1. Fundamentos de programación

1. La programación

Si nos fijamos en la segunda acepción del diccionario sobre el significado de *programar*, vemos que se refiere a una técnica de ordenación de las acciones que hay que realizar para llevar adelante cualquier tipo de proyecto.

programar Conjugar

1. tr. Formar programas, previa declaración de lo que se piensa hacer y anuncio de las partes de que se ha de componer un acto o espectáculo o una serie de ellos.

2. tr. Idear y ordenar las acciones necesarias para realizar un proyecto. U. t. c. prnl.

3. tr. Preparar ciertas máquinas o aparatos para que empiecen a funcionar en el momento y en la forma deseados.

4. tr. Elaborar programas para su empleo en computadoras. U. t. c. intr.

Real Academia Española © Todos los derechos reservados

La programación, genéricamente, tiene uso en prácticamente todas las disciplinas. En educación, por ejemplo, es preciso programar un curso antes de llevarlo a cabo. Es decir, hay que decidir qué contenidos se van a impartir y en qué orden, qué medios de evaluación se van a utilizar, qué criterios de calificación se van a utilizar, etc.

Por ello, independientemente de que hablemos de programación informática o no, nos vamos a quedar con la idea de que la programación es un método para organizar ideas.

2. Algoritmos

Se define por algoritmo al conjunto ordenado y limitado de instrucciones que resuelven un problema o tipo de problema.

Ejemplo. Algoritmo de la suma

- Tomar la cifras que se encuentren más a la derecha de todos los sumandos
- Mientras (haya cifras):
  - Sumar las cifras que se han tomado
  - ¿El resultado es menor de 10?
    - Sí:
      - Escribir el resultado bajo las cifras sumadas
    - No:
      - Escribir la unidad del resultado bajo las cifras sumadas
      - Escribir la decena del resultado sobre las cifras que se encuentran a la izquierda de las sumadas
  - Tomar las cifras que se encuentren a la izquierda de las sumadas
- Fin mientras

La algoritmia forma parte del álgebra matemático y estamos acostumbrados a utilizarla con bastante asiduidad.

3. Pseudocódigo

Una de las formas más habituales de escribir algoritmos es la utilización de pseudocódigos.

El pseudocódigo es una forma de escribir las instrucciones empleando un lenguaje cercano a nosotros por lo que resulta muy fácil de entender. Pese a ello, deben seguirse ciertas reglas para que el pseudocódigo pueda ser entendido sin subjetividades por cualquier persona.

Estas reglas las iremos viendo según las necesitemos durante el curso. De momento, solo vamos a ver la siguientes:

- Debe iniciarse con la instrucción: **Inicio** seguida del nombre del algoritmo. El nombre del algoritmo lo elige el programador o le vendrá dado por las instrucciones que reciba por parte de quien le asigne el trabajo
- Debe finalizarse con la instrucción: **Fin** seguida del nombre del algoritmo
- Las instrucciones se escriben de forma ordenada

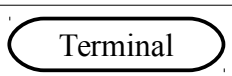
Ejemplo. Pseudocódigo del algoritmo de la suma

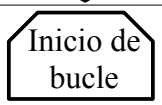
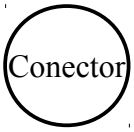
- Inicio suma
- Tomar la cifras que se encuentren más a la derecha de todos los sumandos
- Mientras (haya cifras):
  - Sumar las cifras que se han tomado
  - ¿El resultado es menor de 10?
    - Sí:
      - Escribir el resultado bajo las cifras sumadas
    - No:
      - Escribir la unidad del resultado bajo las cifras sumadas
      - Escribir la decena del resultado sobre las cifras que se encuentran a la izquierda de las sumadas
  - Tomar las cifras que se encuentren a la izquierda de las sumadas
- Fin mientras
- Fin suma

4. Diagramas de flujo

Una técnica gráfica de representar programas es la utilización de diagramas de flujo o flujogramas «*flowchart*». Estos diagramas se componen de una serie de símbolos, con indicaciones en su interior, interconectados mediante flechas que indican el camino del programa.

Algunos de los símbolos son:

	Indica el inicio o final del algoritmo
	Simboliza las acciones a realizar por parte del equipo programado.

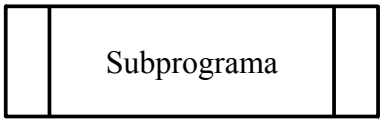
	Representa la entrada de datos desde el exterior al equipo programado o la salida
	Se utiliza cuando es necesario tomar decisiones que impliquen la realización de diferentes instrucciones
	Representan el principio y el final de un bucle. Dentro de un bucle se encuentran las instrucciones que deben repetirse mientras se dé alguna condición
	
	Los conectores se utilizan para unir partes del programa que no quepan en la página en la que se está realizando la representación del algoritmo. Dentro del conector se suele indicar una letra mayúscula que debe coincidir en las dos partes que están unidas.

5. Subprogramas

Una de las acciones más útiles que hay que realizar a la hora de diseñar cualquier tipo de programa es la de subdividir el problema en problemas más sencillos, de forma que pueda resolverse como una sucesión de pequeños problemas.

Para ello, se realiza el pseudocódigo de cada uno de los subprogramas y, desde el programa principal, se va llamando a cada uno de los subprogramas. Un subprograma puede contener, a su vez, llamadas a otros subprogramas.

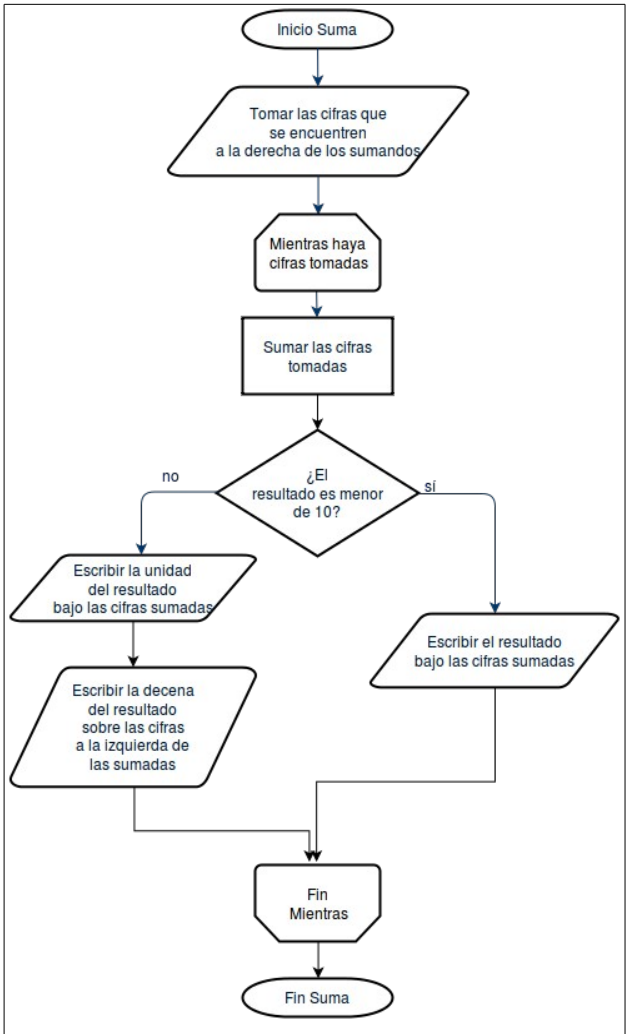
Las llamadas a subprogramas se simbolizan en flujogramas mediante el símbolo siguiente:



6. Variables

Es bastante habitual que necesitemos que el programa guarde cierta información durante su ejecución. Para ello se utilizan las variables.

Existen diferentes tipos de variables en función



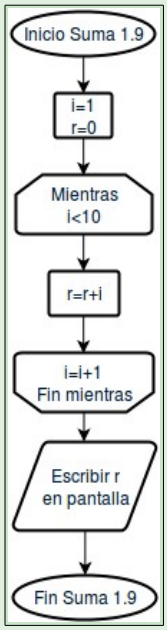
Ejemplo. Flujograma del algoritmo de la suma

del tipo de información que se necesite conservar (variables de texto, de números enteros, de números binarios, de números en coma flotante, vectores, etc).

La asignación del valor a la variable se suele realizar en pseudocódigo utilizando el símbolo igual y es posible operar con ellas.

Ejemplo. Algoritmo que suma los números del 1 al 9

- Inicio suma1.9
- i=1, r=0
- Mientras (i<10):
  - r=r+i
  - i=i+1
- Fin mientras
- Muestra en pantalla el valor de r
- Fin suma1.9



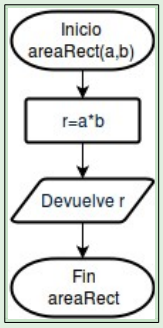
Los subprogramas pueden recibir datos en variables y enviar resultados para que sean utilizados en otras partes de un programa.

En pseudocódigo, se va a utilizar un paréntesis en el que se indica las variables que recibe el subprograma, así como el orden en el que las recibe.

Para indicar el envío del resultado vamos a utilizar la instrucción **devuelve**.

Ejemplo: subprograma que calcula el área de un rectángulo

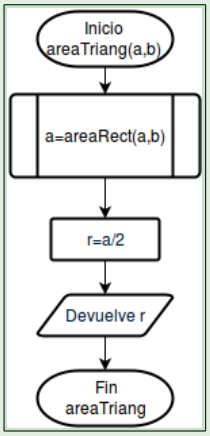
- Inicio areaRect (a, b)
- r=a*b
- Devuelve r
- Fin areaRect



No es necesario que el programa y el subprograma utilicen el mismo nombre de variables, ya que estos funcionan independientemente.

Ejemplo: subprograma que calcula el área de un triángulo a partir del subprograma anterior

- Inicio areaTriang (a,b)
- a=areaRect(a,b)
- r=a/2
- Devuelve a
- Fin areaTriang



Vamos a utilizar los programas anteriores para intentar entender cómo se utilizan las variables. Imaginemos que llamamos desde un programa con la instrucción "area=areaTriang(3,4)".

El subprograma areaTriang va a meter el número 3 dentro de su variable **a** y el número 4 dentro de **b**. Después, va a llamar a areaRect utilizando estas dos variables ("a=areaRect(a,b)").

El subprograma areaRect recibe lo que areaTriang tenía en **a** de areaTriang y lo mete en **a** de areaRect. Del mismo modo, recibe lo que areaTriang tenía en su **b** y lo mete en la suya (daos cuenta de que aunque las variables tengan el mismo nombre, al estar en distinto subprograma, se guardan en distintos sitios de la memoria del ordenador).

Acto seguido, areaRect introduce en su variable **r** el resultado de multiplicar sus **a** y **b** (r=a*b=3*4=12) y devuelve este valor a quien le hizo la llamada (en este caso areaTriang).

areaTriang introduce en su variable **a** el resultado obtenido al llamar a areaRect (machacando el valor que tenía anteriormente, por lo que a pasa de valer 3 a valer 12), e inmediatamente, divide el valor que había en **a** entre dos guardándolo en **r** (r=a/2=12/2=6). Por último, devuelve el resultado a quien le hizo la llamada.

7. Programación estructurada

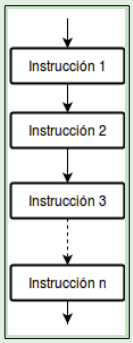
Desde finales de los años 1970, surge una forma de programar que daba lugar a programas fiables y eficientes, y que se ha mantenido hasta la actualidad.

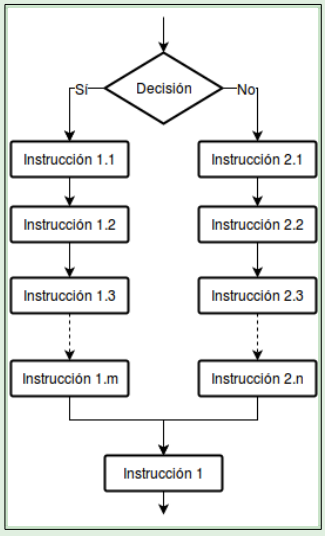
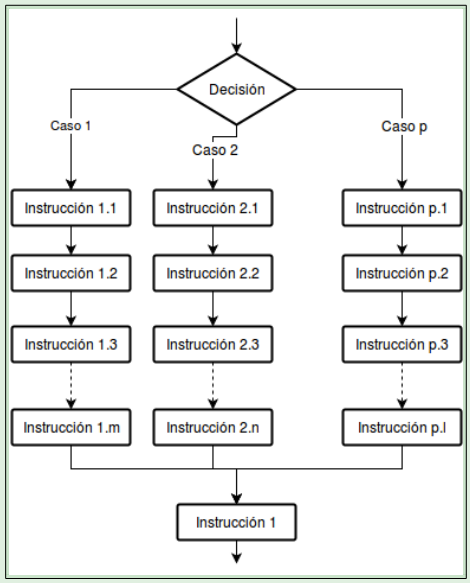
La programación estructurada tiene una serie de reglas sencillas y divide la programación en una serie de bloques o estructuras de programación que se repiten y combinan para formar cualquier tipo de programa.

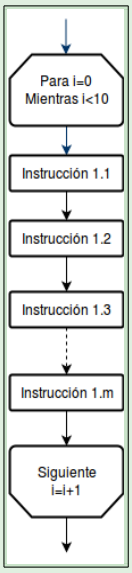
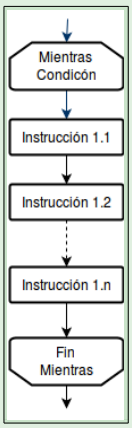
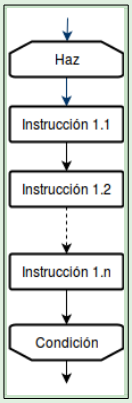
De esta forma, para programar solo vamos a utilizar tres tipos de estructuras de instrucciones:

- **Secuencia**: instrucciones sucesivas (una detrás de otra)
- **Selección**: se establecen instrucciones que se ejecutan cuando se cumple (o no) una determinada condición
- **Iteración**: conjunto de instrucciones que se repiten mientras se cumpla una determinada condición

7.1. Tipos de estructuras de control

Estructura secuencial		
Secuencia	Conjunto de instrucciones que se ejecutan secuencialmente (una detrás de otra)	

Estructuras de selección		
Selección simple	Hay dos conjuntos de instrucciones a seguir en función del valor de una solución a una decisión de tipo sí/no	
Selección múltiple	Existen diferentes conjuntos de instrucciones a seguir en función de la respuesta a una decisión que puede tener múltiples soluciones	

Estructuras iterativas		
Iteración tipo For-Next	Se da un número de vueltas concretas a un conjunto de instrucciones utilizando un contador y una condición para volver a repetir. En cada vuelta se actualiza el valor del contador. En el ejemplo • se le otorga valor cero a la variable i (contador) • las instrucciones se repiten mientras i valga menos de 10 • en cada vuelta i aumenta su valor sumándole 1 al valor anterior	
Iteración tipo While	Se da un número de vueltas a un conjunto de instrucciones mientras se cumpla una determinada condición	
teración tipo Do-While	Se da una primera vuelta a un conjunto de instrucciones y se repite mientras se cumpla una determinada condición	

Los programas se componen de una combinación de las estructuras anteriores. Las estructuras se pueden encontrar una detrás de otra o una dentro de otra, formando todo tipo de combinaciones.

8. Programación orientada a objetos (POO)

Desde mediados de los años 1980 comienza una nueva forma de programar que se mantiene hasta hoy en día. En esta, se considera **objeto** a lo que se quiere programar y tiene un enfoque bastante más cercano a nuestro entendimiento que los métodos anteriores.

La programación orientada a objetos tiene, además, su propia norma gráfica para representar los programas en su conjunto, llamada UML (Unified Modeling Language).

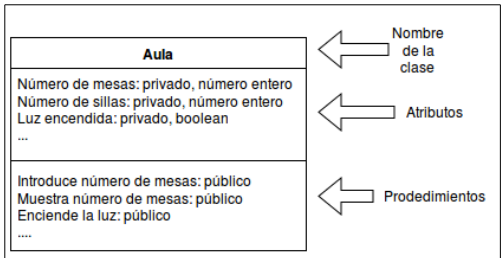
8.1. Conceptos de la POO

Clase

Una clase contiene las propiedades y el comportamiento de un tipo de objeto concreto. Las clases tienen los **atributos** (variables) que debe necesita el objeto y los **métodos** que dicho objeto puede realizar (en POO, hablar de procedimiento es equivalente a hablar de subprograma).

Los programas se componen de un conjunto de clases.

Tanto las variables como los procedimientos pueden ser públicos (accesibles desde cualquier parte del programa) o privados (solo accesibles desde un procedimiento de la misma clase). Existen otro tipo de procedimientos, pero no los vamos a utilizar, de momento, en este curso.



Representación UML de una clase

Un ejemplo de clase podría ser la clase “aula”. Esta clase podría tener una serie de atributos como, por ejemplo: número de mesas, número de sillas, número de ventanas, si tiene o no proyector, si está la luz encendida o apagada, si está la calefacción encendida o apagada, etc. Como métodos, podríamos tener “enciende luz”, “apaga luz”, “nombre del alumno x”, etc.

Objeto

Los objetos se crean a partir de las clases y se puede trabajar con ellos. Un ejemplo, sería “aula de 4ºA”. El objeto “aula de 4ºA” pertenecería a la clase “aula”. Todo lo programado sobre la clase “aula” lo tiene el objeto “aula de 4ºA” y puede utilizarse para crear cualquier otro objeto (en POO, se denomina **instanciar un objeto de la clase x**). De esta forma, si la clase está bien programada, podríamos instanciar otras aulas, como “aula de 2ºB”, “aula de 3ºA”, cada una con su propio valor dentro de los atributos (“aula de 2ºB” podría tener “número de mesas” con valor “30” y “aula de 4ºA” con valor “28”, ya que cada objeto puede tener sus propios valores).

Un objeto puede ser un atributo de otra clase. Por ejemplo, la clase “instituto”, podría tener un vector de objetos “aulas” (un vector es un conjunto).

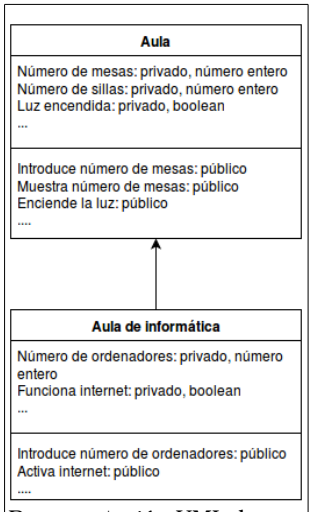
Herencia

Una de las características más útiles de la POO es el concepto de **herencia**. Una clase puede heredar de otra, de forma que tiene los atributos y métodos de la clase de la que parte sin necesidad de ser programados.

Por ejemplo, la clase “aula de informática” puede heredar de la clase “aula”. De esta forma, no es necesario volver a programar los métodos y atributos que ya están en la clase “aula”, sino solo los específicos de la clase “aula de informática” (“número de ordenadores”, por ejemplo).

Así, cuando se cree un objeto de la clase “aula de informática” (por ejemplo, “aula de informática 2”) este objeto incluirá todo lo que se ha programado dentro de “aula” y lo que se ha programado dentro de “aula de informática”.

En UML se representa la herencia como una flecha desde la clase que hereda hasta la clase de la que parte (super clase o padre).



Representación UML de una herencia

Atributos

Es el conjunto de variables que contiene una clase. En UML se representa como una lista bajo el nombre de la clase.

Métodos

Conjunto de subprogramas que contiene una clase. En UML se representa como una lista bajo los atributos.

Evento

Se denomina evento a una acción que puede ejercitarse sobre una clase y que implica que se ejecute un método de la misma. Las eventos más típicos pueden ser que se pulse una tecla concreta del teclado, que se haga “click” en el ratón, que un temporizador llegue a un determinado valor (por ejemplo, que haya pasado un tiempo desde que se instanció la clase), etc.

8.2. Ventajas de la POO

La principal ventaja de la POO es que se programa por bloques independientes. Un programa es un conjunto de clases que pueden ser analizadas y reprogramadas independientemente al resto del programa. Por ello, es muy útil cuando se tienen equipos de personas programando.

Por otro lado, al poder utilizar la herencia, se facilita enormemente la labor de programación si se ha hecho un buen análisis previo del programa.