

# **Programación de Inteligencia Artificial**

*Conforme a contenidos del «Curso de Especialización en Inteligencia Artificial y Big Data»*



**Programación de  
Inteligencia\_Artificial**

**Universidad de Castilla-La Mancha**

Escuela Superior de Informática  
Ciudad Real



**No todo es blanco o negro.** Note que en un mismo proyecto software pueden convivir varios lenguajes de programación. Por ejemplo, se podría utilizar un lenguaje compilado para implementar el núcleo funcional del proyecto y un lenguaje interpretado para aquellas cuestiones menos relevantes desde el punto de vista computacional.

#### 1.1.4. Resumen

En esta sección se ha ofrecido una visión general del proceso de desarrollo de software, entendido como un proceso complejo que va más allá de la programación. Particularmente, se han agrupado las fases identificadas en cuatro grandes etapas: especificación, ii) diseño, iii) desarrollo y iv) pruebas. Asimismo, se ha establecido la relación de este proceso con dos destrezas fundamentales que deberían interiorizarse por parte de cualquier ingeniero software: i) el uso de metáforas como mecanismo para manejar de manera eficaz la abstracción y ii) la importancia de la simplicidad como eje fundamental para diseñar, programar y mantener código. Esta introducción general ha quedado complementada con una visión general sobre buenas prácticas de desarrollo y referencias a cuestiones relacionadas con *clean code*.

Por otro lado, se ha ofrecido al lector una comparativa general de lenguajes de programación compilados e interpretados, relacionando la misma con su uso como herramienta para desarrollar proyectos de IA. En la siguiente sección se abordarán las características deseables en un lenguaje de programación para IA.

## 1.2. Programación de Inteligencia Artificial

La programación de aplicaciones de IA comparte la gran mayoría de aspectos y cuestiones a considerar por parte de otro tipo de proyecto donde el desarrollo software sea el principal protagonista. Sin embargo, es posible identificar un conjunto de características concretas que han de valorarse cuando se aborda un proyecto de IA. Particularmente, en esta sección se discute un listado de 5 características específicas a tener en cuenta a la hora de elegir un lenguaje de programación. Este listado se complementa con fragmentos de código y con un caso práctico diseñado para ilustrarlas y compararlas cuando se usan los lenguajes de programación C y Python, respectivamente.

### 1.2.1. Consideraciones previas

En los últimos años, resulta evidente que la transformación digital y la automatización han causado una revolución en términos industriales. Uno de los actores cuyo peso ha ido creciendo significativamente es la Inteligencia Artificial. Por lo tanto, la demanda de profesionales que sepan cómo diseñar, desarrollador, validar y mantener proyectos basados en IA se irá incrementando progresivamente en el futuro.



**Ética en IA.** Más allá de la parte técnica vinculada al desarrollo de software e IA, existen otras consideraciones, como la ética, que están inherentemente vinculadas al uso de IA. Este capítulo, no obstante, está centrado en la parte técnica y de desarrollo.

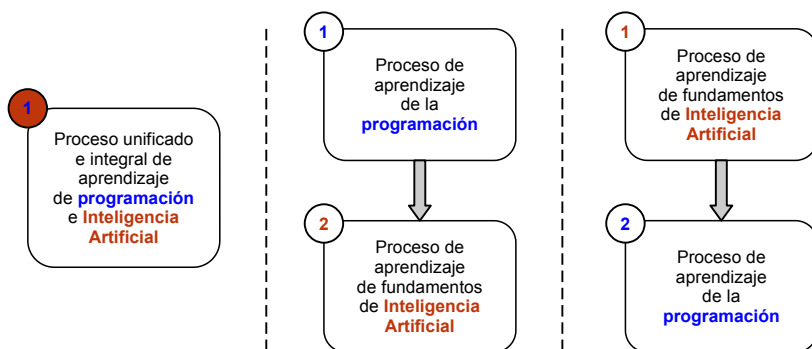
En este contexto, a la hora de **aprender las competencias necesarias para la programación de software para IA**, existen diversos caminos que se podrían recorrer (ver figura 1.6):

1. Aprender a programar software de manera simultánea a conocer los fundamentos en los que se basa la IA.
2. Aprender a programar software de manera previa a conocer los fundamentos de IA.
3. Aprender los fundamentos de IA de manera previa a abordar los aspectos que conforman el núcleo de la programación actual.

Cada opción tiene sus ventajas y desventajas. La primera opción puede resultar demasiado ambiciosa, debido a que sería necesario abordar tanto la complejidad que representa el aprendizaje de la programación, incluyendo la capacidad de pensar de forma abstracta, como los fundamentos de las principales técnicas y algoritmos de IA. No obstante, si el aprendizaje está guiado por proyectos aplicados y concretos, es posible abordarlo desde un punto de vista incremental.

La segunda opción es probablemente la más extendida en gran cantidad de currículos vinculados al ámbito de *Computer Science*. En este sentido, disponer de una buena base de programación (particularmente de algoritmia, análisis de complejidad y diseño de estructuras de datos), facilita el estudio y entendimiento de los fundamentos en los que se apoyan gran parte de las técnicas de IA existentes. En este sentido, el hecho de disponer de la programación como herramienta lista para usarse en el ámbito de la IA puede simplificar la comprensión y aplicación práctica de dichas técnicas a problemas concretos.

Por otro lado, la tercera opción estaría más vinculada con un perfil donde los fundamentos matemáticos pueden tener más presencia, al menos en primera instancia, de forma que el aprendiz domine primero la parte algorítmica y, posteriormente, aprenda a programar tanto a nivel transversal como a nivel más aplicado a la IA.



**Figura 1.6:** Esquema conceptual de los diferentes caminos de aprendizaje a la hora de aprender IA y desarrollo software.

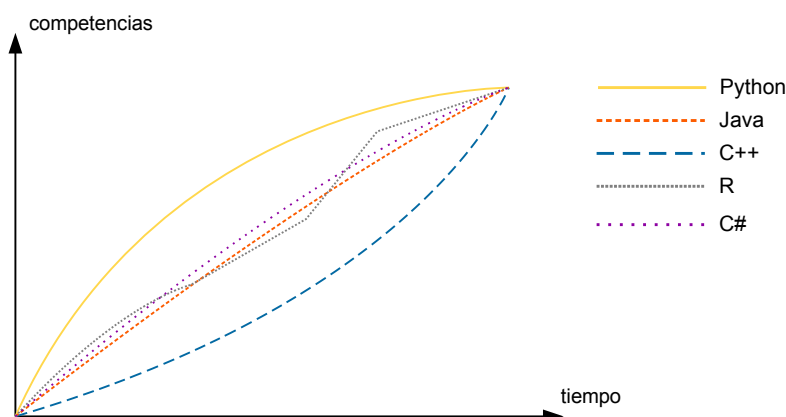


**Introduction to Machine Learning.** Este término es uno de los más buscados cuando una persona decide abordar el aprendizaje automática desde una perspectiva de la programación. Le recomiendo que revise y reflexione sobre cómo se aborda en la literatura existente, incluyendo las asignaturas que llevan por título su nombre. A modo de referencia, la asignatura del MIT (Massachusetts Institute of Technology) *Introduction to Machine Learning* tiene 3 prerrequisitos: i) Programación (mencionando específicamente el lenguaje Python), ii) Cálculo y iii) Álgebra Lineal.

### 1.2.2. Características deseables en un lenguaje 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 código.
2. **Capacidad de prototipado rápido**, debido a la propia naturaleza de las aplicaciones de IA, que suelen variar considerablemente su implementación 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.



**Figura 1.7:** Curvas de aprendizaje de 5 lenguajes de programación utilizados para programar IA.

4. **Existencia de bibliotecas para IA**, debido a la importancia de reutilizar código existente que acelere la generación de prototipos.
5. **Comunidad de desarrollo**, debido a las ventajas que ofrece compartir experiencias y soluciones a problemas previamente abordados por otra persona.

A continuación, estas características se abordan con más detalle.

## Simplicidad

La importancia de la simplicidad, en un contexto general de desarrollo de software, ya introdujo en la sección 1.1.1. La simplicidad como características deseable de un lenguaje de programación está relacionada tanto con la curva de aprendizaje necesaria para dominarlo como con la sencillez a la hora de aplicar mecanismos propios del lenguaje.

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. La figura 1.7 muestra, de manera gráfica, la curva de aprendizaje de 5 lenguajes vinculados al desarrollo de aplicaciones de IA.

En relación a los **mecanismos ofrecidos por el propio lenguaje de manera nativa**, es decir, los que forman parte de su estándar, idealmente deberían ser fáciles de entender y utilizar de manera productiva. Los mecanismos que están diseñados para mejorar la productividad de un desarrollador suelen implicar un periodo de tiempo en el que la curva de aprendizaje se hace más pronunciada, ofreciendo beneficios a más largo plazo. Un ejemplo clásico es el mecanismo de plantillas ofrecido por el lenguaje de programación C++ para manejar la programación genérica [Str13], es decir, independiente del tipo de datos que manejen las funciones o las clases. A modo de ejemplo, el lector podría pensar en un algoritmo que funciona para clasificar, automáticamente, tipos de objetos diferentes en base a una función de utilidad. Si bien el beneficio de esta característica resulta evidente, la depuración de código con plantillas introduce una complejidad no desdeñable. El listado 1.1 muestra un ejemplo de uso de plantillas de función en C++. Se invita al lector a probarlo y a introducir, a propósito, un error de compilación.

### Capacidad de prototipado rápido

El desarrollo de aplicaciones de IA suele iniciarse con una fase de prototipado que permita validar un concepto o algoritmo de la forma más ágil posible. En otras palabras, los primeros prototipo software no suelen prestar demasiada atención a la calidad del código o a su rendimiento.

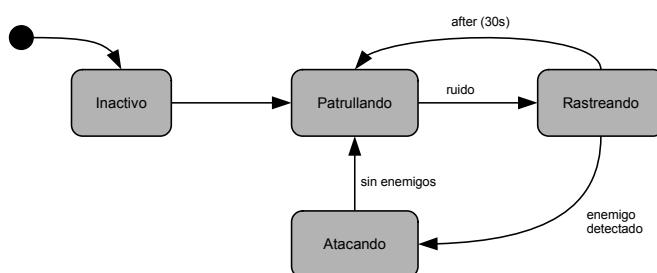
Listado 1.1: Ejemplo de función en C++ que hace uso de plantillas

```
1 #include <iostream>
2 using namespace std;
3
4 template <class T>
5 T GetMax (T a, T b) {
6     T result;
7     result = (a > b) ? a : b;
8
9     return (result);
10 }
11
12 int main () {
13     int i = 5, j = 6, k;
14     long l = 10, m = 5, n;
15
16     k = GetMax<int>(i, j);
17     cout << k << endl;
18
19     n = GetMax<long>(l, m);
20     cout << n << endl;
21
22     return 0;
23 }
```

Por el contrario, resulta más relevante que el lenguaje escogido facilite tanto la generación de la primera versión como su adaptación. Esta característica de prototipado rápido se puede relacionar con tres aspectos fundamentales:

1. La **experiencia del programador de IA con un determinado lenguaje** de programación. Por ejemplo, si el programador tiene una gran experiencia con Java, podría escoger este lenguaje para implementar su primer prototipo, ya que le resultará relativamente fácil programar el código de un determinado algoritmo. Si la experiencia del programador con un lenguaje específico no es relevante, el segundo y el tercer aspecto, descritos a continuación, toman más relevancia.
2. La capacidad del lenguaje para **ejecutar y probar código de manera ágil**. Este aspecto está directamente vinculado con el uso de un lenguaje interpretado, donde el código se ejecuta *al vuelo* sin necesidad de un proceso de compilación previo.
3. La existencia de **código que se pueda reutilizar** por parte del desarrollador de manera directa. Debido a la importancia de esta cuestión en el contexto de desarrollo de aplicaciones de IA, la misma se discute más adelante con mayor nivel de detalle.

A modo de ejemplo, considere la necesidad de desarrollar un prototipo que permita modelar un sencillo autómatas o máquina de estados. Por cuestiones de simplicidad, puede asociar un autómatas con un modelo de comportamiento basado en estados, eventos y acciones. Un estado sirve para representar una situación; un evento es un suceso que dispara el cambio de un estado a otro; una acción es un tipo de comportamiento. La figura 1.8 muestra un sencillo autómatas para modelar el comportamiento de un enemigo artificial en un juego de espionaje.



**Figura 1.8:** Máquina de estados que define el comportamiento de un enemigo en un juego.

Si necesitara ofrecer una implementación que soportara la máquina de estados de la figura 1.8, en un contexto de prototipar y validar el comportamiento de un enemigo artificial, probablemente optaría por un lenguaje que facilitara ambas acciones. En este contexto, el listado 1.2 muestra una posible implementación, en el

lenguaje Python, de una clase base que representa un estado genérico. Como se puede apreciar, en la línea ③ se declara la lista que contendrá el nombre de los estados a los que se puede transitar desde el estado actual. Por otro lado, la función `switch`, implementada entre las líneas ⑤ y ⑩, es la responsable de la transición efectiva entre estados, si esta es posible (vea la condición de la línea ⑥).

Listado 1.2: Implementación sencilla de una máquina de estados en Python (I)

```

1 class State(object):
2     name = "General State"
3     allowed_transitions = []
4
5     def switch(self, new_state):
6         if new_state.name in self.allowed_transitions:
7             self.__class__ = new_state
8             print 'Current:', self, ' => switched to new state', new_state.name
9         else:
10            print 'Current:', self, ' => switching to', new_state.name, 'is not allowed.'
11
12    def __str__(self):
13        return self.name

```

Esta clase base se podría especializar, a través de relaciones de herencia, para modelar los estados contemplados en la figura 1.8, así como para definir las transiciones permitidas. Estos aspectos se reflejan en el listado 1.3. Como puede comprobar, y a modo de ejemplo, desde el estado *Patrolling* solo se pueden realizar transiciones hasta el estado *Tracking*.

Listado 1.3: Implementación sencilla de una máquina de estados en Python (II)

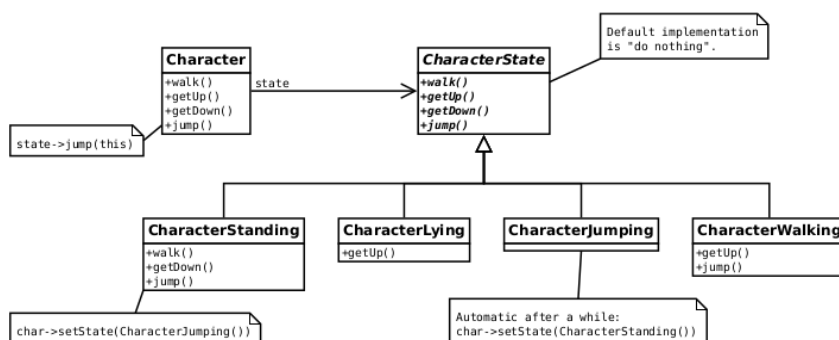
```

1 class Inactive(State):
2     name = "Inactive"
3     allowed = ['Patrolling']
4
5 class Patrolling(State):
6     name = "Patrolling"
7     allowed = ['Tracking']
8
9 class Tracking(State):
10    name = "Tracking"
11    allowed = ['Patrolling', 'Attacking']
12
13 class Attacking(State):
14    name = "Attacking"
15    allowed = ['Patrolling']

```

Finalmente, ya es posible modelar y utilizar una máquina de estados, considerando la definición general de estado y la funcionalidad básica de transitar de un estado a otro. El listado 1.4 muestra el código, junto con un conjunto de instrucciones de prueba. Note cómo en la línea ⑤ se define la función `change` que, a su vez, delega en la función `switch` del estado actual el cambio de estado hacia `new_state`.





**Figura 1.9:** Representación gráfica del patrón de diseño State aplicada al modelado de la IA de un enemigo virtual en un videojuego [VC15].

#### Listado 1.4: Implementación sencilla de una máquina de estados en Python (III)

```

1 class StateMachine(object):
2     def __init__(self):
3         self.state = Inactive()
4
5     def change(self, new_state):
6         self.state.switch(new_state)
7
8 if __name__ == "__main__":
9     enemy_basic_AI = StateMachine()
10    enemy_basic_AI.change(Inactive)
11    enemy_basic_AI.change(Patrolling)
12    enemy_basic_AI.change(Tracking)
13    enemy_basic_AI.change(Attacking)
  
```

Como ha podido comprobar, gracias a unas cuantas líneas de código y en relativamente poco tiempo, es posible ofrecer una implementación base de lo que sería el patrón de diseño *State* [GHJV94], ampliamente utilizado en la programación de IA y representado visualmente en la figura 1.9. En este ejemplo concreto se ha utilizado el lenguaje de programación Python; se invita al lector a prototipar este ejemplo en el lenguaje de programación que le resulte más cercano y a comparar el código obtenido con el discutido previamente.



**Transiciones dirigidas por eventos y transiciones temporales.** ¿Cómo aumentaría el prototipo introducido para manejar transiciones que ocurren cuando se dispara un evento o cuando pasa un determinado tiempo?

## Legibilidad

Los lenguajes de programación utilizados mayoritariamente para el desarrollo de aplicaciones de IA son lenguajes de programación de alto nivel. En otras palabras, ofrecen un alto **nivel de abstracción** para que el programador no tenga que preocuparse de dónde y cómo se ejecuta el código que implementa. La cantidad de abstracción proporcionada determina cómo de alto, medio o bajo es el nivel vinculado al lenguaje. Así, y con carácter general, un lenguaje de alto nivel tiene un grado de legibilidad alto, ya que las abstracciones ofrecidas por el mismo estarán más cercanas a la hora de entender código.

El precio a pagar por utilizar un lenguaje de programación de alto nivel reside en las limitaciones existentes a la hora de optimizar código para una determinada máquina o considerando restricciones de implementación (memoria necesaria o tiempo de ejecución). No obstante, y con carácter general, esta contrapartida no tiene un impacto relevante en el contexto del desarrollo de aplicaciones de IA debido a dos motivos: i) en las primeras fases del desarrollo de IA el rendimiento del código no suele ser relevante, y ii) las herramientas que generan código máquina a partir de código fuente de alto nivel, especialmente los compiladores, son capaces de generar un resultado lo suficientemente optimizado en la mayoría de los casos.

Por otro lado, la legibilidad está relacionada con las facilidades que ofrece un lenguaje para facilitar el seguimiento del código de una manera lógica. Un ejemplo concreto sería la indentación de código en Python, la cual se refiere al uso de espacios al principio de una línea de código. Python utiliza sangrías para resaltar bloques de código. El espacio en blanco es el carácter utilizado para incluir sangrías. Así, todas las sentencias con la misma distancia a la derecha pertenecen al mismo bloque de código. Si un bloque tiene que ser anidado en un nivel de profundidad mayor, simplemente se indenta más a la derecha.

El listado 1.5 muestra un ejemplo de indentación en Python. El simple hecho de introducir espacios ya facilita el seguimiento de código. Note, por ejemplo, cómo en la línea ⑫ y sucesivas se aprecia visualmente el bloque de código asociado al bucle *while*.

## Existencia de bibliotecas para IA

La IA es un campo de trabajo amplio que se apoya sobre un conjunto de dominios o materias. Algunas están relacionadas con las matemáticas, como por ejemplo la lógica, la probabilidad, las matemáticas continuas; otras con aspectos de obtención y procesamiento de información, como la percepción multi-sensorial, el razonamiento, el aprendizaje; otras con cuestiones relativas a ciencias sociales, como las relaciones de confianza, la seguridad, la ética, el bien social.

Listado 1.5: Ejemplo en Python que refleja las bondades del uso de indentación

```
1 # Incluir módulos de pygame...
2 WIDTH, HEIGHT = 1200, 800
3 FPS = 30
4
5 if __name__ == "__main__":
6     pygame.init()
7
8     clock = pygame.time.Clock()
9     soccerField = SoccerField(WIDTH, HEIGHT)
10
11     while True:
12         tick_time = clock.tick(FPS)
13         for event in pygame.event.get():
14             if event.type == QUIT:
15                 pygame.quit()
16                 sys.exit()
17
18         soccerField.render()
19         pygame.display.update()
```

Algunos autores, como Russell y Norvig en el libro *Artificial Intelligence: A Modern Approach* [SN19], unifican todas estas cuestiones alrededor del concepto de agente inteligente, entendiendo así la IA como el estudio de los agentes que reciben estímulos de un entorno y llevan a cabo acciones (ver figura 1.2). La gestión y procesamiento de estos estímulos determina el comportamiento de estos agentes, existiendo agentes reactivos, agentes que planifican en tiempo real, agentes basados en la teoría de la decisión o agentes basados en algoritmos de *deep learning*. En cualquier caso, es importante destacar que gran parte de estos fundamentos se basan en **algoritmos que han sido implementados en múltiples ocasiones**, empleando diferentes lenguajes de programación.



**Programas = Algoritmos + Estructuras de Datos.** Si bien en términos de re-utilización de código para la programación de IA solemos pensar en términos de algoritmos ya implementados, también es importante considerar el soporte que ofrece un lenguaje en relación a las estructuras de datos que ofrece. Recuerde que una estructura de datos bien diseñada facilitará la programación de un algoritmo, independientemente del lenguaje de programación empleado.

Por ejemplo, considere un algoritmo de búsqueda binaria sobre un array ya ordenado. El listado 1.6 muestra un ejemplo de implementación en Python. Como puede imaginar, resultaría ventajoso disponer de una versión lista para usarse de dicho algoritmo, en lugar de implementarlo desde cero.

Listado 1.6: Posible implementación de un algoritmo de búsqueda binaria


```
1 def binary_search (vector, low, high, x):
2
3     # Caso base de la recursividad
4     if high >= low:
5         mid = (high + low)
6
7         # ¿Elemento en la posición central del array?
8         if vector[mid] == x:
9             return mid
10
11        # Si el elemento central es mayor que x,
12        # x solo puede estar en el subarray izquierdo
13        elif vector[mid] > x:
14            return binary_search(vector, low, mid - 1, x)
15
16        # Si no, x solo puede estar en el subarray derecho
17        else:
18            return binary_search(vector, mid + 1, high, x)
19
20    # x no está en el vector de entrada
21    else:
22        return -1
```

En este sentido, la existencia de bibliotecas o módulos que faciliten la programación de IA se convierte en una característica deseable a la hora de elegir, o utilizar, un determinado lenguaje de programación. Particularmente, debería prestar especial atención a la **existencia de bibliotecas que cubran los siguientes temas:**

- Redes neuronales.
- Aprendizaje supervisado y no supervisado.
- Procesamiento natural.
- Procesamiento de textos.
- Modelado y caracterización de sistemas expertos.
- Procesamiento matemático, particularmente estadística y probabilidad.
- Visión por computador, debido a la relación del análisis y procesamiento de imágenes con la IA.
- Simuladores y motores de física.

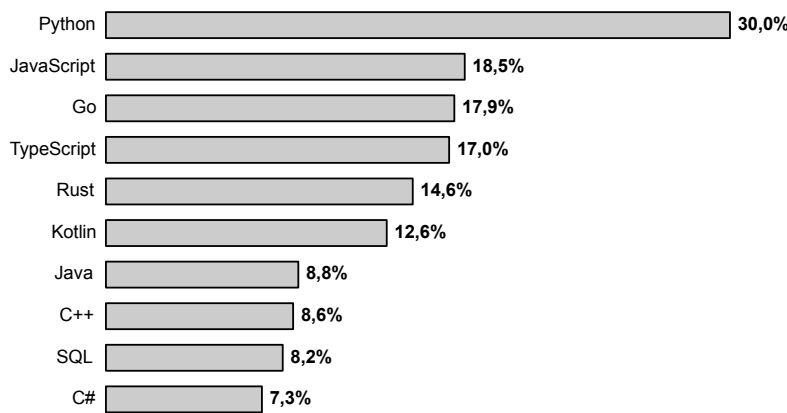
Comunidad de desarrollo

La **popularidad de un lenguaje** es definitivamente una característica que ha de evaluar a la hora de elegir un lenguaje de programación como herramienta para solucionar un problema, ya esté relacionado o no con la IA. Si un lenguaje es popular, el número de programadores que lo estén usando será elevado. Cuanto mayor sea el número de usuarios de un lenguaje de programación, mayor será la probabilidad de que alguno de ellos se haya enfrentado a un problema similar al que usted esté afrontando en un momento determinado.



**El efecto Stack Overflow.** En los últimos años han proliferado los sitios web que permiten lanzar preguntas y ofrecer respuestas relacionadas con aspectos del desarrollo de software, desde una perspectiva transversal. Una de las referencias, por excelencia, es Stack Overflow<sup>4</sup>.

Así pues, la existencia de una comunidad de desarrollo amplia y activa es una característica deseable a la hora de elegir un lenguaje de programación para IA. Como ya se introdujo anteriormente en este mismo capítulo, existen índices, como el índice TIOBE, que nos permiten conocer la demanda de los lenguajes de programación actuales, factor que se puede utilizar como posible indicador del nivel de comunidad existente en torno a ellos.



**Figura 1.10:** Listado de los 10 lenguajes de programación más deseados, de acuerdo a un estudio realizado por StackOverflow, representado en porcentaje de desarrolladores que no están desarrollando con una tecnología concreta pero que han expresado un interés en desarrollar con ella. Fuente: Stack Overflow Developer Survey 2020.

Otro aspecto a considerar está relacionado a la **existencia de bibliotecas software** que no formen parte del estándar del lenguaje a considerar. En el caso de aplicaciones de IA, la referencia directa a estudiar sería el número de bibliotecas o proyectos activos en dicho dominio. En esencia, es posible inferir que si existe

una comunidad de desarrolladores preocupados por mejorar la funcionalidad asociada a un lenguaje, entonces será más probable que la propia comunidad sea más receptiva a la hora de dar soporte a otros programadores que están comenzando a aprender un lenguaje.

Finalmente, otro indicador relevante que se puede estudiar es el **número de programadores en activo** vinculados a cada lenguaje. No obstante, la popularidad de un lenguaje en un determinado momento temporal puede representar una mejor aproximación a la hora de buscar contenido actualizado.

### 1.2.3. Caso práctico. Búsqueda de patrones en archivos de texto

En este apartado, se discute un caso práctico en el que se pretenden desarrollar y comparar varias soluciones, utilizando diferentes lenguajes de programación, para resolver un sencillo problema de búsqueda de patrones en archivos de texto. Se pretende proporcionar un ejemplo práctico en el que sea posible identificar las características deseables para la programación de IA, previamente discutidas, a la hora de elegir un lenguaje de programación. Aunque se trata de un problema sencillo y con poco alcance, nos servirá como referencia inicial, considerando especialmente las propiedades de simplicidad, capacidad de prototipado rápido y legibilidad.

Para caracterizar el caso práctico se asumen los siguientes **requisitos funcionales**:

1. El texto a procesar se encuentra en un archivo de texto. El nombre de este archivo se proporciona por la línea de comandos como primer argumento.
2. El programa debe ser capaz de identificar e imprimir cuántas veces se encuentra un determinado patrón en el contenido del archivo previamente mencionado. El patrón se proporciona por la línea de comandos como segundo argumento.
3. El programa debe ofrecer un control de errores mínimo. Particularmente, si no se proporciona un nombre de archivo o no se especifica el patrón a buscar, entonces el programa debe mostrar un mensaje de error y finalizar.

El listado 1.7 muestra una posible **implementación en C** del problema planteado, de acuerdo a los requisitos preestablecidos.

Listado 1.7: Implementación en lenguaje C del programa que busca un patrón de texto

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5
6 int main (int argc, char *argv[]) {
7     FILE *fp;
8     char *line, *pattern;
9     int occurrences;
10    size_t len;
11    ssize_t read;
12
13    if (argc < 3) {
14        fprintf(stderr, "Error! You need to provide the filepath and the pattern.\n");
15        fprintf(stderr, "Synopsis: ./02_simple_patterns_error_control <file> <pattern>\n");
16        exit(EXIT_FAILURE);
17    }
18
19    pattern = argv[2];
20    occurrences = 0;
21    len = 0;
22
23    if ((fp = fopen(argv[1], "r")) == NULL) {
24        fprintf(stderr, "Error opening file %s!\n", argv[1]);
25        exit(EXIT_FAILURE);
26    }
27
28    while ((read = getline(&line, &len, fp)) != -1) {
29        if (strstr(line, pattern) != NULL) {
30            occurrences++;
31        }
32    }
33
34    fclose(fp);
35
36    printf("The word %s appeared %d times in the file %s\n",
37        argv[2], occurrences, argv[1]);
38
39    return 0;
40 }
41 }
```

A continuación, se exponen algunas reflexiones con respecto a las **características deseables** previamente comentadas:

1. Simplicidad. El código introduce una cierta complejidad inherente a la hora de manejar el lenguaje C, como por ejemplo el uso de punteros (variables que almacenan direcciones de memoria) para gestionar las líneas del archivo, el patrón y la propia línea de comandos. El programador tendría que estar familiarizado con sus fundamentos y con el uso de los operadores unarios \* y &.

2. Capacidad de prototipado rápido. En el caso del lenguaje C, esta característica está acoplada al nivel de experiencia del programador. El hecho de utilizar un lenguaje compilado, implica la re-compilación del código cada vez que se introduce un nuevo cambio. Adicionalmente, ha sido posible emplear funciones existentes para leer el contenido del archivo y buscar un patrón, respectivamente, pero quizá se echa en falta una mayor agilidad a la hora de procesar archivos y controlar errores.
3. Legibilidad. A nivel global, el código resulta legible y el flujo principal se sigue gracias a los bloques de código definidos. No obstante, la llamada a la función `getline()` en la línea ②28 no ofrece un alto nivel de abstracción. El código relativo al control de errores no está demasiado cerca, desde el punto de vista semántico, de lo que realmente se pretende controlar, es decir, de si se proporcionaron un nombre de archivo y un patrón.

Por otro lado, el listado 1.8 muestra una posible **implementación en Python** del problema planteado, de acuerdo a los requisitos preestablecidos.

**Listado 1.8: Implementación en Python del programa que busca un patrón de texto**

```

1  #!/usr/bin/python3
2
3  # Import module sys
4  import sys
5
6  # Variable to store how many times the pattern was found
7  occurrences = 0
8
9  try:
10     # Open the file whose name is specified in the first argument
11     with open(sys.argv[1]) as f:
12         for line in f:
13             # The words of every line are stored as a list
14             words = line.split()
15             # Check for matches
16             for w in words:
17                 # The pattern to be found is specified in sys.argv[2]
18                 if w == sys.argv[2]:
19                     occurrences += 1
20
21     print("The word", sys.argv[2], "appeared",
22           occurrences, "times in the file", sys.argv[1])
23     f.close()
24
25 except IndexError:
26     print("Error! You need to provide the filepath and the pattern.")
27     print("Synopsis: python3 02_simple_patterns_error_control <file> <pattern>")

```



A continuación, se exponen algunas reflexiones con respecto a las **características deseables** previamente comentadas:

1. Simplicidad. El código es sencillo y compacto, incluso aunque en esta solución se hayan introducido dos bucles for anidados. El número total de líneas es 27, en comparación a las 41 de la solución en lenguaje C.
2. Capacidad de prototipado rápido. En el caso del lenguaje Python, la curva de aprendizaje es asequible, y resulta posible escribir programas como el del listado 1.8 con cierta agilidad. Hubiera sido posible utilizar directamente el intérprete de Python para prototipar el código.
3. Legibilidad. A nivel global, el código resulta muy legible y el flujo principal se sigue gracias a los bloques de código definidos. Note cómo la función de apertura de archivos condiciona, y facilita, la estructura del código. El control de errores mediante bloques try-except también incrementa el nivel semántico en comparación a la solución anterior en C.

Con respecto a las características deseables vinculadas a la existencia de bibliotecas y a la comunidad de desarrollo, el presente caso práctico no resulta lo suficientemente específico para establecer una comparativa más concreta. Por una parte, las bibliotecas utilizadas en las soluciones planteadas forman parte de los estándares de los lenguajes C y Python. Por otra parte, resultaría sencillo identificar tanto en libros académicos como en portales web orientados al desarrollo problemas similares al aquí abordado.



**Más allá de C y Python.** En este punto, se invita al lector a desarrollar una solución al problema discutido en este apartado, empleando un lenguaje de programación diferente a los aquí utilizados.

#### 1.2.4. Resumen

En esta sección se han identificado y discutido 5 características específicas que han de considerarse a la hora de utilizar un lenguaje de programación en el contexto de un proyecto de IA. Estas características son las siguientes: i) simplicidad, ii) capacidad de prototipado rápido, iii) legibilidad, iv) existencia de bibliotecas para IA, y v) comunidad de desarrollo. Esta discusión se ha completado con un caso práctico, vinculado a la búsqueda de patrones de texto e implementado en los lenguajes C y Python.

Por otro lado, se ha invitado al lector a que reflexione acerca de cómo enfocar el aprendizaje de los fundamentos de IA y la programación de software. Esta cuestión no ha de estar relacionada directamente con un determinado lenguaje de programación.

# Bibliografía

---

- [BC04] K. Beck and Andres C. *Extreme Programming Explained: Embrace Change, 2nd Edition*. Addison-Wesley Professional, 2004.
- [Bec04] K. Beck. *Test Driven Development: By Example, 1st Edition*. Addison-Wesley Professional, 2004.
- [GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.
- [HF10] J. Humbel and .D Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.
- [KM01] Nilesh N Karnik and Jerry M Mendel. Centroid of a type-2 fuzzy set. *information SCIences*, 132(1-4):195–220, 2001.
- [Mam74] Ebrahim H Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the institution of electrical engineers*, volume 121, pages 1585–1588. IET, 1974.
- [Mar08] R.C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice-Hall, 2008.
- [McC04] S. McConnell. *Code Complete: A Practical Handbook of Software Construction, 2dn Edition*. Microsoft Press, 2004.
- [SHG<sup>+</sup>14] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D Ebner, C. Vinay, and M. Young. Machine learning: The high interest credit card of technical debt. *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, pages 1–9, 2014.

- [SN19] Russell. S. and P. Norvig. *Artificial Intelligence: A Modern Approach, 4th US ed.* Prentice Hall, 2019.
- [Str13] B. Stroustrup. *The C++ Programming Language, 4th Edition.* Addison Wesley, 2013.
- [VC15] D. Vallejo and Martín C. *Desarrollo de Videojuegos. Un Enfoque Práctico. Módulo 1: Arquitectura del Motor.* Amazon CreateSpace, 2015.
- [Zad99] L.A. Zadeh. Fuzzy logic = computing with words. *Computing with Words in Information, Intelligent Systems* 1:3–23, 1999.