1. Generaciones de lenguajes de programación

Desde que se crearon los primeros ordenadores, con el paso del tiempo se va incrementando el poder y el nivel de abstracción de los diferentes lenguajes de programación. Sin embargo, las nuevas generaciones de lenguajes no consiguen sustituir a los de generaciones anteriores totalmente:

a) Lenguajes de primera generación (1GL): Código máquina.
b) Lenguajes de segunda generación (2GL): Lenguajes simbólicos (con símbolos) de bajo nivel: lenguajes ensamblador y macroensamblador.
c) Lenguajes de tercera generación (3GL): Lenguajes más comprensibles para el programador. Lenguajes de alto nivel con características imperativas. Ejemplos: C, PASCAL, Fortran, ALGOL, BASIC …
d) Lenguajes de cuarta generación (4GL): Se continúa la tendencia hacia una mayor abstracción y poder en las sentencias de programación. Diseñados con un propósito concreto, para abordar un tipo concreto de problemas. Ejemplos: NATURAL, Matemática, SQL, QBE, Matlab, ...
e) Lenguajes de quinta generación (5GL): Con estos lenguajes el programador tan sólo debería establecer qué problema ha de resolverse, las condiciones o restricciones que debe reunir la solución, y la máquina se encargaría de resolverlo. Se usan en inteligencia artificial. Ejemplos: Prolog, LISP, IDL de ICAD, OPS5, Mercury, KL-ONE, ...

Inicialmente, todos los lenguajes de más alto nivel que el ensamblador fueron denominados de "tercera generación". Después se introdujo el término "cuarta generación" para tratar de diferenciar los (entonces) nuevos lenguajes declarativos (como Prolog y lenguajes de propósito específico) que parecían expresarse de forma más cercana al usuario (al lenguaje natural) de los lenguajes originales de alto nivel, de características imperativas.

En los lenguajes 4G se utilizan herramientas que aumentan la productividad, aunque es discutible el considerarlas lenguajes. Algunas de estas herramientas tienen que ver con:

- el acceso a bases de datos.
- la generación de formularios, presentación de datos, creación de pantallas.
- la generación de informes.
- la creación de interfaces gráficas.
- la generación de código automático.
- la posibilidad de programación en manera visual.

No hay consenso en las características de los lenguajes 4G y 5G. La clasificación en generaciones fue abandonándose por otras más precisas, como su paradigma de programación: orientados a objetos, declarativos, funcionales, ...

REFERENCIAS GENERACIONES E HISTORIA DE LENGUAJES DE PROGRAMACIÓN:

https://en.wikipedia.org/wiki/Programming_language_generations
https://en.wikipedia.org/wiki/Fourth-generation_programming_language#Examples
http://www.alegsa.com.ar/Dic/lenguaje%20de%20cuarta%20generacion.php
https://es.wikipedia.org/wiki/Historia_de_los_lenguajes_de_programaci%C3%B3n
https://es.wikipedia.org/wiki/Teor%C3%ADa_de_lenguajes_de_programaci%C3%B3n

2. Paradigmas de programación

Todo programa es desarrollado para satisfacer determinada necesidad o necesidades de los que vayan a ser sus usuarios, pero la forma en que se programan, el estilo de programación, o la filosofía de cómputo subyacente, pueden variar mucho.

Paradigma imperativo

En este paradigma los programas están compuestos por un conjunto de instrucciones que el ordenador tiene que ejecutar. El programador da órdenes a la computadora. De ahí el nombre: imperativo. Los datos se almacenan en variables en memoria principal.

Se basa en el funcionamiento nativo del procesador, que, una tras una, ejecuta las instrucciones del programa en código máquina.

Los primeros lenguajes de alto nivel son característicos de este paradigma: **C**, **Fortran**, **Cobol**, **BASIC**, **Pascal**, ...

Paradigma declarativo

No sigue la estructura nativa de la máquina, sino que el programa consiste en una serie de declaraciones, de especificaciones, que el ordenador debe cumplir (y cumplirá, sin más ayuda del programador).

Algunos lenguajes representativos de este paradigma: **SQL**, **HTML**, **CSS**...

Tanto el paradigma funcional, como el lógico, son en realidad subparadigmas del declarativo, pero tienen entidad propia.

Paradigma Funcional

Un programa en el paradigma funcional consiste en la declaración de una serie de funciones (al estilo matemático, con uno o varios dominios de entrada y uno de salida). Las funciones pueden estar definidas en función (valga la redundancia) de otras funciones. La ejecución de un programa en este paradigma es, simplemente, la llamada a una de las funciones del programa.

Algún lenguaje representativo de este paradigma: **LISP**, **Scheme** (dialecto de LISP), **Erlang**, **F#**, **Haskell**, ...

Lógico

Los programas en el paradigma lógico están compuestos por una serie de hechos y reglas lógicas. La ejecución en este tipo de programas se realiza mediante una pregunta lógica, que el motor lógico del entorno de ejecución responderá en función de su “conocimiento” (los hechos y reglas declarados por el programador).

Lenguaje representativo de este paradigma: **Prolog**. Otros: **Mercury**, **Gödel**, **ALF** (Another Logical Framework), ...

Conceptos relacionados con la programación imperativa

Programación estructurada

Digamos que la programación imperativa puede ser estructurada o no estructurada. El uso de la sentencia goto (y otras de salto) pueden dar una estructura lógica a los programas muy intrincada. Está demostrado que esto no hace falta, independiente del objetivo del programa. Que todo el programa pueda ser descompuesto en bloques de código con una sola entrada y sola salida en su flujo de ejecución es la base de la programación estructurada. De este modo aparecen tres grandes tipos de estructuras para componer los programas:

- La **estructura secuencial**: cada instrucción, o cada fragmento de código, se ejecuta uno tras otro.
- Las **estructuras alternativas**: sólo uno entre varios fragmentos de código es el que se ejecuta.
- Las **estructuras repetitivas**: un fragmento de código se repite 0 o más veces.

Con solo estas estructuras puede hacerse cualquier programa que se necesite.

Paradigma procedimental

Decir que un lenguaje de programación permite programar mediante el paradigma procedimental es prácticamente lo mismo que decir que es de tipo imperativo. Los procedimientos son un conjunto de instrucciones que pueden ejecutarse como una unidad. Al referirse a los procedimientos se está resaltando la posibilidad de descomponer el código en fragmentos, entrando en la idea de la programación modular (un programa está formado por varios módulos, deseablemente, con una **alta cohesión** en su contenido y una **baja dependencia** entre ellos.)

Paradigma de la programación orientada a objetos

En la programación orientada a objetos, los programas son vistos como un conjunto de objetos que, para obtener el resultado deseado, interactúan entre sí por medio de mensajes. Los objetos con las mismas características y comportamiento se agrupan en clases, creando nuevos tipos de datos, y las clases se clasifican de nuevo por supertipos y subtipos. En este contexto es donde aparecen los mecanismos de la herencia. Unos objetos pueden estar asociados con otros formando estructuras mediante las que colaborar.

Las clases son los módulos en este paradigma. Agrupan datos y operaciones. Cada clase lleva asociadas una responsabilidades que debe cumplir, como buen módulo de software que son.

El polimorfismo (la posibilidad de que los elementos del programa tomen diferentes formas) es otra de las características asociadas a este paradigma.

El paradigma de la programación orientada a objetos favorece mucho la generación de nuevas librerías, nuevos tipos de datos y la reutilización de código. Al trabajar con este paradigma la impresión puede ser la de estar trabajando con objetos (modelos) similares a los del mundo real (los del programa pertenecen, por supuesto, al mundo virtual) y entre los que se pueden dar los mismos tipos de relaciones (conceptuales y físicas) que en la realidad.

Algunos lenguajes representativos de este paradigma: **Smalltalk**, **Java**, **C++**, **C#**, ...

Lenguajes de programación multiparadigma

Cada lenguaje de programación suele está centrado en satisfacer, al menos, algunas de las cualidades de un único paradigma de programación. Puede darse el caso, sin embargo, de que permita programar desde diferentes perspectivas (que cubra suficientemente las características de varios paradigmas a la vez). A estos lenguajes de programación se les denomina multiparadigma. Es el programador, entonces, el que elige con qué estilo programar.

Hay características comunes entre los distintos paradigmas. Por ejemplo, es habitual cierta retrocompatibilidad entre los lenguajes que permiten la programación orientada a objetos, basándose en la programación imperativa; las clases de la programación orientada a objetos son, en realidad, declaraciones; en un lenguaje imperativo como C se pueden declarar funciones, luego puede aportar posibilidades de programación funcional; ...

Como un ejemplo de lenguaje multiparadigma, está **Python**, que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional; o el lenguaje **Scala**, que integra programación orientada a objetos y programación funcional; etc...

Ejemplos de programación funcional y lógica

Programación funcional

Primer ejemplo:

Tenemos la función suma definida externamente así:

Nombre de la función: suma
Dominios de entrada: dos números enteros.
Dominio de salida: un número entero.

Como parte del programa se define la función como:

suma(a: entero, b: entero): entero ::= a+b

Para invocar a la función se escribe

suma(2,3)

y se obtiene el resultado:

5

La función de multiplicación del programa podría quedar definida utilizando la función suma:

producto(a, b)::= si b es 0, 0
sino suma(producto(a, b-1), a)

La potencia o exponenciación podría utilizar la multiplicación:

potencia(a,b)::= si b es 0, 1
sino producto(potencia(a, b-1), a)

y así, sucesivamente.

Segundo ejemplo, en lenguaje LISP, de una función recursiva:

Se recuerda la definición matemática de factorial. Factorial(x) es 1 si x es 0: caso base. Factorial(x) es x por factorial(x-1), si x>0: caso de llamada recursiva.

La definición de la función factorial en lenguaje LISP sería así:

```
************************************************************
  ;Función factorial hecha con recursividad no final
  (defun factorial (n)
     (if (= 0 n)
       1                              ; caso base
       (* n (factorial (- n 1)))     ; caso recursivo
     )
  )
************************************************************
```

La invocación siguiente:

```
  (factorial 4) ;
```

Nos devolvería:

```
  24
```

4*3*2*1, da 24.

Programación lógica

Ejemplo 1, escrito en PROLOG:

Se definen algunos hechos: Juan es padre de Pepe, Pepe de Ramiro. Los nombres deben ir en minúscula.

```
papa_de(juan,pepe).
papa_de(pepe,ramiro).
```

Se definen la regla o reglas necesarias: ¿Cuándo se es abuelo de alguien? Los caracteres "`:-`" se leen "si". La coma "y". Las variables deben ir en mayúscula.

```
abuelo_de(X,Y) :-
    papa_de(X,Z),
    papa_de(Z,Y).
```

Por último, se puede consultar al sistema lógico, empezando con `:-`.

¿Es Juan padre de Pepe?

```
:-papa_de(juan,pepe)
Success
```

Cierto

¿Es Juan padre de Ramiro?

```
:-papa_de(juan,ramiro)
Failure
```

Falso

¿Quién es hijo de Juan?

```
:-papa_de(juan,X)
X = pepe
Success
```

Juan sí tiene un hijo: Pepe.

¿Qué preguntas se podrían hacer con la regla abuelo? ¿Cuál sería la regla para ser hermanos?

Ejemplo 2, también en PROLOG, con respuestas múltiples:

Se enuncian unos predicados: rojo es un color, azul es un color, verde es un color.

```
color(rojo).
color(azul).
color(verde).
```

Se pregunta por ¿Qué colores hay?

```
:-color(C).
C = rojo
Success
```

(Se piden más respuestas)

```
C = azul
Success
```

(Se piden más respuestas)

```
C = verde
Success
```

(Se piden más respuestas)

```
Failure
```

Ya no hay más colores.