Tema 2. Características de los lenguajes de programación para IA

Lenguajes de alto nivel

Para mitigar los inconvenientes de los lenguajes ensambladores surge una tercera generación de lenguajes conocida como los **lenguajes de alto nivel.**

El principal objetivo de los lenguajes de alto nivel es proporcionar **portabilidad** al permitir desarrollar programas independientes de la máquina sobre la que se ejecutan.

Otro de los principales objetivos es proporcionar una sintaxis similar al **lenguaje natural** con el fin de facilitar el proceso de desarrollo de nuevos programas. Es por esta razón que estos lenguajes se consideran de alto nivel, por estar más cerca del lenguaje humano que del lenguaje de las máquinas.

Los lenguajes de programación han seguido evolucionando eventualmente en una **cuarta y quinta generación** proporcionando características para facilitar aún más la creación de nuevos programas.

Debido a esto, muchos lenguajes que inicialmente eran considerados de alto nivel ahora son considerados de bajo nivel. Ejemplos de **lenguajes de alto nivel** son: Java, JavaScript, Python, Ruby, C++, PHP, etc.

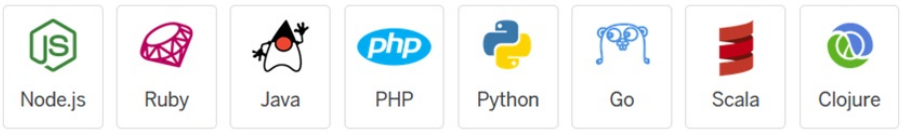


Ilustración 1. Ejemplos de lenguajes de programación de alto nivel. Fuente: https://www.heroku.com/

Al igual que ocurre con los lenguajes ensambladores, los lenguajes de alto nivel deben ser **traducidos a lenguaje máquina** para poder ser ejecutados sobre el hardware. En función de cómo se realiza esa traducción, los lenguajes de alto nivel pueden ser **compilados** o **interpretados.**

Tema 2. Características de los lenguajes de programación para IA

En función de cómo es leído el código programado por la máquina destino, un lenguaje puede ser **compilado** o **interpretado.**

En los **lenguajes compilados** el compilador traduce el código de alto nivel (código fuente) al lenguaje máquina.

En los lenguajes **interpretados,** un software llamado intérprete es el que lee y ejecuta el código en la máquina destino línea a línea.

Compiladores

Los **compiladores** son programas informáticos que traducen el código fuente desarrollado en un lenguaje de alto nivel hacia código máquina, creando así programas ejecutables.

El compilador lleva a cabo la **traducción de un lenguaje a otro** por medio de una serie de fases como el preprocesado, análisis léxico, análisis sintáctico, análisis semántico, etc.

Intérpretes

Un **intérprete** es un programa informático que analiza el código fuente y lo traduce a código máquina instrucción por instrucción cuando se ejecuta.

La principal desventaja de los intérpretes es que la interpretación del código es más lenta que la ejecución del código compilado ya que el intérprete debe analizar cada sentencia del programa y ejecutarla. Es por esta razón que se les asocia un mejor rendimiento a los lenguajes compilados.

Por otro lado, el tiempo tomado para compilar un programa puede ser más elevado que el tiempo usado por los intérpretes en analizar y ejecutar. Por tanto, los intérpretes pueden ofrecer una **mayor flexibilidad en las fases de desarrollo** al evitar tener que esperar que todo el código se compile para poder ejecutarlo y probarlo.

En general, los **lenguajes de programación de alto nivel** tienden a asociarse con una de estas dos categorías, lenguajes **compilados** o **interpretados.** En la práctica también existen lenguajes que emplean ambos procesos, teniendo una fase de compilación a código intermedio o *bytecode* y una fase de interpretación de este *bytecode*. Es el caso de lenguajes como Java o Lisp.

Tema 2. Características de los lenguajes de programación para IA

2.3. Sintaxis y estructura de los lenguajes de programación

Todos los lenguajes de programación comparten elementos comunes, aunque cada uno tiene su propia sintaxis para expresarlos:

- Tipos de datos
- Variables
- Estructuras de datos
- Operadores
- Estructuras de control
  - Condicionales
  - Repetitivas
- Funciones

Tipos de datos

Los programas informáticos trabajan con datos. Esos datos pueden ser de diferente naturaleza o tipo: números, textos, caracteres, valores booleanos, etc.

Un **sistema de tipos** permite clasificar los valores y expresiones en diferentes tipos de datos y las operaciones permitidas con cada uno de ellos. El objetivo de este sistema es verificar la validez de las sentencias programadas cuando se utilizan datos, establece las formas en las que los programas pueden ser escritos.

Tipado fuerte vs. Tipado débil

Un lenguaje de programación **débilmente tipado** permite que un valor de un tipo pueda ser operado como un valor de otro tipo, por ejemplo, intercambiar textos y números. Este mecanismo proporciona flexibilidad al poder trabajar con un mismo dato en diferentes contextos.

Un lenguaje de programación **fuertemente tipado** no permite realizar operaciones sobre tipos equivocados. Este mecanismo proporciona robustez y seguridad al evitar posibles errores derivados de operaciones con distintos tipos.

Tipado estático vs. Tipado dinámico

Un lenguaje de programación tiene **tipado estático** cuando las comprobaciones de tipos se realizan durante la compilación, de forma que si alguna se incumple el código no puede compilarse.

Por el contrario, un lenguaje de programación es **dinámicamente tipad**o si la comprobación de tipos se realiza durante la ejecución del programa en lugar de su compilación.

Variables

Una **variable** en programación es un contenedor de información, contiene un dato en memoria durante la ejecución del programa. Las variables permiten reutilizar un valor en memoria a lo largo de diferentes partes de un programa.

Las variables se conforman con un **identificador** y un **valor.** El identificador es el nombre que el programador asigna con el objetivo de hacer referencia a la misma en diferentes puntos del programa.

El valor es el dato que almacenan. Puede ser desde un valor numérico entero, decimal hasta caracteres o texto.

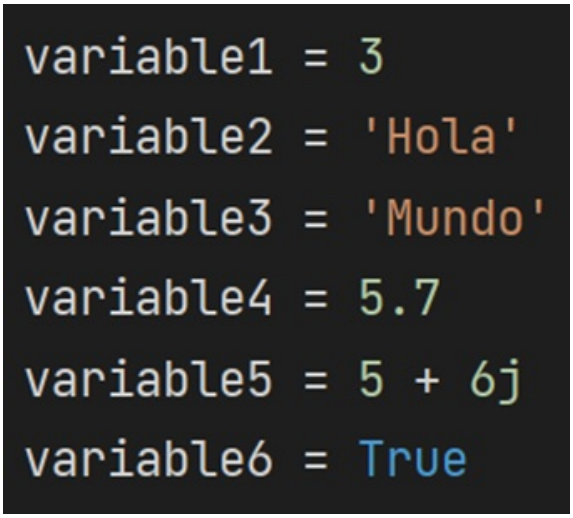


Ilustración 2. Ejemplo de variables en Python.

Creación de variables

Una **variable** puede ser **declarada,** lo que significa que únicamente se crea con un identificador, pero no se le asigna un valor.

Por medio de un **operador de asignación,** una variable previamente declarada puede tomar un valor.

También es posible inicializar una variable al mismo tiempo que es declarada, en una misma línea de código.

Nomenclatura

La **nomenclatura** utilizada para crear el identificador de una variable es elegida por el programador, pero debe seguir una serie de reglas y buenas prácticas:

- No debe contener espacios.
- No debe comenzar por un número.
- No puede ser una palabra reservada del lenguaje de programación que se esté utilizando.

- Debería comenzar por letra minúscula.
- Sigue unas convenciones de formato, por ejemplo:
  - En Java se utiliza el formato camelCase .
  - En Python se utiliza el formato snake_case .

Estructuras de datos

Las variables permiten almacenar un solo dato a la vez. En ocasiones es necesario trabajar con más de un dato al mismo tiempo, por ejemplo, una lista de emails de contacto.

Las estructuras de datos son contenedores específicos para trabajar con más de un dato simultáneamente.

```
# listas
names = ['Alberto', 'Mary', 'Mike']
# tuplas
customer = (1, 'Larry', 34, 50000, True)
# diccionario
employee = {
    "customer_id": 1,
    "store_id": 1,
    "first_name": "MARY",
    "last_name": "SMITH",
}
# conjunto
laptops = {"asus", "dell", "hp", "msi", "lenovo"}
```

Ilustración 3. Ejemplo de estructuras de datos en Python: listas, tuplas, diccionarios y conjuntos.

Tema 2. Características de los lenguajes de programación para IA

Operadores

Los **operadores** son símbolos que indican al compilador o intérprete la ejecución de una determinada operación aritmética, lógica o de comparación.

Los operadores permiten llevar a cabo cálculos y evaluar condiciones haciendo uso de variables.

En cada lenguaje se pueden definir de una manera, ya sea mediante un símbolo o con letras.

```
# Operadores aritméticos
x = 5
y = 7
# suma
result = x + y
# resta
result = x - y
# multiplicación
result = x * y
# division
result = x / y
# modulo (resto de la division)
result = x % y
# exponente
result = x ** y
# division floor o division suelo
result = x // y
```

Ilustración 4. Ejemplo de operadores aritméticos en Python.

Operadores aritméticos

Los o**peradores aritméticos** permiten llevar a cabo operaciones matemáticas sobre datos numéricos. Algunos ejemplos extraídos de diferentes lenguajes de programación:

- Suma: +
- Resta: -
- Multiplicación: *
- Exponenciación: **
- División: /
- División suelo: //
- Resto: %

Operadores de comparación

Operadores de comparación, el resultado es verdadero o falso:

- Mayor que: >, >=
- Menor que: <, <=
- Igual que: =
- Igual que con comprobación de tipos: ===
- Distinto que: !=

Operadores lógicos

Permiten evaluar más de una condición a la vez:

- Conjunción: y (and)
- Disyunción: o (or)
- Negación: ! (not)

Operadores de asignación

Permiten asignar un valor a una variable además de poder realizar una operación aritmética a la vez y asignar el resultado:

- =
- +=
- -=
- *=
- /=

Estructuras de control

El flujo de ejecución normal de los programas es **secuencial,** de arriba abajo, sentencia a sentencia.

Ese flujo de ejecución puede ser alterado utilizando **estructuras de control** que permitan crear bifurcaciones en el código además de poder repetir determinados fragmentos.

Para ello existen las estructuras de control **condicional** y de **repetición.**

Tema 2. Características de los lenguajes de programación para IA

El principal objetivo de las estructuras de control es poder crear **programas más flexibles** y con **mayor rango de posibles acciones** al permitir evaluar distintas condiciones además de iterar sobre estructuras de datos.

Estructuras de control condicional

Las **estructuras de control condicional** permiten evaluar una o más condiciones a fin de ejecutar un bloque de código y u otro en función de si se cumplen o no.

Las estructuras condicionales más comunes entre los diferentes lenguajes de programación son:

- **If:** equivale a Si. Ejecuta el código que envuelve si se cumple la condición o condiciones que evalúa.
- **Else:** equivale a Entonces. Ejecuta el código que envuelve si no se cumple la condición o condiciones que evalúa la estructura if anterior.
- **Else-If:** equivale a Entonces Si. Ejecuta el código que envuelve si no se cumple la condición o condiciones que evalúa la estructura if anterior y además se cumple las que envuelve con su propio if.
- **Switch:** equivale a un interruptor. Funciona de manera similar a un if combinado con else-if. Evalúa los diferentes casos posibles para el valor de una variable y proporciona un código para cada uno de ellos.

```
if check1:
    print("")
elif check2:
    print("")
elif check3:
    print("")
elif check4:
    print("")
else:
    print("else")
```

Ilustración 5. Ejemplo de estructuras de control en Python.

Dentro de las estructuras condicionales se encuentran a menudo operadores lógicos and y or que permiten evaluar más de una condición a la vez.

```
if check1 or check2:
    print("Verdadero")
    print("Verdadero2")

if check1 or (check2 and check3):
    print("Verdadero")
    print("Verdadero2")
```

Ilustración 6. Ejemplo de estructuras de control en Python utilizando operadores lógicos.

Estructuras de control repetitivo

Las **estructuras de control repetitivo** permiten que un mismo bloque de código pueda ser ejecutado más de una vez durante la ejecución de un programa.

Se utilizan principalmente cuando se quiere realizar una operación varias veces, por ejemplo, recorrer los números en una estructura de datos para calcular el sumatorio.

Las estructuras de control repetitivo pueden ser **determinadas** o **indeterminadas.**

Una estructura de control repetitivo es determinada cuando tiene acotado un rango máximo de repeticiones, es el caso de los **bucles for.**

```
text = "tesla"

for char in text:
    print(char)
```

Ilustración 7. Ejemplo de estructura de repetición for en Python.

Por otro lado, las estructuras de control repetitivo son **indeterminadas** cuando la repetición depende de si se cumplen ciertas condiciones. Las condiciones se verán alteradas dentro del bucle de forma que cuando se logre el objetivo las condiciones terminen la ejecución de este. Es el ejemplo de los **bucles while.**

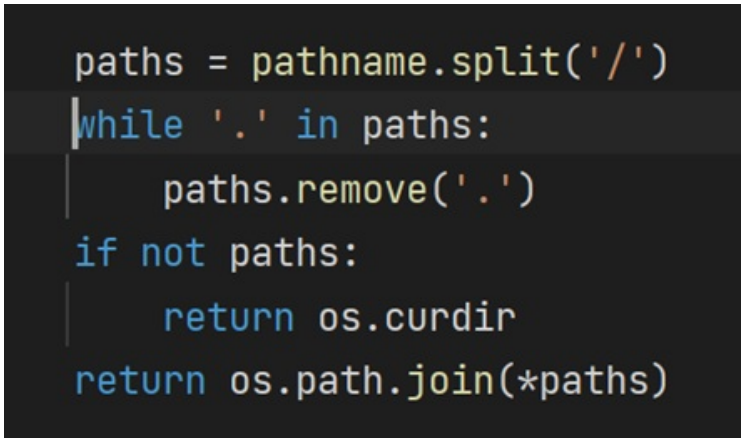


Ilustración 8. Ejemplo de estructura de repetición while en Python.

Funciones o procedimientos

Una **función** o procedimiento es un bloque de código que puede ser reutilizado en diferentes partes de un programa sin necesidad de duplicar el código que contiene.

Las funciones tienen dos partes bien diferenciadas:

- La signatura: es el conjunto de palabras que identifican la función: su visibilidad, su tipo de retorno, su identificador, sus parámetros.
- El cuerpo: de la función es el código que ejecuta, puede ser una o múltiples instrucciones.

Las variables y estructuras de datos que se crean dentro de una función existen durante la ejecución de esta, pero no fuera.

Ilustración 9. Ejemplo de funciones en Python.

Tema 2. Características de los lenguajes de programación para IA

2.4. Paradigmas de programación

La complejidad es el principal factor por el cual el código de los programas se vuelve complejo y difícil de mantener en el tiempo. Cuanta menos complejidad tenga un programa, más fácil será de leer y también de mantener y evolucionar, por ende, menos costes implicará su desarrollo.

Uno de los principales enfoques para lidiar con la complejidad son los **paradigmas de la programación.**

Un paradigma de programación hace referencia a un estilo de programación. No se trata de un lenguaje específico, si no de la forma en la que se programa y estructura el código.

Existen dos principales familias de **paradigmas de programación:**

- Programación **imperativa**
- Programación **declarativa**

Programación imperativa

La **programación imperativa** es un paradigma a través del cual se especifican en el código todos los pasos que debe realizar el programa para lograr un objetivo. Para ello se utilizan variables, estructuras de control condicional y repetitivo.

La **programación imperativa** se encarga de describir el **cómo** se debe llevar a cabo un proceso paso por paso, definiendo el flujo de control como sentencias que cambian el estado del programa.

Tema 2. Características de los lenguajes de programación para IA

Dentro de la programación imperativa existen múltiples enfoques:

- Programación **estructurada**
- Programación **orientada a objetos**

Programación estructurada

La **programación estructurada** es un paradigma que busca mejorar la legibilidad y reducir la complejidad de un código por medio del uso de **sentencias de control.**

La programación estructurada emplea 3 estructuras para la creación de programas:

- Secuencia. La estructura secuencial es la que se da de forma natural en el lenguaje, porque las sentencias se ejecutan en el orden en el que aparecen en el programa, es decir, una detrás de la otra.

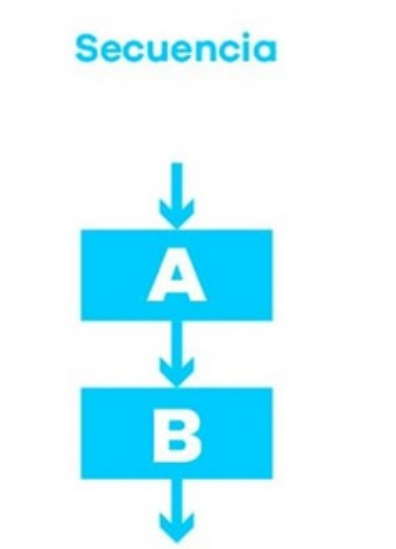


Ilustración 10. Ejemplo de estructura secuencial.

- **Selección o condicional.** La estructura condicional se basa en que una sentencia se ejecuta según el valor que se le atribuye a una variable booleana. Una variable booleana es aquella que tiene dos valores posibles. Por tanto, esta estructura se puede ejecutar de dos formas distintas, dependiendo del valor que tenga su variable.

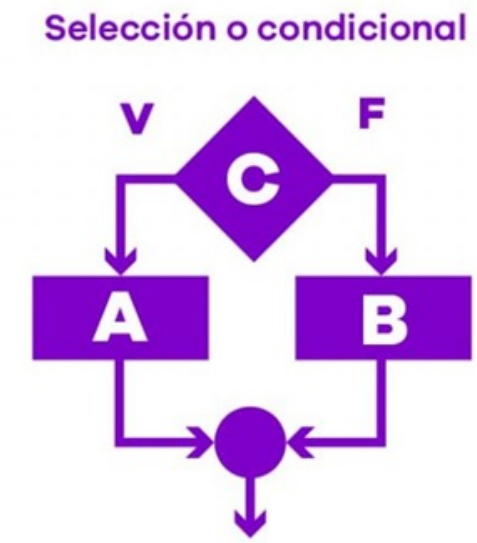


Ilustración 11. Ejemplo de estructura de control condicional.

- **Iteración (ciclo o bucle).** La estructura de repetición ejecuta una o un conjunto de sentencias siempre que una variable booleana sea verdadera. Para los bucles o iteraciones, los lenguajes de programación usan las estructuras while y for.

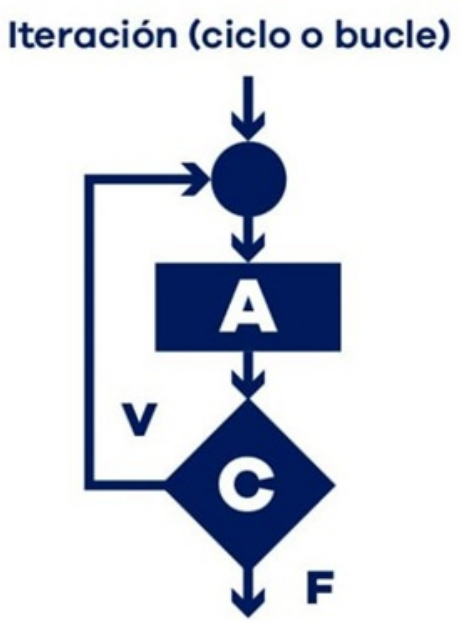


Ilustración 12. Ejemplo de estructura de control repetitivo o de iteración.

La **programación procedimental** es un tipo de programación estructurada y se basa en el uso de procedimientos o funciones que modifican el estado del programa.

Programación orientada a objetos

La **Programación Orientada a Objetos (POO)** es un paradigma que organiza los programas como objetos, elementos compuestos por variables (estado) y métodos (comportamiento).

Los elementos básicos de la programación orientada a objetos son las **clases** y los **objetos.** Las clases actúan como plantillas que especifican los atributos y métodos que tendrán los objetos.

Los **objetos** son las **instancias** particulares que se crean a partir de una clase. Ejemplo: a partir de la clase vehículo se pueden crear muchos objetos vehículo, uno para representar un Toyota, otro para representar un Ford, uno puede estar en marcha, otro estacionado, etc.

Otros **mecanismos** de la programación orientada a objetos son:

- Encapsulación
- Herencia
- Abstracción
- Polimorfismo
- Serialización
- Asociación
- Composición

La programación orientada a objetos se utiliza especialmente en el desarrollo de programas complejos y grandes **sistemas de software** de forma **escalable.**

Dentro de la programación orientada a objetos se emplea el uso de patrones de diseño con el fin de promover el uso de **buenas prácticas** y la creación de código de calidad.

Los **patrones de diseño** son soluciones a problemas comunes en software, no son código en sí, son formas de estructurar los elementos de la programación orientada a objetos (clases, objetos, abstracción).

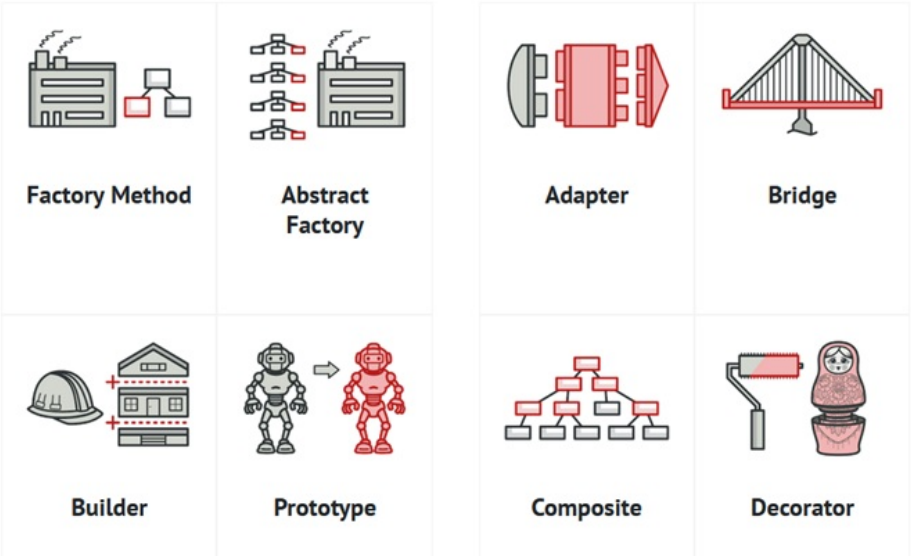


Ilustración 13. Ejemplo de estructuras de datos en Python. Fuente: https://refactoring.guru/design-patterns/catalog

Programación declarativa

De forma paralela a la programación imperativa se desarrolló el paradigma declarativo. La **programación declarativa** define la lógica del programa, pero no detalla el flujo de control.

En la **programación declarativa** el programador define qué es lo que espera del programa sin definir cómo debe ser implementado, es decir, sin programar los pasos necesarios para obtenerlo. Este paradigma se enfoca en qué es necesario obtener en lugar de especificar el cómo obtenerlo.

Dentro del paradigma de la programación declarativa destacan algunos subparadigmas como:

- Programación funcional
- Programación lógica

Tema 2. Características de los lenguajes de programación para IA

Programación funcional

Entre los elementos sintácticos de los lenguajes de programación se encuentran las funciones. Una función es un bloque de sentencias para ejecutar una determinada tarea de forma reutilizable a lo largo de un código.

La **programación funcional** es un paradigma que consta de llamadas a funciones concatenadas en las que cada parte del programa se interpreta como una función.

De esta forma, dentro de la programación funcional las funciones pueden adoptar distintas estructuras.

Un ejemplo de uso de la programación funcional es la parametrización del comportamiento, la cual consiste en poder utilizar como parámetro de una función otra función. Siendo posible también utilizar una función como resultado o retorno de otra función.

La **programación funcional** se emplea a menudo en conjunto a la programación orientada a objetos, permitiendo la creación de aplicaciones flexibles y escalables.

Características de la programación funcional:

- Inmutabilidad de los datos
- Transparencia referencial
- Modularidad
- Mantenibilidad
- Funciones puras
- Inmutabilidad de los datos

Se debe poder crear nuevas variables y estructuras de datos **sin tener que modificar** las ya existentes.

Al evitar los datos existentes y crear copias se evitan posibles **efectos secundarios.**

Favorece el uso del **paralelismo,** evitando que una ejecución altere el estado de otra ejecución que tiene lugar simultáneamente.

- Transparencia referencial

Una función es **referencialmente transparente** si siempre devuelve el mismo resultado cuando es invocada con los mismos valores de entrada en sus argumentos.

Siempre devuelve el mismo resultado para la misma entrada, independientemente de que se ejecute múltiples veces y en diferentes lugares de un programa.

- Modularidad

L a **modularidad** es una técnica para diseñar software en pequeños módulos o fragmentos independientes, cada uno con todo lo necesario para ser ejecutado y proporcionar una **funcionalidad.**

La modularidad incrementa la **productividad** debido a que los módulos pueden ser programados más rápido y con capacidad para ser reutilizados en diferentes programas.

Alternativamente, los programas modulares pueden ser **probados** y **depurados** de manera más sencilla y rápida.

- Mantenibilidad

La **mantenibilidad** de un software hace referencia a la facilidad con la que se puede evolucionar y corregir fallos en el mismo.

Tema 2. Características de los lenguajes de programación para IA

Cuando un software es muy complejo, introducir nuevos cambios y mejorar las funcionalidades existentes se vuelve cada vez más difícil y costoso.

La **deuda técnica** es una técnica que mide la entropía del software, es decir, el coste acumulado por utilizar código que no cumple con los estándares de calidad y que no mantiene una coherencia con otras partes del programa.

En el largo plazo, una deuda técnica alta puede hacer que los costes de mantenimiento de un software se disparen.

Ilustración 14. Ejemplo de resultados de calidad en la plataforma SonarCloud. Fuente: https://sonarcloud.io/

- **Funciones puras**

Las **funciones puras** son también conocidas como funciones libres de efectos secundarios o funciones sin estado.

Estas funciones cumplen con las siguientes condiciones:

- Retorna el mismo resultado para los mismos argumentos de entrada siempre.
- No tiene efectos secundarios: no modifica datos en base de datos, no altera el estado de variables externas, etc.

Un ejemplo de función pura puede ser una función suma, que recibe dos parámetros y devuelve la suma de ambos sin modificar ninguno de ellos en el proceso.

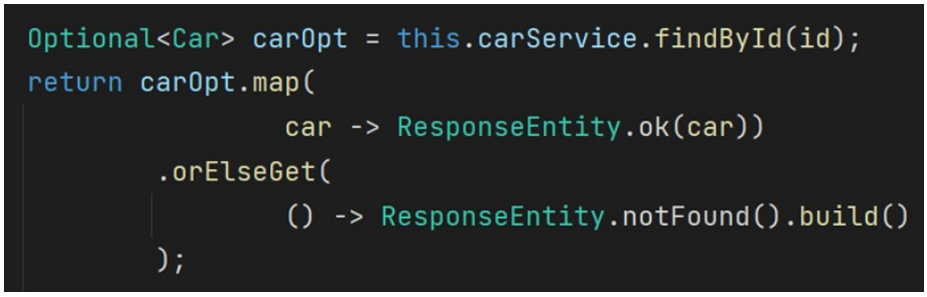
Por el contrario, una función impura es aquella que tiene efectos secundarios, por ejemplo, modificando los argumentos que recibe en la entrada. Las funciones utilizadas en el paradigma imperativo son generalmente impuras ya que modifican el estado del programa o toman valores de consola diferentes en cada lectura.

- **Funciones de orden superior**

Las **funciones de orden superior** toman una o más funciones como argumentos en la entrada e incluso pueden retornar una función.

Las funciones de orden superior dan lugar a la **parametrización del comportamiento,** ya que los funciones representan un comportamiento y puede ser pasado como parámetro a su vez a otras funciones.

Ejemplo de programación funcional en Java



```
Optional<Car> carOpt = this.carService.findById(id);
return carOpt.map(
                car -> ResponseEntity.ok(car))
        .orElseGet(
                () -> ResponseEntity.notFound().build()
        );
```

Ilustración 15. Ejemplo de controlador REST que devuelve un objeto Coche o una respuesta HTTP Not Found en función de si existe en la base de datos o no.

Programación lógica

El paradigma de la **programación lógica** utiliza un enfoque declarativo basado en la lógica formal.

Se compone de **hechos** y **reglas** en un determinado ámbito de dominio o problema en lugar de utilizar instrucciones. Por ejemplo:

A es verdadero si B, C y D son verdaderos.

Dentro de los lenguajes del paradigma de la programación lógica destaca **Prolog,** abreviatura de *programming in logic.*

Tema 2. Características de los lenguajes de programación para IA

2.5. Características de los lenguajes de programación para IA

Los **lenguajes de programación** son la herramienta principal para explorar y construir programas informáticos en el área de la inteligencia artificial.

Los **programas de inteligencia artificial (IA)** llevan a cabo tareas como el aprendizaje, razonamiento, reconocimiento de información simbólica en contextos, etc.

En las primeras etapas, los lenguajes de programación se utilizaban principalmente para realizar **cálculos numéricos.**

Con el fin de construir programas para simular la inteligencia humana es necesario definir algoritmos capaces de procesar **información de cualquier tipo.** Para ello, es necesario trabajar con símbolos o información que no sean únicamente datos numéricos.

La **computación simbólica** consiste en el desarrollo de algoritmos que permiten manipular expresiones y símbolos matemáticos. Estos símbolos son combinados con reglas o heurísticas para llevar a cabo resolución de problemas y procesar información. Algunos campos en los que se utiliza la computación simbólica son:

- Álgebra computacional
- Resolución de teoremas
- Sistemas de planificación
- Diagnosis
- Reescribir sistemas

- Representación del conocimiento
- Sistemas expertos

La **simulación de inteligencia** se divide en diferentes ramas o subproblemas de forma que sea posible crear sistemas inteligentes. Las más importantes son:

- Razonamiento y resolución de problemas
- Representación del conocimiento
- Planificación
- Aprendizaje automático (Machine Learning)
- Procesamiento del lenguaje natural (PLN)
- Percepción: visión por computador, reconocimiento facial, de voz, de objetos, etc.

Entre las **características** necesarias para que un lenguaje de programación sea idóneo para inteligencia artificial destacan:

- Paradigma declarativo
- Manejo de estructuras de datos
- Búsqueda
- Optimización
- Lógica
- Métodos probabilísticos
- Manejo de incertidumbre
- Clasificación

- Estadística
- Recursión y Back tracking
- Aprovechamiento de los recursos computacionales (CPU, GPU, etc.)

Por lo general, los problemas de inteligencia artificial tienden a ser muy **específicos del dominio** en cuestión sobre el que operan (sector bancario, sector sanitario, marketing, etc.). Por ello, las estrategias heurísticas o de resolución de problemas deben desarrollarse empíricamente a través de enfoques de **generación y prueba** o ensayo y error.

L a **programación de IA** difiere notablemente de los enfoques de ingeniería de software estándar donde la programación generalmente comienza a partir de una especificación formal detallada. En la programación de IA, la propia **implementación** es en realidad parte del proceso de **especificación** del problema.

Debido a la **naturaleza difusa** de muchos de los problemas de IA, la programación de IA se beneficia notablemente cuando se utilizan lenguajes de programación que liberan al programador de la mayoría de los aspectos técnicos (asignación y liberación de espacio en memoria, construcción de tipos de datos, etc.).

Según esto, el **paradigma declarativo** beneficia a la programación de IA al permitir crear programas sin necesidad de especificar todos los pasos para ejecutar el flujo de control, simplemente poder especificar qué es lo que se necesita.

Dentro de la programación declarativa se encuentra la **programación funcional** y la **programación lógica**. Ambos paradigmas se basan en formalismos matemáticos, y los lenguajes de programación más conocidos en este ámbito son **Lisp, Prolog y Haskell**.

Tema 2. Características de los lenguajes de programación para IA

En la actualidad muchos lenguajes soportan **múltiples paradigmas de programación**, pudiendo ser utilizados tanto para el desarrollo de software Enterprise como para el desarrollo de software de IA.

Otro factor clave es la existencia de **frameworks** y **librerías** en constante evolución para el desarrollo de software de IA como es el caso de los lenguajes **Python** y **Java**.

El proceso de resolución de problemas a través de la **ciencia de datos** involucra una serie de pasos:

- Acceso y extracción de los datos
  - Sensores
  - Archivos
  - Bases de datos
  - APIs
- Análisis de los datos
  - Exploración de los datos
  - Limpieza y tratamiento de datos
- Preprocesamiento de los datos
- Modelado
- Visualización e interpretación de resultados