INTRODUCCIÓN A LA PROGRAMACIÓN

1.1. Introducción a la programación.

1.1.1.Problemas, algoritmos y programas.

Un **algoritmo** es una secuencia de acciones que se realizan para resolver un problema. La tarea de programar en realidad se trata de resolver problemas. La programación se encarga de resolver problemas a través de algoritmos. El algoritmo debe especificar las acciones que se deben ejecutar y en qué orden deben ejecutarse. Un algoritmo es independiente del lenguaje de programación. Es decir, una vez tengamos un algoritmo que resuelve un problema, podemos aplicarlo sobre cualquier lenguaje de programación.

Un programa es un algoritmo escrito con las sentencias específicas de un lenguaje de programación concreto. Un algoritmo recibe información de entrada, la procesa, y genera información de salida.

No todos los problemas pueden resolverse utilizando un ordenador. Solo se podrán resolver aquellos problemas que tengan una solución mecánica. Es decir, aquellos que se resuelvan por medio de una secuencia de instrucciones determinada. Son los denominados problemas computacionales o algorítmicos. Como por ejemplo, los relacionados con el cálculo numérico, el tratamiento de palabras, o la representación gráfica. Un ejemplo de problema computacional podría ser obtener el área de un cuadrado a partir de su lado. En cambio, un problema no computacional podría ser averiguar el color favorito de una persona a partir de su nombre.

1.1.2.Programas y programar.

La tarea de la programación en aplicaciones reales de una cierta envergadura es bastante compleja. Según la complejidad del problema a resolver podemos hablar de:

- **Programación a escala reducida**: número reducido de líneas de programa, intervención de una sola persona. Por ejemplo, un programa para ordenar números.
- **Programación a gran escala**: muchas líneas de programa, equipo de programadores. Por ejemplo, el desarrollo de una sistema operativo.

Los programas tienen un ciclo de vida en el que se distinguen dos etapas: desarrollo y mantenimiento.

1.1.3.Lenguajes y modelos de programación.

A nivel máquina, un **programa** es una sucesión de palabras expresadas en código binario (secuencia de 0s y 1s), en posiciones consecutivas de memoria que representan instrucciones o datos. Es el **lenguaje máquina**. Es evidente que los programas escritos en lenguaje máquina resultan ilegibles.

Los **lenguajes ensambladores** hacen uso de mnemotécnicos e identificadores para instrucciones y datos.

Ambos son lenguajes tan próximos a la máquina que se denominan **lenguajes de bajo nivel**.

Frente a los lenguajes de bajo nivel, se encuentran los **lenguajes de alto nivel**:

- Disponen de operadores y estructuras más próximas al lenguaje humano, esto permite al programador construir instrucciones de forma más sencilla.
- Son más seguros que el código máquina y ayudan a no cometer errores evidentes.
- El código es transportable (independiente de la máquina).
- El código es más legible.

1.1.4.Compiladores e intérpretes.

Hay dos formas de traducir un programa escrito en un lenguaje de alto nivel al lenguaje máquina: interpretar y compilar.

En la **interpretación** se traduce a lenguaje máquina cada instrucción de manera individual en tiempo de ejecución (de una en una).

En la **compilación** se traducen de manera global todas las instrucciones por medio de un programa (compilador). En los lenguajes compilados se debe hacer siempre una compilación previa para poder ejecutar el programa resultante.

1.2.Representación de algoritmos.

Para representar un algoritmo se suele utilizar algún método que permite independizarlo del lenguaje de programación utilizado. Esto nos permitirá codificar más tarde a través de cualquier lenguaje de programación. Existen diversas técnicas, podemos destacar:

- **Diagramas de flujo** (ordinogramas) Utilizan símbolos gráficos estándar y escriben los pasos del algoritmo dentro de los símbolos. A través de las líneas de flujo se indican la secuencia en que se deben ejecutar.
- **Lenguajes de descripción de algoritmos** (Pseudocódigo).

Pseudocódigo.

Es un **lenguaje** para la **descripción de algoritmos** que procura mantener las propiedades de los diagramas de flujo: **sencillez**, **claridad**, **normalización** y **flexibilidad**. Su notación se basa en el **lenguaje natural**. La estructura general del pseudocódigo está dividida en dos partes:

- **Nombre y entorno del programa**: tipos, nombre de las variables y constantes que se utilizan.
- **Desarrollo ordenado y estructurado del algoritmo**: esta segunda parte se corresponde con el diagrama de flujo.

1.2.1.Elementos de un algoritmo.

Un algoritmo está formado por:

1. **Datos**: son los valores que describen los objetos con los que opera un ordenador, como pueden ser números o texto. Los datos se almacenan en variables y constantes. Los datos pueden ser de tipo: entero, real, texto y boolean. Representan cantidades numéricas, cadenas de texto y valores (verdadero o falso). Los datos se guardan en variables, que son espacios de memoria reservados para almacenar un tipo de datos.
2. **Expresiones**: es una combinación de constantes, variables, operadores, paréntesis y nombres de funciones. Las expresiones pueden ser aritméticas o lógicas.

1. Las **expresiones aritméticas** tendrán resultado numérico, y tienen operandos (con constantes y variables numéricas) y operadores (aritméticos).

Operador	Operación	Operador	Operación
+	Suma o signo	+=	Suma y asignación
-	Resta o signo	-=	Resta y asignación
*	Multiplicación	*=	Multiplicación y asignación
/	División	/=	División y asignación
%	Módulo	%=	Módulo y asignación
++	Incremento en 1	--	Decremento en 1

El resultado de las **expresiones lógicas** será de tipo booleano (verdadero o falso). Estas operaciones están formadas por:

- **Operandos**: contantes, variables y expresiones lógicas.
- **Operadores relacionales**:

Operador	Operación
==	Igual
!=	Distinto
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Las **expresiones** pueden agruparse a través de operadores lógicos. Este tipo de operadores se utilizan para hacer varias comprobaciones de forma conjunta, dichas expresiones pueden utilizar:

- **Operandos**: constantes, variables y expresiones lógicas.
- **Operadores relacionales**.
- **Operadores lógicos** (not, and, or, xor).

Operador	Operación	Significado
!	NOT	Negación lógica
&&	AND	Conjunción lógica (Y lógico)
||	OR	Disyunción (O lógico)
^	XOR	Disyunción Exclusiva (XOR)

x	y	x && y	x || y	x ^ y	!x
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

2. **Instrucciones**: acciones básicas que implementan los algoritmos, como leer o escribir en variables.

1.3.Instrucciones en pseudocódigo.

Las instrucciones se clasifican en:

- **Declarativas**: anuncian el **uso de variables y constantes**, se debe indicar el **nombre** de la variable y el **tipo de dato** que contendrá.
- **Primitivas**: las ejecuta el **procesador** de manera **inmediata**; no dependen de ningún otro objeto o condición. Las operaciones de **entrada leen valores** y los **asignan** a un **identificador** en memoria, las de **salida envían datos** a un dispositivo desde la memoria, y las de **asignación obtienen** el resultado de una expresión y la **guarda** en un identificador en memoria.
- **Control**: evalúan **expresiones lógicas** para controlar la **ejecución** de otras **instrucciones** o **alterar el orden de ejecución** normal (**ejecución secuencial**). Hay **alternativas** (ejecutan una instrucción o conjunto de instrucciones después de evaluar una condición) **y repetitivas** (el bloque de código puede ejecutarse más de una vez).

Bibliografía.

Material de Edu Torregrosa Llácer (aulaenlanube.com)

Links útiles

- Sitio oficial de Python - Documentación
- Tutorial oficial de Python
- W3Schools - Python
- El libro de Python