

Algoritmos y Programación

Víctor Giordano
Comunidad IT

Temario

- Objetivo de la programación
- Acciones, interlocutor y Algoritmos
- Redacción algorítmica
- Dato e Información
- Pseudo Código
 - Estructura de un algoritmo
 - Tipos de Acciones
 - Constantes, Variables, Asignación y Expresiones.
 - Condiciones Lógicas
- Resolución de un primer problema

Objetivo de la Programación

- ¿Cual es?
 - La resolución de problemas de tipo computacionales
- ¿Cuando un problema es computacional y cuando no?
 - Cuando se cuenta con una computadora que se pueda hacer el trabajo para resolver el problema.
 - Depende el contexto

Resolución de problema Real

- Se nos pincha el neumático de un rueda
 - Tenemos que cambiar la rueda
 - ¿Como hacemos?
 - Seguimos una serie de pasos o **acciones**.
 - Tengo mis manos inhabilitadas, como le comunico a otro de hacer esta tarea.
 - Que nivel de detalle requiere este otro para efectuar las acciones que le voy detallando

Resolución de problema Real (II)

- Para algunas personas bastará con:
 1. Cambiar la rueda.

Resolución de problema Real (III)

- Otras en cambio, entenderán esto:
 1. Levantar el auto
 2. Quitar la rueda pinchada
 3. Traer la rueda de auxilio
 4. Colocar la rueda de auxilio
 5. Bajar el auto
 6. Guardar la rueda pinchada

Resolución de problema Real (IV)

- Pero si estoy tratando con alguien que realmente no tiene experiencia (solo tiene voluntad). Tal vez tenga que descomponer las acciones en otras acciones más sencillas de realizar.
- Para la acción 1
 1. Traer el Gato
 2. Colocar el gato en la ranura
 3. Subir el gato hasta que la rueda se separe del suelo
- Para la acción 4
 1. Poner la rueda en posición
 2. Ajustar las 4 tuercas

Definiciones

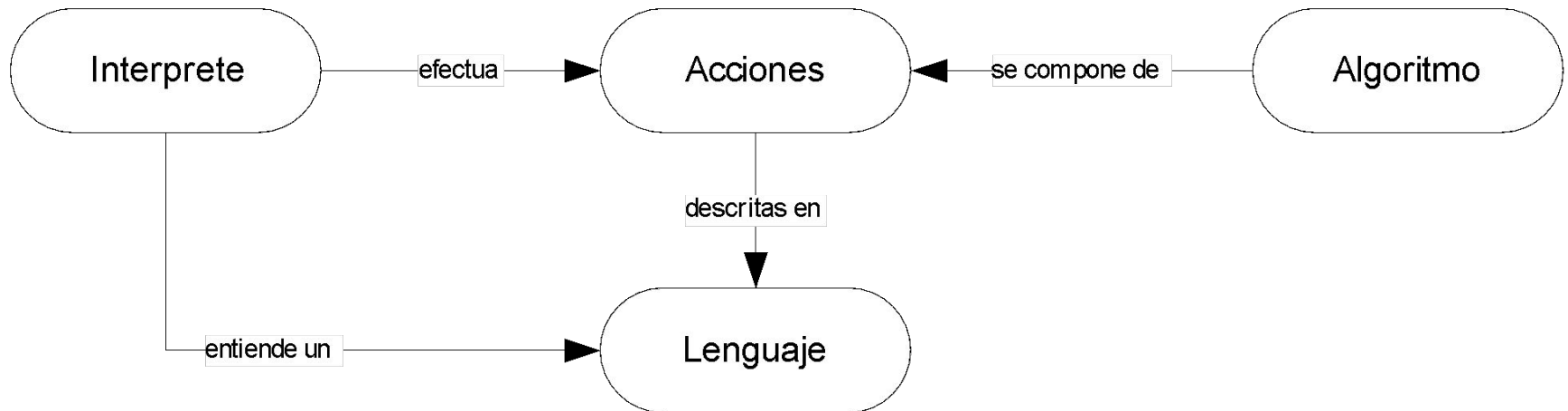
- **Interlocutor** es toda entidad (persona u objeto) que puede comprender un método enunciado (conjunto de acciones) y llevarlo a cabo (Ejecutarlo)
- **Acción** es una descripción de algo que se tiene que hacer y modifica el **ambiente**.
- No todo **acción** la puede efectuar cualquier **interlocutor**.
 - Acciones Primitivas
 - Acciones No Primitivas

Definiciones (II)

- **Algoritmo:** Es una secuencia ordenada de acciones orientada a resolver un problema o determinada cuestión.
- **Lenguaje:** Es el que hablamos con el interlocutor para describir los algoritmos.

Recapitulando definiciones

- La ***redacción algorítmica*** consiste en narrar un conjunto de **acciones** en un **lenguaje** para que las realice un **interlocutor**.



Redacción algorítmica

- Hagamos un algoritmo para cambiar un foco quemado

Redacción algorítmica

- Hagamos un algoritmo para cambiar un foco quemado
 1. Colocar escalera
 2. Sacar foco quemado
 3. Poner nuevo foco
 4. Guardar escalera

Redacción algorítmica (II)

- Ahora con más detalle.
 1. Colocar escalera
 2. Subir escalera
 3. Desenroscar foco quemado
 4. Bajar escalera
 5. Agarrar foco nuevo
 6. Subir escalera
 7. Enroscar foco nuevo
 8. Bajar escalera
 9. Guardar escalera

Redacción algorítmica (III)

- Reflexionemos en cómo pensamos...
 1. Colocar escalera
 2. Sacar foco quemado
 1. Subir escalera
 2. Desenroscar foco quemado
 3. Bajar escalera
 3. Poner nuevo foco
 1. Agarrar foco nuevo
 2. Subir escalera
 3. Enroscar foco nuevo
 4. Bajar escalera
 4. Guardar escalera

Redacción algorítmica (IV)

- Primero lo pensamos en los términos más generales posibles.
 - Con mucho nivel de abstracción
- Luego pasamos a descomponer cada paso “grande” en una serie de pasos “chiquitos”
- Cada paso grande se conoce con el término de un **módulo**
 - Cada uno cumple una **función bien definida**
 - Cada uno de ellos, a veces, **se puede efectuar independientemente del otro**
 - Si ya tuviera el foco sacado, entonces no debería efectuar instrucciones para sacarlo nuevamente.

Redacción algorítmica (V)

- Lo anterior es parte del pensamiento modular que se conoce como filosofía de programación **top-down**
 - Es en realidad todo lo que tenemos a mano para resolver un problema cuando no lo conocemos con detalles
 - No es conveniente empezar con los detalles, porque nos perdemos!!!
 - Empezamos de lo más general porque nos es más fácil.
 - Y vamos de lo más **general a lo más específico**
- Parece fácil pero es **difícil**
 - Requiere práctica (Conocimiento en el dominio de la solución)
 - Creatividad
 - Haber tenido un buen día 😊

Redacción algorítmica (VI)

- ¿Para pensar?
 - **En el caso anterior**
 - ¿Que paso con la corriente?
 - Si saco o meto un foco y esta todo encendido probablemente sea el último algoritmo que ejecute en mi vida.
 - Sabiendo esto:
 - » Ó Asumo como **supuestos**
 - » Ó explícito las instrucciones para desconectar energía y conectar energía.
- Los **supuestos** son hechos que no están en el enunciado pero sí afectan a la resolución del problema.
 - Ayudan a demarcar claramente que es lo que estamos haciendo y **que no**.

Ejercitación

Práctica 1


Del Problema real a su resolución por computadora

- Problemas de tipo Computacionales
 - Interlocutor: Traductor a lenguaje máquina
 - Lenguaje: **Pseudo Código.**
 - Usado para meter los pies en la programación.
 - Énfasis en la redacción de algoritmos usando elementos de una computadora.
 - Acciones
 - Instrucciones + Datos

Pregunta al paso

- ¿Qué es un Dato?
- ¿Qué es información?

Observemos Diferencias

- 43440436
-  43440436
- El teléfono de Juan es 43330456

Entendamos Diferencias



43440436

Dato

Información

El teléfono de Juan es 43440436

Dato

Información

Hagamos otra lectura

- Juan tiene **20** años
- Roberto tiene **21** años
 - Hemos tomado como un dato la **edad**
- **Juan** tiene 20 años
- **Roberto** tiene 20 años
 - Tomamos como un dato al **sujeto**
- En una oración puede existir muchos **datos**, siendo la interpretación de los mismos lo que nos da información.

Dato e Información

- Un dato es una representación simbólica de una característica o propiedad de una entidad
 - No tiene sentido por sí solo.
 - Si lo procesamos o interpretamos entonces construimos información
 - Asociamos datos con algo o alguien.
(Contextualizar)

Dato e Información

- Un dato en la computadora se termina representando con un **valor**.
 - O sea que, en esencia, un **dato** es un valor acerca de algo.
 - Puede ser un número o una palabra o algo más grande, lo que sea necesario para poder describir el objeto

Como se almacena un dato

- Los seres humanos almacenamos datos
 - En las neuronas
- Las computadoras también almacenan datos
 - En la memoria
 - Pero no almacenan datos como lo hacemos nosotros
 - Tiene que haber un **proceso** mediante el cual **se transforme** ese dato manejable por los humanos a un dato manejable por la computadora.

Como se almacena un dato (II)

- Entonces la cantidad de libros en una mesa, el color de un auto, ¿Qué son dentro de la computadora?.
 - Un serie de bits.
 - Pues solo maneja dígitos binarios.
- Es decir que los datos se almacenan bajo **representación binaria** que pueda ser manejada por la computadora.
 - En esencia, los datos en una **computadora son valores que se guardan como una serie de bits.**

Que es un bit

- **Binary Digit**
- Elemento bivalente del computador que permite almacenar un 0 o un 1, es la unidad más chica de datos.

Sí.. por sí sola no dice mucho, a lo sumo dice si será verdadero o falso una determinada propiedad de una entidad
- Se visualiza como un voltaje eléctrico o impulso de corriente, positivo (1) o negativo (0) , que se encuentra en los circuitos de la memoria

Como se almacena un dato (III)

- Representar datos, aún más simples, como los dígitos decimales, una letra, un nombre o un color, requiere de una transformación desde el mundo real a alguna forma de **representación binaria** que pueda ser manejada por la computadora.
 - Números como bits
 - Letra como bits
 - Palabras como bits
- Datos más complejos como una imagen, una canción, un video o la trayectoria de un avión también son **representados en forma binaria**.
 - Imagen como bits
 - Video como bits
- Todo se representa como bits para guardarse en una computadora.

Como se almacena un dato (III)

- Representar datos, aún más simples, como los dígitos decimales, una letra, un nombre o un color, requiere de una transformación desde el mundo real a alguna forma de **representación binaria** que pueda ser manejada por la computadora.
 - Números como bits
 - Letra como bits
 - Palabras como bits

Como se almacena un dato (IV)

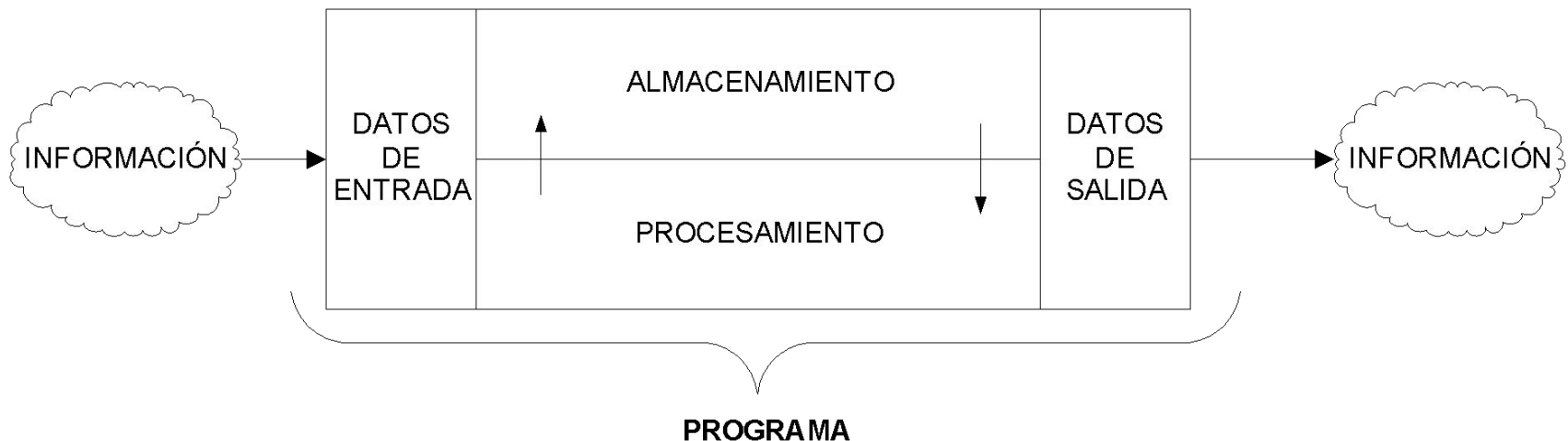
- Datos más complejos como una imagen, una canción, un video o la trayectoria de un avión también son **representados en forma binaria**.
 - Imagen como bits
 - Video como bits
- **Todo** se representa como **bits** para guardarse en una computadora.

Algoritmo = Instrucciones + Datos

- **Dato:** Un valor (acerca de algo).
- **Instrucción:** Es una operación que puede efectuar la computadora.
- **Las instrucciones manipulan datos.**
 - Las instrucciones se ejecutan en el orden en el cual se escriben, una tras de otra, es decir, en secuencia.
 - El orden en que se ejecutan se denomina comúnmente **flujo de ejecución**.

Definición Elemental de programa

- Un programa contiene un conjunto de **algoritmos** para **procesar** y **almacenar** volúmenes de **datos**.



Estructura de un algoritmo

Un algoritmo tiene dos grandes bloques:

`algoritmo` <nombre>

Declaración de datos

Acciones

`fin_algoritmo`

En lenguajes de alto nivel, como pseudocódigo, una **acción** puede involucrar muchas **instrucciones**.

Declaración de datos

- Un algoritmo computacional trabajará los datos de alguna forma.
- ¿En dónde ponemos estos datos?
- Estos datos se guardan en elementos que según, su criterio de mutabilidad, se denominan:
 - **Constante**: No valor es inmutable.
 - **Variable**: Su valor puede variar.

Variables y Constantes

- Elementos de datos identificables que tiene tres propiedades:
 - **Tienen un nombre** que las identificable
 - **Tienen un tipo** que describe su uso
 - Luego veremos esto de los tipos.
 - **Tienen un contenido:** que es el valor o dato que almacenan.

Variables y Constantes (II)

- Se declaran al comienzo de un algoritmo

`algoritmo a`

`var Número : x`

`var Número : y`

`...`

Más declaraciones de variables

Acciones (Cada una en un renglón aparte)

`fin_algoritmo`

- Se puede declarar dos en una misma línea

`var Numero : x, y;`

Grupos de acciones

- En la computadora no tenemos a nuestra disposición cualquier tipo de acción como lo veníamos haciendo describiendo los algoritmos en lenguaje natural
- Se distinguen tres “grandes” grupos de acciones
 - Todos los problemas que queramos resolver debemos describirlos en términos de estas.
 - TRANQUILOS!!
 - El teorema de Bohm-Jacopini demuestra que todo algoritmo se puede describir con los tres grupos de acciones a continuación.

Grupos de acciones (II)

- Simple / Secuencia: Operación que no altera el flujo de ejecución, es hacer alguna operación.
- Selección / Condicional: Permite tomar decisiones, modificando el flujo de ejecución a partir de un condición.
- Iteración: Permiten repetir acciones bajo un cierta condición o determinada cantidad de veces, alterando el flujo de ejecución.

Simple / Secuencia

- **Suele escribirse una por renglón**
- **Varios tipos:**
 - **Asignación**
 - Operación para darle valor a una variable a partir de una expresión.
 - **Invocación a subalgoritmos**
 - Desde un algoritmo se invoca a otro algoritmo que se ejecuta como si fuera un acción.
 - Ejemplo son: leer(...), imprimir(...)

Asignaciones

- Operación para darle valor a una **variable** a partir de una **expresión**.

`<var> ← <expr>`

- Primero se **resuelve la expresión** a la derecha, y el **resultado** se le asigna a la **variable** de la izquierda.

• `x ← 2 * 2`

• `theBeatles ← 5 - 1`

Expresiones

- Son **representaciones de un cálculo o cuenta** necesaria para obtener un resultado que deseamos.
- Es una combinación de ***operandos*** conectados con ***operadores***, que se evalúan y producen un valor.

Operadores

- Son **símbolos especiales** que llevan a cabo **operaciones** específicas con uno, dos o tres **operandos**, y luego devolver un resultado.
- Se usan para construir **expresiones**.
 - Operadores aritméticos: **+**, **-**, **/** y *****
 - $2 + 5$
 - $3 * 4 + 1 / 2$
- OK, ¿y que son los **operandos**?

Operandos

- Es el término correcto para referirse a los **elementos de entrada** que toman los **operadores** para hacer su cálculo u operación.
- Pueden ser variables, constantes, literales ,o inclusive, otras **expresiones**.
 - Por ejemplo:
 - $(2 * 5 + 1) + (3 / 2)$: Dos expresiones como operandos de una suma (el operador).

Operandos (II)

- Woa, woa.. ¿Entonces los **operandos** pueden ser **expresiones**?
 - Así es!
 - El término **expresión** como *concepto* puede interpretarse como una **generalización** del **operando** que toma un **operador**.
- Me siento un poco mareado...
 - Es normal, pero tranquilos... vamos a “acomodar” un poco los conceptos clasificando a las **expresiones** según su complejidad

Tipos de expresiones

(según complejidad)

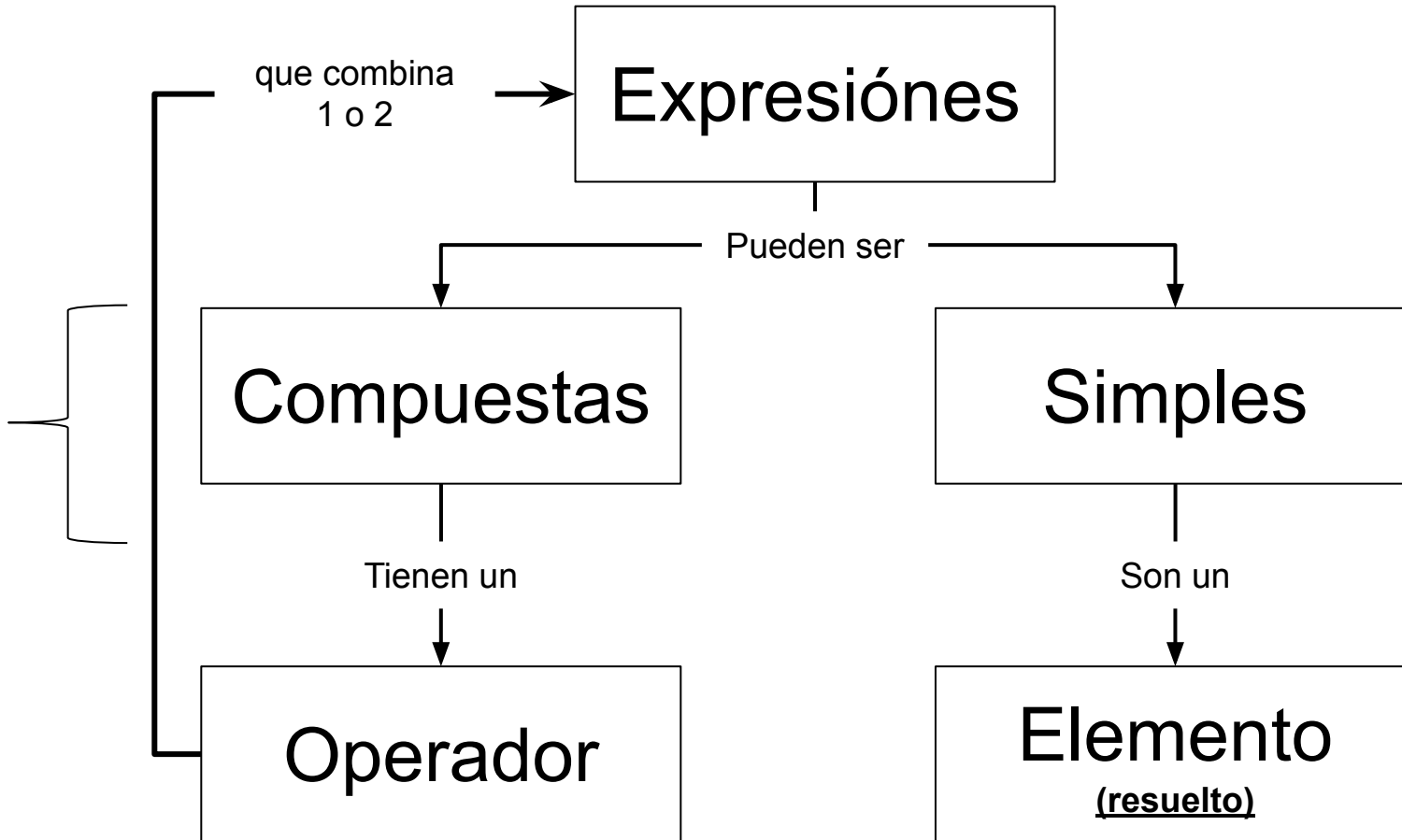
- Simple
 - Consiste en **un único elemento**
 - x
 - 5
 - π
 - No se necesita realizar alguna operación para obtener su resultado.

Tipos de expresiones (II)

(según complejidad)

- Compuesta
 - Consiste en un **conjunto de elementos** en donde se reconocen **operandos** vinculados con **operadores**
 - $x + 5$
 - $5 * PI / x$
 - $-(10)$
 - Se necesita efectuar al menos una operación (o más) para obtener su resultado.

Recapitulando Conceptos



Asignaciones (II)

- `<Nombre Var> ← <Expr>`

– Ejemplo:

- `X ← 5`
- `W ← X + 5 / PI`
- `Z ← 10 / 2 + 3 * (2 - 4)`
- `Z ← Z + 1`

– Ejemplo 2:

- `i ← 0`
- `i ← i + 2`
- `i ← (i * i) + 1`
- `i ← i - 5`

Lectura

- **leer** (X)
- Permite introducir un valor que se guardará en esa variable.

Solo funciona con variables

Pues su valor debe variar acorde al valor que se introduzca llegado el momento

Impresión

- **imprimir** ("Bienvenido al programa")
- Muestra ese texto en pantalla

Programa Saludo

```
algoritmo a
    var Cadena : nombre
    leer (nombre)
    imprimir ("Hola ")
    imprimir (nombre)
    imprimir (" , como estas?")
fin_algoritmo
```

Manos a la obra

- Hagamos un algoritmo que solicite el ingreso de dos números y calcule su producto.
- Hagamos un algoritmo que solicite el ingreso de dos números y calcule su promedio.

Selección Simple

- Es una acción que sirve para tomar una **decisión** y ejecutar **bajo una condición** ciertas acciones
 - Podemos elegir que acciones hacer y cuáles no

```
si <COND> entonces
    Acciones
{ sino
    Acciones }
fin_si
```

- La <COND> es una expresión que retorna un valor lógico.

Condiciones lógicas

- Son expresiones que pueden ser verdaderas o falsas a la hora de evaluarse.
 - Retornan valores lógicos
- Operadores de relación
 - “<”, “<=”, “>”, “>=”, “==” y “!=”
- Operadores Lógicos
 - “Y”, “O” y “NO”

Selección Simple (II)

- Ejemplos:

```
si x > 10 entonces
    x ← 0
sino
    x ← 20
fin_si
```

```
si x == 10 Y y != 20 entonces
    x ← 0
    y ← 0
sino
    x ← 20 + y
    y ← y + 20
fin_si
```


Selección Simple (III)

si $\langle \text{COND}_1 \rangle$ entonces

Acciones

sino si $\langle \text{COND}_2 \rangle$ entonces

Acciones

sino si $\langle \text{COND}_3 \rangle$ entonces

Acciones

...

sino

Acciones

fin_si

Manos a la obra (II)

- Hagamos un algoritmo que solicite el ingreso de dos números y muestre por consola cuál de ellos es mayor.
- Hagamos un algoritmo para determinar si tres palabras (ingresadas por el usuario) son iguales o no.

Ejercitación

Práctica 2

Iteración

- Sirven para repetir una acción o un conjunto de acciones una determinada cantidad de veces o bajo cierta condición.
 - La palabra iterar viene de **repetir**
- Una acción de iteración que usaremos se llama **mientras** y se escribe de esta forma:

```
mientras <COND> hacer
    Acciones
fin_mientras
```

- Donde <COND> es una expresión que retorna un valor de verdadero o falso.

Iteración (II)

- Se evalúa la condición
 - Si es verdadera
 - Ejecutan las acciones
 - Se vuelve al primer paso
 - Si es falsa
 - Fin
- Al conjunto de acciones que se ejecutan de forma reiterada se lo conoce como **Ciclo** o **Bucle**.
- Una **iteración** es equivalente a una **ejecución del ciclo o bucle**.

Ejemplo de utilidad

- Tengo que calcular el peso de todos los estudiantes del aula
- Sabemos que para un estudiante

```
var Numero : peso, pesoTotal;  
leer (peso)  
pesoTotal ← peso
```

Ejemplo de utilidad (II)

- Si somos dos

```
var Numero : peso, pesoTotal;
```

```
pesoTotal ← 0
```

```
leer (peso)
```

```
pesoTotal ← pesoTotal + peso
```

```
leer (peso)
```

```
pesoTotal ← pesoTotal + peso
```

Ejemplo de utilidad (III)

- Si somos 15

```
pesoTotal ← 0
```

```
leer (peso)
```

```
pesoTotal ← pesoTotal + peso
```

```
... (otras 13 veces)
```

```
leer (peso)
```

```
pesoTotal ← pesoTotal + peso
```

- Ufff... se vuelve largo y tedioso escribir esto tantas veces.

Ejemplo de utilidad (IV)

```
pesoTotal ← 0
mientras MeQuedenEstudiantesSinPesar hacer
    leer (peso)
    pesoTotal ← pesoTotal + peso
fin_mientras
```

- Vemos que realmente son útiles cuando debo reiterar la ejecución de un conjunto de acciones.
 - Algo que ocurre muy menudo

Ejemplo de utilidad (V)

- La pregunta pendiente es, cómo hago exactamente para repetir la ejecución de dichas acciones tantas veces como quiera.
 - Lo entendimos con palabras pero hay que escribirlo en pseudocódigo.

Ejemplo de utilidad (VI)

- El secreto está en definir la condición de iteración correctamente.
- Una forma muy conocida consiste en ir contando la cantidad de vueltas que vamos dando y cuando llegamos a la cantidad deseada terminamos
 - Tenemos que ir guardando la cantidad de vueltas que vamos dando, para ello usamos una variable.

```
cont ← 0
mientras cont < 5 hacer
    cont ← cont + 1
fin_mientras
```

- Dicha variable recibe el nombre de contador

Ejemplo de utilidad (VII)

Teníamos a

```
pesoTotal ← 0
mientras MeQuedenEstudiantesSinPesar hacer
    leer (peso)
    pesoTotal ← pesoTotal + peso
fin_mientras
```

Aplicando lo visto

```
pesoTotal ← 0
cont ← 0
mientras cont < 5 hacer
    leer (peso)
    pesoTotal ← pesoTotal + peso
    cont ← cont + 1
fin_mientras
```

Y solamente tenemos que variar cambiar el 5 por otro valor para contemplar más o menos estudiantes.

- De hecho podemos poner cualquier expresión que queramos
 - En particular podemos usar una variable cuyo valor se define ingresando teclado

Conclusiones

- Siempre que exista un procesamiento para un conjunto de elementos.
 - Tenemos que usar una iteración que vaya tomando por cada ciclo un elemento y lo **procese**.

```
mientras quedenElementos hacer
    procesarUnElemento
fin_mientras
```

- No siempre es sencillo, pero la idea, fue, es y será la misma.

Conclusiones (II)

- De forma más general
 - Siempre que tengamos que hacer un conjunto de pasos.
 - Es conveniente usar una iteración para ir haciendo en cada ciclo un paso

```
mientras quedenPasos hacer
    hacerUno
fin_mientras
```

Iteración (III)

Inflar un neumático es un buen ejemplo de lo que es una iteración aplicada a la vida real

```
mientras presiónGoma < 30 hacer  
    presiónGoma ← presiónGoma + 1  
fin_mientras
```

Enunciado

- Realizar un algoritmo (usando el **mientras**) que muestre 5 veces por pantalla un mensaje que diga “Hola Amigo”.

Acercándome a la solución...

```
algoritmo saludarA5Amigos
  mientras meFaltenAmigosPorSaludar hacer
    imprimir ("Hola amigo")
  fin_mientras
fin_algoritmo
```

Manos a la obra

Vamos con a analizar y resolver
un problema práctico un poquito
más complicado...

Enunciado

- Realizar un algoritmo que sume los primeros 10 números naturales.
 - Primero veamos cómo lo haríamos sin usar acciones de iteración
 - Y obtendremos algunas conclusiones

Resolución sin Iteración A

```
algoritmo sumar10Naturales
  var Número : suma
  suma ← 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
fin_algoritmo
```

- Es una solución válida!
 - Ahora resolvamos la suma nosotros
 - Sí ustedes y yo, sin calculadora!
 - Perdonen pero tengo que poner los paréntesis pues la suma está hecha para tomar de a **dos** elementos!
 - ((((((((((1 + 2) + 3) + 4) + 5) + 6) + 7) + 8) + 9) + 10)
 - UFFF....

Resolución sin Iteración A (II)

Permiso, agrego un cero al comienzo para que quede más claro... el cero a veces aclara la situación

- ((((((((((0 + 1) + 2) + 3) + 4) + 5) + 6) + 7) + 8) + 9) + 10)

(((((((((((← Acomodo un poco los paréntesis al costado mientras voy resolviendo por la izquierda

(0 + 1) + 2) + 3) + 4) + 5) + 6) + 7) + 8) + 9) + 10)	0 + 1 = 1)
(1 + 2) + 3) + 4) + 5) + 6) + 7) + 8) + 9) + 10)	1 + 2 = 3)
(3 + 3) + 4) + 5) + 6) + 7) + 8) + 9) + 10)	3 + 3 = 6)
(6 + 4) + 5) + 6) + 7) + 8) + 9) + 10)	6 + 4 = 10)
(10 + 5) + 6) + 7) + 8) + 9) + 10)	10 + 5 = 15)
(21 + 7) + 8) + 9) + 10)	21 + 7 = 28)
(28 + 8) + 9) + 10)	28 + 8 = 36)
(36 + 9) + 10)	36 + 9 = 45)
(45 + 10)	45 + 10 = 55)

55

- Notar cómo en un paso **siguiente** se suma lo del **anterior**

Resolución sin Iteración B

```
algoritmo sumar10Naturales
  var Numero : suma ← 0
  suma ← suma + 1    (0 + 1 = 1)
  suma ← suma + 2    (1 + 2 = 3)
  suma ← suma + 3    (3 + 3 = 6)
  suma ← suma + 4    (6 + 4 = 10)
  suma ← suma + 5    (10 + 5 = 15)
  suma ← suma + 6    (15 + 6 = 21)
  suma ← suma + 7    (21 + 7 = 28)
  suma ← suma + 8    (28 + 8 = 36)
  suma ← suma + 9    (36 + 9 = 45)
  suma ← suma + 10   (45 + 10 = 55) // En suma nos queda lo que queríamos
```

fin_algoritmo

- Lo que está en **amarillo** es lo que vale suma en el paso actual (durante la resolución de la expresión de la asignación).
- Lo que está en **azul** es lo que valdrá en el paso siguiente (como consecuencia de la asignación)
- Notar cómo en un paso **siguiente** se suma lo del **anterior**

Observaciones

- Los anteriores algoritmos son correctos, pero...
 - Si en vez de sumar los 10 primeros números, tengo que sumar los primeros 20, o los primeros 5.
 - No esta contemplado
 - Peor aún, qué ocurre si la cantidad de números a sumar se determina en tiempo de ejecución.
 - No esta contemplado

Conclusiones

- Es un algoritmo **correcto**, pero muy rígido y atado a una situación concreta.
 - ¿Esta mal?
 - No, no esta mal.

Enunciado 2

- Realizar un algoritmo que sume los primeros N números naturales.
 - Donde “ N ” es un número arbitrario de
- ¿Se puede?
 - Claro que sí.
 - Pero el esquema rígido anterior ya no sirve.
 - Esta es una **efectiva motivación** que hacía falta para emplear las acciones de iteración en la resolución

Resolución con Iteración

```
algoritmo sumar10NaturalesConIteración
  var Número : suma
  suma ← 0
  mientras quedenNumerosPorSumar hacer
    suma ← suma + proximoNumero
  fin_mientras
fin_algoritmo
```

- Pensemos que en cada iteración sumo un número de la serie.
 - La serie sería 1,2,3,4,5,6,7,8,9,10

Resolución con Iteración (II)

```
algoritmo sumar10NaturalesConIteración
  var Número : suma
  suma ← 0
  mientras quedenNumerosPorSumar hacer
    suma ← suma + proximoNumero
  fin_mientras
fin_algoritmo
```

- ¿Cuántos pasos tengo que hacer en total?
- ¿Y cómo voy obteniendo los números dentro de cada iteración?

¿Cuántas iteraciones?

- Si en cada paso sumó un número, entonces en N pasos sumaré N números.
 - Así que si tengo **20 números**, se tendrá que **20 repeticiones**.
 - Si tenemos **30 números** entonces **30 repeticiones**.
- Esto sugiere cuántas veces tengo que repetir las acciones.
 - Tantas veces como números se desean sumar!

¿Cómo iterar?

- Ya vimos como hacer esto. Pero repasamos
 - El secreto está en definir la condición de iteración correctamente.
 - Para esto tenemos ir contando la cantidad de vueltas que vamos dando y cuando llegamos a la cantidad deseada terminamos
 - Tenemos que ir guardando la cantidad de vueltas en algún lado
 - **Usamos una variable**

```
cont ← 0
mientras cont < 5 hacer
    cont ← cont + 1
fin_mientras
```

Dicha variable recibe el nombre de **contador** por ser, justamente, el papel que juega.

¿Cómo voy obteniendo los números?

- Los tenemos que ir generando de alguna manera tal que en cada ciclo obtenga un número que sea diferente del anterior, así sumo todos los números.
 - Un número diferente por ciclo.
- Podría pensar en primero sumar el 1, en el segundo ciclo sumar el 2, en el tercero sumar el 3, y así hasta llegar a N.

Generando los números

- Fijémonos en algo interesante de todo esto

```
cant ← 0
mientras cant < 5 hacer
    cant ← cant + 1
    imprimir (cant)
fin_mientras
```

Resolución 2 (II)

- Vamos a suponer que se quiere sumar los primeros 5.

```
algoritmo sumarPrimeros10Naturales
  var Número : suma, numero
  numero ← 0
  suma ← 0
  mientras numero < 5 hacer // cuando número sea 5, ya habré sumado
    todos
      numero ← numero + 1 // "genero" el siguiente número
      suma ← suma + numero // se lo sumo acumulándolo en SUMA
  fin_mientras
fin_algoritmo
```

- Dos técnicas de mucho uso.
 - **Acumulador**: Variables que acumulan valores a lo largo de los ciclos.
 - **Contador**: **Acumulador** de incremento unitario.

Resolución 2 (III)

```
algoritmo sumarPrimeros10Naturales
  var Número : suma, numero
  const Número : N = 5;
  numero ← 0
  suma ← 0
  mientras numero < N hacer
    numero ← numero + 1
    suma ← suma + numero
  fin_mientras
fin_algoritmo
```

- Si quiero sumar los primeros 15 números naturales basta con cambiar el valor de la constante N por 16.

Conclusiones

- Es muy importante saber manejar las acciones de iteración.
 - Son estas las que le permiten extender un algoritmo para varios casos.
 - Las técnicas de acumulador y contador presentadas para este caso se aplican en muchos otros casos.

Agregándole Salida

- Hasta ahora solo hicimos “cuentas” internas y no hay ningún resultado visible al usuario, solo hubo procesamientos internos
- Pensemos en que es de nuestro interés mostrar por pantalla el resultado de nuestra operación.
- Es por ello que se agregamos la salida para informar del resultado.

Agregándole Salida (II)

- Agregándole la funcionalidad para informar los resultados

```
algoritmo sumarPrimeros10Naturales
    var Número : suma, numero
    const Número : N = 10;
    numero ← 0
    suma ← 0
    mientras numero < N hacer
        numero ← numero + 1
        suma ← suma + numero
    fin_mientras
    imprimir ("La suma de los primeros "+ N+ " números naturales es " +suma)
fin_algoritmo
```

Agregándole Entrada

- Si le agregamos una constante se podría modificar la cantidad de valores a sumar modificando solo esa constante. Sin embargo, ello, significa una modificación del algoritmo cada vez.
- Como puedo hacer para sin modificar el programa hacer que se sume una cantidad arbitraria de números que se defina en momento de ejecución
- Una lectura de datos le proporcionará generalidad.

Agregándole Entrada (II)

```
algoritmo sumarPrimerosNaturales
  var Número : suma, numero, cantidad
  leer (cantidad) // me aseguro que funcione para cualquier número
  numero ← 0
  suma ← 0
  mientras numero < cantidad hacer
    numero ← numero + 1
    suma ← suma + numero
  fin_mientras
  imprimir ("La suma de los primeros "+ cantidad+ " números naturales es
  " + suma)
fin_algoritmo
```

Para pensar

- Los primera forma de resolver este ejercicio:
 - $\text{suma} \leftarrow 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$
- Es una forma totalmente y lógicamente correcta.
 - Pero está estancado a resolver el problema en particular.
- Pero es mejor un algoritmo que es extensible y genérico para el caso de sumar “N” números.
- En la programación se premia más a las soluciones genéricas dado que pueden aplicarse en múltiples contextos y ayudar a resolver otros problemas.

Conclusión

- La computadora no hace nada mágico sino que muchas veces la programamos para que nos imite en como hacemos nosotros las cosas.
 - Solo que es mucho más veloz
 - Mucho más precisa
 - No tiene días malos...

Ejemplos para afianzar lo visto

- Hagamos un algoritmo que calcule la productoria de los primeros “N” números naturales.
- Hagamos un algoritmo que calcule la “divisoria” de los primeros “N” números naturales.
- Hagamos un algoritmo que sume los primeros “N” números pares.

Confucio Dijo:

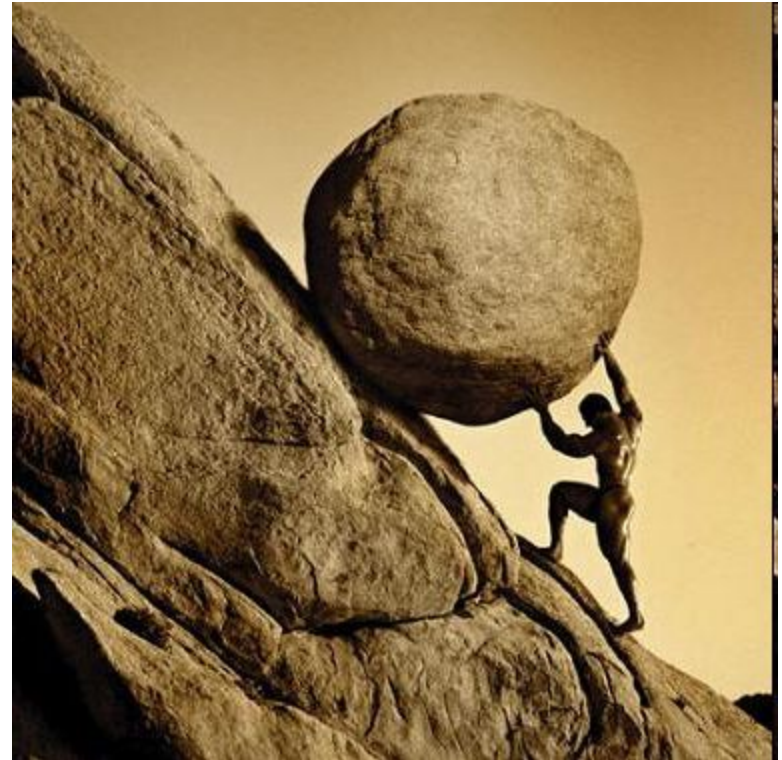
- Oigo y Olvido
- Veo y entiendo
- **Hago** y comprendo



Yo insisto diciendo

- **HAGAN**

- Es el camino más fácil



FIN