación estructurada) sino como un conjunto de objetos que colaboran entre ellos para realizar acciones.

Su primera desventaja es clara: no es una programación tan intuitiva como la estructurada. A pesar de eso, alrededor del 55% del software que producen las empresas se hace usando este paradigma. Los principales lenguajes orientados a objetos: Ada, C++, C#, Delphi, Java, etc.

Principales características:

- Los objetos del programa tendrán una serie de atributos que los diferencian unos de otros.
- Se define clase como una colección de objetos con características similares.
- Mediante los llamados métodos, los objetos se comunican con otros produciéndose un cambio de estado de los mismos.
- Los objetos son, pues, como unidades individuales e indivisibles que forman la base de este tipo de programación.

Ventajas:

- El código es más reutilizable.
- Si hay algún error, es más fácil de localizar y depurar en un objeto que en un programa entero.

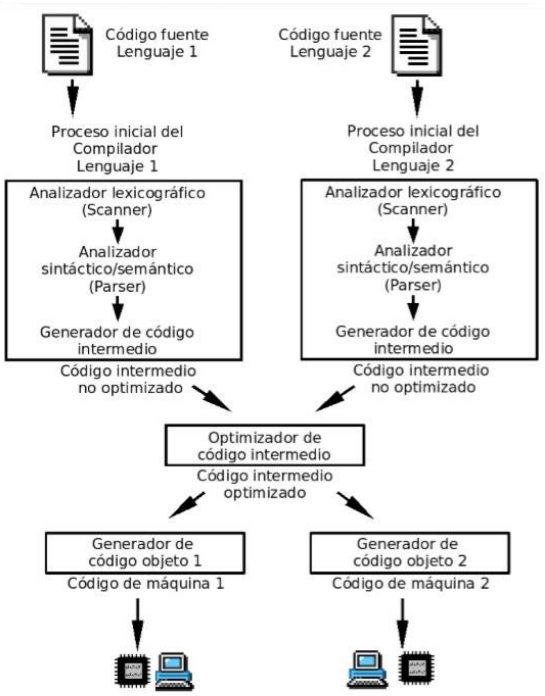
Según la forma de ejecución

Los lenguajes de programación se pueden clasificar atendiendo a la forma de ejecutarse. Diferenciando entre lenguajes compilados y lenguajes interpretados.

Lenguajes Compilados

En estos tipos de lenguajes existe un traductor de código fuente a código máquina denominado compilador. El compilador realizará la traducción y además informará de los posibles errores. Una vez subsanados, se generará el programa traducido a código máquina, conocido como código objeto. Este programa aún no podrá ser ejecutado hasta que no se le añadan los módulos de enlace o bibliotecas, durante el proceso de enlazado. Una vez finalizado el enlazado, se obtiene el código ejecutable.

Algunos ejemplos de este tipo de lenguajes son: Pascal, Fortran, Algol, C, C++, etc. En la siguiente imagen se puede ver cómo sería el proceso de compilado.



Lenguajes Interpretados

Se caracterizan por estar diseñados para que su ejecución se realice a través de un intérprete. Cada instrucción escrita en un lenguaje interpretado se analiza, traduce y ejecuta tras haber sido verificada. Una vez realizado el proceso por el intérprete, la instrucción se ejecuta, pero no se guarda en memoria.

Intérprete: Es un programa traductor de un lenguaje de alto nivel en el que el proceso de traducción y de ejecución se llevan a cabo simultáneamente, es decir, la instrucción se pasa a lenguaje máquina y se ejecuta directamente. No se genera programa objeto, ni programa ejecutable.

Ejemplos de lenguajes interpretados son: Perl, PHP, Python, JavaScript, etc. Lenguajes virtuales

A medio camino entre los lenguajes compilados y los interpretados, existen los lenguajes

que podemos denominar virtuales. Es el caso del Lenguaje Java. Java puede verse como compilado e interpretado a la vez, ya que su código fuente se compila para obtener el código binario en forma de bytecode, que son estructuras parecidas a las instrucciones máquina, con la importante propiedad de no ser dependientes de ningún tipo de máquina (se detallarán más adelante). Los ficheros que contienen los bytecodes de Java tienen extensión .class. La Máquina Virtual Java se encargará de interpretar este código y, para su ejecución, lo traducirá a código máquina del procesador en particular sobre el que se esté trabajando.

Comparativa general

En primer lugar, es importante destacar que, desde el punto de vista funcional, es posible implementar

mediante un lenguaje interpretado cualquier programa que se pueda implementar con un lenguaje compilado, y viceversa. Sin embargo, cada opción, como se ha introducido anteriormente, tiene sus ventajas y desventajas. Estas se resumen a continuación.

Con respecto a los lenguajes compilados:

- Generación de código eficiente, que se puede ejecutar un número arbitrario de veces, para la máquina destino. En otras palabras, la sobrecarga incurrida por el proceso de compilación, una vez que el código ha sido validado, se reduce a un único proceso.
- Posibilidad de aplicar optimizaciones durante el proceso de compilación.
- Incremento del rendimiento con respecto a un lenguaje interpretado.

Respecto a los lenguajes interpretados:

- Facilidad a la hora de implementar y prototipar código.
- Con carácter general, reducción de la complejidad a la hora de emplearlos como toma de contacto en relación a un lenguaje compilado.
- El código se puede ejecutar al vuelo, sin necesidad de un proceso previo de compilación.

Escenarios de aplicación

Teniendo como referencia estas cuestiones, se establecen dos escenarios generales que pueden servir como referencia a la hora de elegir un lenguaje de programación compilado o interpretado:

1. Si el contexto del proyecto lo permite, un lenguaje interpretado resultaría más adecuado para llevar a cabo tareas de prototipado rápido o para generar una primera versión de un proyecto con agilidad. Como se ha introducido en este capítulo, las aplicaciones de IA en las que se necesiten probar y adaptar comportamientos con cierta rapidez representan un candidato ideal con respecto a dicha elección.

2. Las tareas más intensas, desde el punto de vista computacional, o aquellas que se ejecuten con más frecuencia dentro de un proyecto software, se podrían relacionar con un lenguaje compilado. Por el contrario, aquellas menos intensas, como por ejemplo una interfaz de usuario, se podrían relacionar con un lenguaje interpretado.

5.2. Lenguajes de programación para IA

En este apartado se describen un conjunto mínimo de propiedades que, idealmente, todo lenguaje de programación utilizado para un proyecto de IA debería poseer. No se trata de ofrecer al lector un listado exhaustivo de características, sino una referencia inicial que sirva como punto de partida a la hora de elegir el lenguaje a utilizar para resolver un determinado problema. Para ello, se han escogido las siguientes 5 características:

1. Simplicidad, debido a la importancia de la misma para escribir y mantener el código.

2. Capacidad de prototipado rápido, debido a la propia naturaleza de las aplicaciones de IA, ya que su

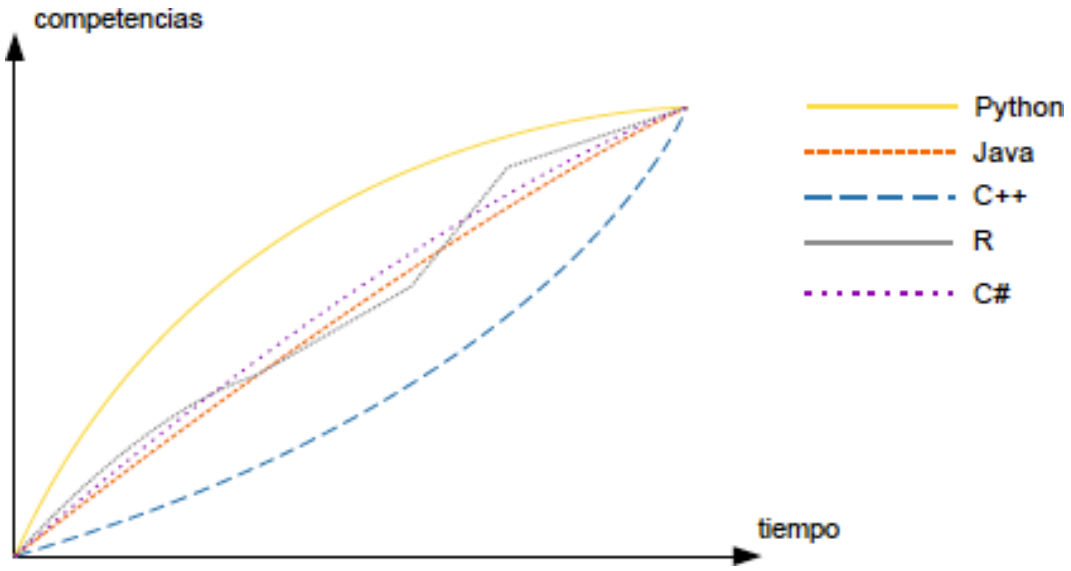
implementación suele variar considerablemente en las primeras etapas.

3. Legibilidad, debido a la necesidad de acercar lo máximo posible al programador y al código, considerando que un número significativo de desarrollos en este ámbito parten de pseudocódigo o algoritmos previamente diseñados.

4. Existencia de bibliotecas para IA, debido a la importancia de reutilizar código existente que acelere el desarrollo y asegure la fiabilidad de la aplicación. No hay que empezar de 0 todos los desarrollos.

5. Comunidad de desarrollo, debido a las ventajas que ofrece compartir experiencias y soluciones a problemas previamente abordados por otra persona.

Respecto a la curva de aprendizaje, un lenguaje de programación para IA debería facilitar un nivel básico de productividad incluso para principiantes. Esta capacidad se justifica con la necesidad de probar y validar conceptos de manera ágil, incluso por desarrolladores que no tenga una gran experiencia en el ámbito de la programación. En el dominio de la IA, al menos en las etapas iniciales de desarrollo, el foco debe estar en la técnica o enfoque de IA aplicado, y no tanto en la eficiencia o calidad del código fuente en su versión inicial. No obstante, es deseable que el programador tenga interiorizadas las competencias necesarias para resolver problemas, de manera independiente al lenguaje de programación, y pensar de forma abstracta.



5.3. Relación de los lenguajes de programación y la IA

Particularmente, los siguientes lenguajes han cobrado especial relevancia, dentro del contexto de desarrollo de IA, en los últimos años:

- **Python**. Probablemente, es el lenguaje de referencia para IA, y que será abordado, con más detalle y en el contexto de un ejemplo práctico, en la presente sección.

- **LISP** (List Processing). Definido, comúnmente, como el lenguaje para IA más antiguo. Actualmente, se utiliza principalmente para problemas de lógica y de aprendizaje automático.

- **PROLOG** (Logic Programming). Similar a LISP en cuanto a trascendencia y evolución históricas, se utiliza más frecuentemente para procesamiento de lenguaje natural.

- **R.** Lenguaje vinculado al desarrollo de IA debido a su facilidad para tratar y visualizar datos, combinando con el auge e importancia actual de data science.

- **Haskell**. Lenguaje funcional y de tipado estático, que se caracteriza por la importancia en la detección de errores en tiempo de compilación. Esta decisión de diseño es uno de los rasgos más relevantes del lenguaje desde su concepción.

- **Java**. Lenguaje de referencia para multitud de dominios y aplicaciones. Debido a su importancia histórica y popularidad, siempre será considerado una opción a la hora de desarrollar un proyecto de IA.

- **C++.** Se trata, probablemente, de uno de los lenguajes de programación más completo y versátiles en la actualidad. Los motivos de usar C++ como lenguaje para IA son similares a los expuestos para el caso de Java, aunque se debe considerar el incremento notable de rendimiento y las capacidades de desarrollo a medio nivel.

5.4. Python como lenguaje de referencia en IA

Los lenguajes de programación para IA van y vienen. Sin embargo, Python se mantiene como uno de los principales lenguajes de referencia debido a dos características principales: i) su simplicidad como entorno de programación, y ii) la accesibilidad que ofrece en términos de funcionalidad ya existente.

Debido a su naturaleza interpretada, es posible instalar y ejecutar Python en las principales plataformas de desarrollo. No existe la necesidad de disponer de un ompilador nativo para traducir las instrucciones, ya que el intérprete de Python se encarga de ello. Además, Python es un lenguaje fácil de usar y entender, incluso para personas que no tienen una formación técnica en el ámbito del desarrollo software. A modo de ejemplo, y desde el punto de vista de la sintaxis del lenguaje, no hay necesidad de puntos y comas constantes, ni de largas declaraciones de tipos, ni de reinventar funciones comunes.

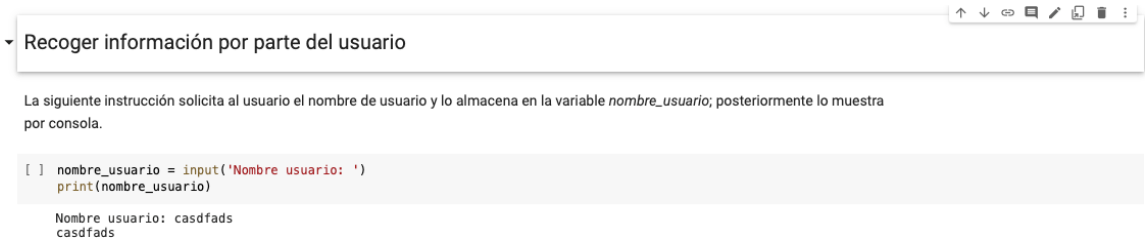
Pero, probablemente, una de las ventajas más importantes de usar Python deriva de su amplio ecosistema funcional. Existe todo un conjunto de bibliotecas externas que facilitan la vida a los desarrolladores de aplicaciones. A la hora de abordar un desarrollo de IA, será prácticamente seguro que ya exista una biblioteca en Python que dé soporte, al menos a nivel general.

Fundamentos de Python

Python es un lenguaje de programación multi-paradigma, donde la programación estructurada (particularmente procedural) y la POO (Programación Orientada a Objetos) tienen mayor presencia que respecto a la programación funcional. Además, Python es un lenguaje basado en el tipado dinámico, es decir, la comprobación de los tipos de datos se hace en tiempo de ejecución. Por otro lado, la gestión de memoria está delegada en un recolector de basura. La biblioteca estándar de Python ofrece un conjunto muy amplio de estructuras de datos y de funcionalidad, que hace que a Python se le asocie el término de batteries included.

A continuación, se incluye un listado de las principales características de Python:

- Sintaxis elegante, que facilita la escritura y lectura de programas.
- Facilidad de uso, que convierte a Python en un lenguaje ideal para prototipar y construir soluciones a medida, sin sacrificar el rendimiento.
- Amplia biblioteca estándar, que da soporte a las tareas de desarrollo más comunes.
- Lenguaje pegamento, que simplifica la extensión del mismo mediante el desarrollo de nuevos módulos escritos en lenguajes compilados, como C o C++.
- Modo interactivo, que facilita la prueba de pequeños fragmentos de código.
- Alta portabilidad, siendo posible ejecutar Python en Windows, Linux, MacOS X, y Unix.
- Licencia open-source. Tanto el software de Python como su documentación está licenciado mediante el PSF (Python Software Foundation) License Agreement, por lo que es posible usar y modificar Python en proyectos comerciales.



Bibliotecas Python para IA

En el ámbito particular de bibliotecas Python para IA, destacan los que se exponen a continuación, los cuales se han clasificado en grandes categorías vinculadas a las temáticas más populares cuando se aplican técnicas de IA.

Redes neuronales

Posteriormente veremos en mayor detalle los fundamentos de las redes neuronales. En este punto basta con entender una red neuronal como una estructura artificial cuyo funcionamiento se asemeja a las redes neuronales que conforman un sistema compuesto por neuronas conectadas entre sí que reciben y transmiten señales entre ellas.

- **FANN (Fast Artificial Neural Network).** Biblioteca de redes neuronales de código abierto, que permite utilizar redes neuronales artificiales multi-capa. Es multi- plataforma y ofrece un entorno de trabajo que facilita el proceso de entrenamiento de la red neuronal. Además de proporcionar binding para Python, es posible utilizar FANN con más de una decena de lenguajes de programación.
- **ffnet**. Biblioteca de IA en Python para implementar redes neuronales pre- alimentadas (feed-forward). Los datos de entrenamiento se pueden visualizar mediante una interfaz gráfica de usuario específica. Integra una función de normalización automática de datos, que ahorra tiempo en la etapa de preprocesamiento. ffnet implementa sus funciones principales en Fortran, aumentando el

rendimiento en comparación con soluciones que están implementadas en Python de manera nativa.

- **PyTorch**. Biblioteca optimizada para deep learning que hace uso de GPUs y CPUs. PyTorch se puede entender como un sustituto del paquete NumPy que se aprovecha de la capacidad computacional ofrecidas por las GPUs, en lugar de utilizar CPUs.

Aprendizaje automático

- **scikit-learn**. Biblioteca de IA en Python, construida sobre las bibliotecas NumPy, SciPy y matplotlib, que da soporte a la implementación de algoritmos de aprendizaje automático. Tiene algoritmos incorporados para clasificar objetos, construir regresiones, agrupar objetos similares en conjuntos (clustering), reducir la cantidad de variables aleatorias, preprocesar datos e incluso elegir el modelo de aprendizaje final.

- **TensorFlow**. Plataforma de código abierto que ofrece mecanismos de soporte para el aprendizaje automático, incluyendo deep learning. TensorFlow ofrece un ecosistema integral y flexible de herramientas, bibliotecas y recursos diseñados para crear aplicaciones de machine learning. Aunque los cálculos se pueden expresar en Python, internamente se ejecutan en C++ para incrementar el rendimiento, ahorrando tiempo de ejecución y aumentando la velocidad de los programas desarrollados con esta biblioteca.

Procesamiento del lenguaje natural

- **NLTK (Natural Language Toolkit).** Biblioteca de propósito general para el procesamiento de texto escrita en Python. Simplifica el procesamiento lingüístico y ofrece funcionalidad para gestionar la tokenización y el etiquetado de texto. Además, permite la identificación de entidades con nombre e incluso la visualización de árboles de análisis sintáctico.

- **Gensim**. Biblioteca gratuita de código abierto en Python para representar documentos como vectores semánticos, de una eficiente, en términos de consumo de recursos, y sencilla (para facilitar su procesamiento). Gensim está diseñado para procesar textos digitales sin estructurar (texto plano) utilizando algoritmos de aprendizaje automático no supervisado.

Modelado y caracterización de sistemas expertos

- **PyCLIPS**. Extensión para el lenguaje Python que incorpora la funcionalidad completa de CLIPS en aplicaciones de Python. Así, es posible dotar a Python de un motor de inferencia sólido, fiable, ampliamente utilizado y bien documentado. CLIPS representa una de las implementaciones de referencia en lo que se refiere a sistemas expertos. Tiene un sistema de inferencia basado en reglas de encadenamiento hacia adelante, así como todas las construcciones imperativas y orientadas a objetos, lo cual permite un control total del flujo de ejecución.

Visión por computador

- **OPENCV (Open Source Computer Vision Library).** Biblioteca de referencia para el desarrollo de aplicaciones de visión computador, la cual ofrece bindings para los lenguajes de programación más populares, incluyendo Python. Se puede definir como una biblioteca de software de visión por computador y aprendizaje automático de código abierto. OPENCV se construyó para proporcionar una infraestructura común para las aplicaciones de visión por computador y para acelerar el uso de la

percepción automática en los productos comerciales. La biblioteca cuenta con más de 2.500 algoritmos optimizados, que incluyen un amplio conjunto de algoritmos de visión por ordenador y de aprendizaje automático, tanto clásicos como de última generación. Estos algoritmos pueden utilizarse para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en vídeos, seguir los movimientos de la cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D a partir de cámaras estereoscópicas, unir imágenes para producir una imagen de alta resolución de toda una escena, encontrar imágenes similares a partir de una base de datos de imágenes, eliminar los ojos rojos de las imágenes tomadas con flash, seguir los movimientos de los ojos, reconocer paisajes y establecer marcadores para superponerlos a la realidad aumentada, entre otras cuestiones.

6. Lenguajes de marcado

Los lenguajes de marcas o marcado nacen de la necesidad de aportar o anotar información sobre el contenido de un documento. Por ejemplo, si se tiene una lista de productos y se quiere saber cuáles son los precios y stock, es necesario indicar de alguna forma qué parte de cada elemento de la lista se corresponde con el precio y qué parte con el stock.

Para conseguirlo, se pueden emplear diversas formas de codificación de dicho documento. Una de ellas es utilizando etiquetas o marcas que añadan esa información extra que necesitamos. El lenguaje definido por el conjunto de etiquetas es lo que se denomina lenguaje de marcas. A su vez, será necesario establecer una serie de reglas que le den validez al lenguaje de marcas que vayamos a utilizar. Estas reglas vendrán introducidas por la sintaxis del lenguaje y su semántica.

A diferencia de los lenguajes de programación, los lenguajes de marcas únicamente se utilizan para estructurar y representar información. Es decir, no son lenguajes que nos permitan construir aplicaciones que realicen alguna funcionalidad lógica. Sin embargo, la línea divisoria entre lo que es un lenguaje de programación y lo que es un lenguaje de marcado puede resultar difusa, por lo que en caso de duda, podemos referirnos a ambos tipos como lenguajes informáticos.

Los lenguajes de marcas más extendidos son HTML y XML, aunque existen multitud de derivados del segundo que lo toman como base para construir otros lenguajes de marcas. Es por ello que XML suele definirse como un meta-lenguaje de marcas.

6.1. Clasificación de los lenguajes de marcas

Existen tres categorías principales de lenguajes de marcado, propuestas por Coombs et al. en 1987 y por Bray en 2003:

- **Presentacionales**: se trata de aquellos lenguajes de marcas utilizados en procesadores de texto tradicionales, es decir, aquellos que utilizan códigos binarios empotrados en los documentos junto al texto para producir una salida modificada. Estas marcas normalmente se encuentran ocultas al usuario que está trabajando en el documento, abstrayéndole del procesamiento interno que realiza el software para construir el documento. El aspecto presentacional se consigue mediante la conversión de esos códigos binarios internos a aspectos del documento como, por ejemplo, palabras en negrita, incrustado de imágenes en línea y construcción de tablas de contenido, entre otros. Un ejemplo de software que utiliza este tipo de lenguajes de marcas podría ser Microsoft Word.

- **Procedurales**: las marcas son incrustadas en línea con el texto del documento para indicar las instrucciones que otros programas tendrán que realizar cuando procesen el texto. De esta forma, se espera que el procesador lea el archivo completo de principio a fin, siguiendo las instrucciones que encuentre para manipular el texto. Los archivos de texto que utilizan este tipo de marcas suelen editarse con las marcas visibles para que puedan ser manipuladas a discreción del usuario. Un ejemplo sería el lenguaje PostScript, utilizado como estándar en los sistemas de impresión para imprimir documentos complejos.

- **Descriptivos**: también conocidos como semánticos, este tipo de lenguajes de marcas se utiliza para realizar anotaciones que describan partes del documento, en lugar de indicar cómo deberían procesarse. Así, el objetivo es el de desacoplar la estructura del documento de cualquier tratamiento o interpretación particular del mismo. Como buena práctica, se recomienda que este tipo de lenguajes de marcas se utilice para describir el texto conceptualmente (p. ej. indicando que el texto es una citación bibliográfica), en lugar de visualmente (p. ej. indicando que el texto debería mostrarse en negrita). Un ejemplo de este tipo lenguajes sería HTML, utilizado para crear páginas web.

En los últimos años se han desarrollado una serie de pequeños lenguajes de marcado, generalmente no estandarizados, que permiten a los usuarios crear texto con formato a través de los navegadores web, como los utilizados en las wikis y en los foros web y comúnmente denominados lenguajes de marcas ligeros. Por ejemplo, Markdown y BBCode pertenecerían a esta familia de lenguajes.

6.2. EXtensible Markup Language (XML)

XML es un lenguaje de marcas que únicamente nos permite almacenar o transportar datos de manera estructurada; su presentación se realizará con alguna otra herramienta. Se basa en etiquetas; definidas entre “<” y “>” si son de apertura, y entre “</” y “>” si son de cierre.

Todas las etiquetas en XML deben cerrarse, ya sea especificando su pareja o mediante etiquetas autocerradas. XML sólo va a permitir un elemento raíz del que colgarán, jerárquicamente, todos los demás. Por ejemplo:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<carta>
   <producto>
      <nombre>Ración de paella</nombre>
      <precio>9.90</precio>
   </producto>
   <producto>
      <nombre>Bocadillo de lomo</nombre>
      <precio>3.95</precio>
   </producto>
</carta>
```

En el anterior ejemplo podemos ver el elemento raíz, que es una carta de un restaurante (etiqueta principal <carta> y cierre </carta>) que contiene varios productos (etiqueta

<producto>) dentro. El primer producto, por ejemplo, tiene de nombre “Ración de paella” (contenido etiqueta <nombre>) y de precio “9.90” (contenido etiqueta <precio>). De esta forma, estamos mostrando la información de la carta de una forma estructurada e indicando el significado de cada una de las piezas de información que aparecen (“Ración de paella”, “9.90”, “Bocadillo de lomo”, “9.90”).A su vez, una etiqueta, puede tener atributos. En el siguiente fragmento de XML, que representa el stock en una tienda de libros, cada libro tiene un atributo llamado category con un valor de tipo cadena asignado.

```
<bookstore>
<book category="children">
<title>Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
```

6.3. MarkDown

MarkDown es un lenguaje de marcas que nos permite intercalar símbolos entre la información de un fichero con el objetivo de ofrecer estilo o semántica a la información. Esta herramienta fue creada en 2004 por John Gruber, y se distribuye de manera gratuita bajo una licencia BSD.

En el siguiente ejemplo veremos un documento markdown con los elementos más utilizados.

```
1  ## Título
2  ### Subtítulo
3
4  Este es un ejemplo de texto que da entrada a una lista genérica de elementos:
5  - Elemento 1
6  - Elemento 2
7  - Elemento 3
8
9  Este es un ejemplo de texto que da entrada a una lista numerada:
10 1. Elemento 1
11 2. Elemento 2
12 3. Elemento 3
13
14 Al texto en Markdown puedes añadirle formato como **negrita** o *cursiva* de una manera muy sencilla.
```

Las # almohadillas son uno de los métodos utilizados en Markdown para crear encabezados. Debes usarlos añadiendo uno por cada nivel. En el anterior ejemplo (línea 1 y 2) podemos ver un título de segundo nivel y otro de tercer nivel, respectivamente.

Para crear listas desordenadas utiliza * asteriscos, - guiones, o + símbolo de suma (ver líneas 5-7).

Para crear listas ordenadas debes utilizar la sintaxis de tipo: «número.» Al igual que ocurre con las listas desordenadas, también podrás anidarlas o combinarlas (ver líneas 10-12).

Debes conocer…

Durante las prácticas utilizaremos MarkDown para añadir explicaciones dentro de nuestro código. Concretamente utilizaremos cuadernos Jupyter dentro de Google Collab. Por ello deberás comprender la utilización de este lenguaje dentro de dichos cuadernos. Para ello te ofrezco un tutorial en el aula virtual.

Para saber más …

Puedes ver más información en el siguiente enlace:

https://markdown.es/sintaxis-markdown/

7. Bibliografía

Materiales formativos FP Online del Ministerio de Educación y Formación Profesional. Módulo de Programación.

Materiales formativos FP Online del Ministerio de Educación y Formación Profesional. Módulo de Entornos de Desarrollo.

Materiales formativos FP Online del Ministerio de Educación y Formación Profesional. Módulo de Programación de Inteligencia Artificial.