UNIVERSIDAD NACIONAL DE HURLINGHAM

Estructuras de Datos

Profesor

Sergio Gonzalez

UNIVERSIDAD NACIONAL DE HURLINGHAM

Unidad 1: Introducción

Profesor

Sergio Gonzalez

OBJETIVOS DE LA MATERIA

- Comprender la noción de dato y de estructuras de datos, y su importancia e interrelación estrecha con la estructura algorítmica de un programa.
- Entender la diferencia entre acceso aleatorio y acceso secuencial.
- Conocer la idea de interface de una estructura de datos

- Conocer la interfaz de distintas estructuras de datos básicas (pilas, colas, listas, árboles, hashing y grafos.) y utilizarlas adecuadamente
- Familiarizarse con las nociones de ámbito y de pasaje de parámetros por valor o referencia.
- Manejar alguno de los principios básicos de diseño de la interfaz de una estructura de datos

- Comprender el concepto de asignación dinámica de memoria, y hacer programas que hagan un manejo dinámico explícito de memoria en forma adecuada.
- Familiarizarse con las tareas de compilar y vincular programas para lograr un ejecutable.
- Poder resolver problemas mediante programas recursivos, y entiender la diferencia entre una resolución recursiva y otra iterativa.

Programa – Unidades

- UNIDAD 1: INTRODUCCIÓN A LENGUAJES DE PROGRAMACIÓN.
- UNIDAD 2: TIPOS DE DATOS ABSTRACTOS.
- UNIDAD 3: FUNCIONES RECURSIVAS, ARREGLOS UNIDIMENSIONALES Y BIDIMENSIONALES.
- UNIDAD 4: PILAS DEFINICIÓN Y TERMINOLOGÍA DE PILA.

- UNIDAD 5: COLAS
- UNIDAD 6: LISTAS
- UNIDAD 7: ARBOLES
- UNIDAD 8: GRAFOS

Bibliografía

- Chris Okasaki, Purely Functional Data Structures. Cambridge University Press, 1998.
- Mark Allen Weiss , Data Structures and Algorithm in C. Addison-Wesley, 1997 (2da edición).

Calendario académico

Segundo cuatrimestre	Inicio de clases	12 de agosto
	Finalización de clases	30 de noviembre
Inscripción a exámenes finales de octubre		23 al 28 de septiembre
Exámenes finales de octubre		30 de septiembre al 5 de octubre
Semana de evaluación integradora		9 al 14 de diciembre
Inscripción a exámenes finales de diciembre		2 al 7 de diciembre
Exámenes finales de diciembre		16 al 21 de diciembre

http://www.unahur.edu.ar/es/calendario-academico

Cursada – Metodo de aprobación

- Asistencia minima: 75% de las clases.
- 2 Parciales c/u con recuperatorio (nota de recuperatorio reemplaza a la del parcial).
- Aprobación de cursada:
  - Promoción (sin final): Promedio de parciales mayor o igual a 7. No menos de 6 en ambos parciales.
  - Regularización: Promedio de parciales mayor o igual a 4. No menos de 4 en ambos parciales.

Cursada – Metodo de aprobación

- Evaluación integradora: Nota cursada entre 4 y 6. Examen de toda la materia en primer fecha de final. Nota de evaluación integradora se promedia con nota de cursada. Posible promoción sin final.
- Final: Examen de materia completa. Anotarse para la fecha deseada. Se aprueba con 4. 10 fechas posibles luego de regularizar.

- POSIBLES FECHAS DE EXAMENES
  - 1er. Parcial: Unidades 1,2,3,4,5 → 30/09/2019
  - Recuperatorio 1er. Parcial → 07/10/2019
  - 2do Parcial: Unidades 6,7,8 → 22/11/2019
  - Recuperatorio 2do Parcial → 29/11/2019

UNIDAD 1: INTRODUCCIÓN A LENGUAJES DE PROGRAMACIÓN (PYTHON).

OBJETIVOS:

- Utilizar reglas sintácticas. Tipos de datos primitivos y referenciales.
- Definir funciones y variables. Operadores. Estructuras de control.
- Incorporar conceptos básicos necesarios para escribir y ejecutar un programa en este lenguaje de programación.
- Proporcionar ejemplos y realizar ejercicios.

Estructura básica de computadoras

- Maquina digital, electrónica y programable para el procesamiento de información.

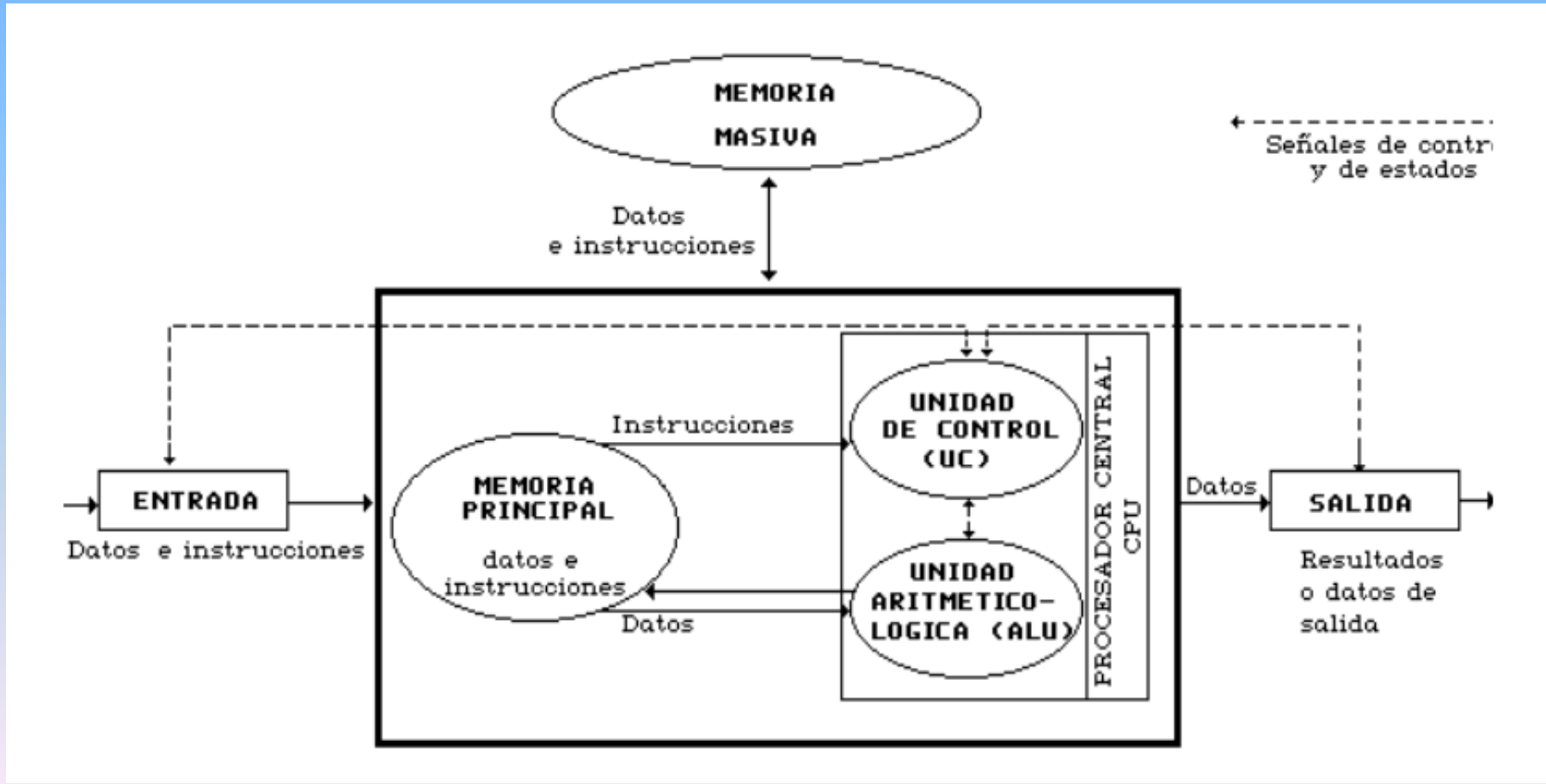
Estructura básica de computadoras

- Operaciones básicas

Estructura básica de computadoras

- Operaciones básicas
  - Entrada de datos
  - Salida de datos
  - Almacenamiento
  - Recuperación
  - Transmisión
  - Recepción
  - Tratamiento

Estructura básica de computadoras



Estructura básica de computadoras

- CPU = UC + ALU
  - Unidades de procesamiento
  - Operaciones básicas: Suma, resta, algebra de Boole, etc.
  - Conjunto de instrucciones
- Memoria: Almacenamiento de datos y programas
  - Memoria principal: Datos en ejecución
  - Memoria masiva: Grandes cantidades de datos

Estructura básica de computadoras

- Estructuración de Memoria

Estructura básica de computadoras

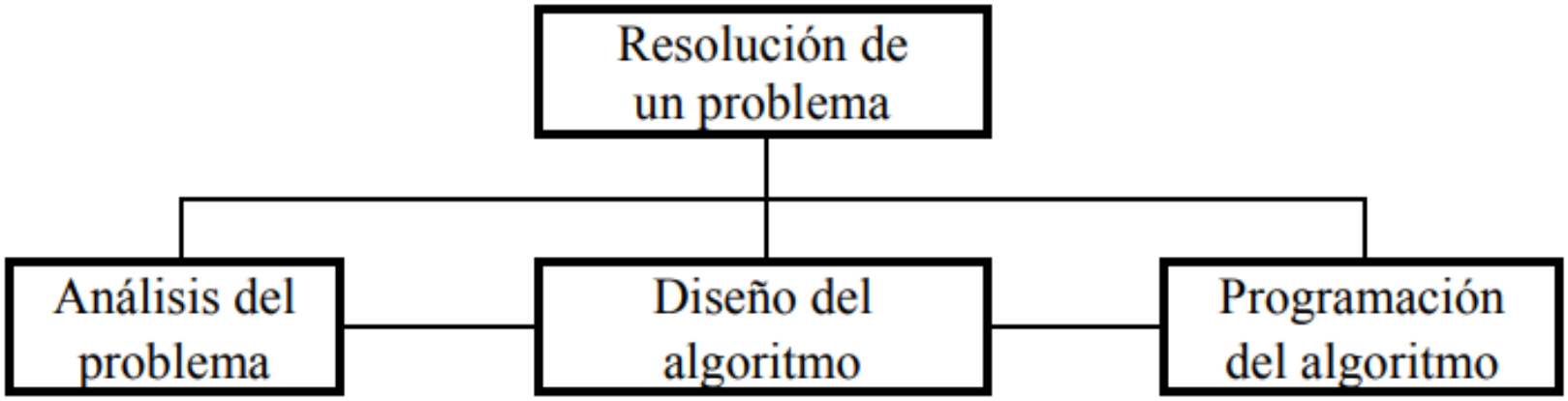
- Estructuración de Memoria
  - Alfabeto binario: Unidad elemental el bit (0 o 1)
  - Conjunto de 8 bits forman un byte
  - 1 Kbyte = 1 KB = 2^{10} bytes = 1024 bytes = 8192 bits

Algoritmos y programas

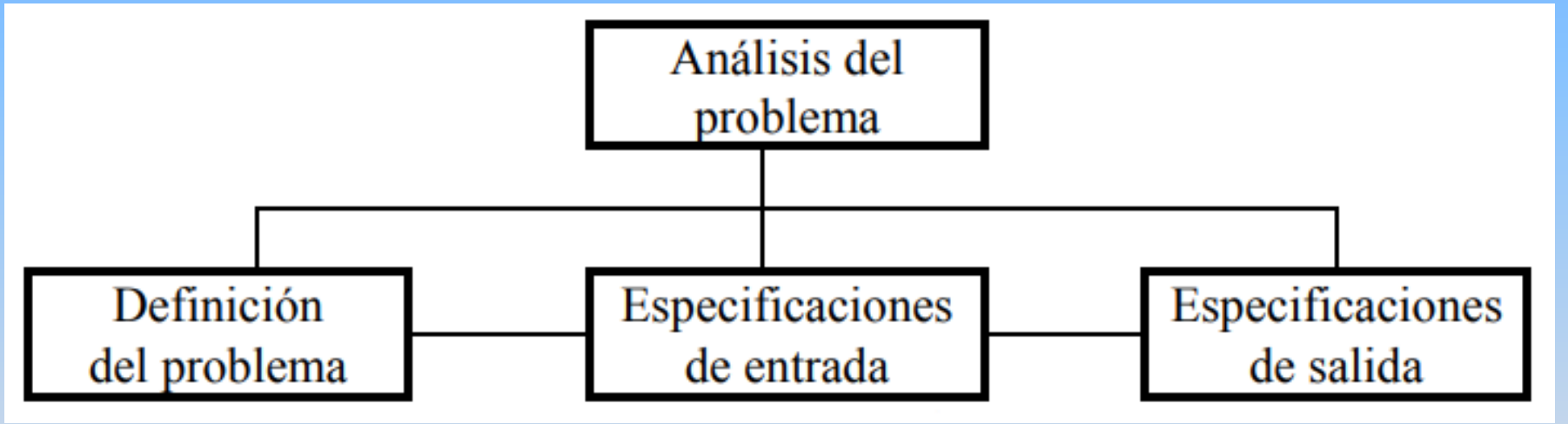
Algoritmos y programas

- Un algoritmo es un conjunto de pasos para resolver un problema.
- Programa: Forma de implementar un algoritmo en una computadora.
- Conjunto de instrucciones que interpreta el procesador.

Algoritmos y programas



Análisis del problema



Análisis del problema: Ejemplo

Elaborar el análisis para obtener el área y la longitud de una circunferencia.

Análisis del problema: Ejemplo

Elaborar el análisis para obtener el área y la longitud de una circunferencia.

1.- Utilizar las fórmulas del área y la circunferencia en función del radio.
2.- Las entradas de datos se reducen al dato correspondiente al radio del círculo. Dada la naturaleza del mismo y el procesamiento al cual lo someteremos, su tipo de dato debe ser un número real.
3.- Las salidas serán dos variables también reales: área y circunferencia.

La finalización de la fase de análisis del problema nos llevaría al siguiente resultado:

Entradas: Radio del círculo (variable RADIO).
Salidas: Superficie del círculo (variable AREA).
Circunferencia del círculo (variable CIRCUNFERENCIA).
Variables: RADIO, AREA, CIRCUNFERENCIA: tipo real.

Diseño de algoritmo

- Orden de realización de los pasos
- Pasos definidos y sin ambigüedad
- Cantidad finita de pasos

Diseño de algoritmo: Ejemplo 1

Diseñar un algoritmo que responda a la pregunta: ¿Qué debo hacer para ver la película XYZ?.

Diseño de algoritmo: Ejemplo 1

Diseñar un algoritmo que responda a la pregunta: ¿Qué debo hacer para ver la película XYZ?.
Un primer análisis nos conduce a un esbozo de solución, descomponiéndolo en cuatro módulos sucesivos:

1 *ir al cine donde proyectan XYZ*
2 *comprar una entrada*
3 *ver la película*
4 *regresar a casa*

Diseño de algoritmo: Ejemplo 2

Averiguar si un número es primo o no, suponiendo que razonamos de la siguiente forma: "Del análisis del hecho de que un número N es primo si sólo puede dividirse por sí mismo y por la unidad, un método que nos puede dar la solución sería dividir sucesivamente el número por 2, 3, 4..., etc. y, según el resultado, podríamos resolver el problema". Un diseño del mismo sería:

Diseño de algoritmo: Ejemplo 2

Averiguar si un número es primo o no, suponiendo que razonamos de la siguiente forma: "Del análisis del hecho de que un número N es primo si sólo puede dividirse por sí mismo y por la unidad, un método que nos puede dar la solución sería dividir sucesivamente el número por 2, 3, 4..., etc. y, según el resultado, podríamos resolver el problema". Un diseño del mismo sería:

1. Inicio
2. Poner X igual a 2 (X = 2, X, variable que representa a los posibles divisores de N)
3. Dividir N por X (N/X)
4. Si el resultado es entero, entonces N no es primo, y saltar al punto 9 (en caso contrario continuar el proceso en el siguiente punto, 5)
5. Incrementar X en una unidad
6. Si X es menor que N saltar al punto 3 (en caso contrario continuar el proceso en el siguiente punto, 7)
7. Declarar N es primo;
8. Saltar al Fin (punto 10)
9. Declarar N no es primo
10. Fin

Representación de algoritmo

- Pseudocódigo
  - Sistema de notación
  - Estructuras
- Diagramas de flujo
  - Organigramas
  - Símbolos gráficos

Pseudocódigo: Ejemplo

Supongamos que tenemos un algoritmo para averiguar si un número es par, que puede ser descrito narrativamente de la siguiente forma: "Si restando consecutivamente doses del número se obtiene el numero 2, es par, si se obtiene otro valor (el 1), entonces es impar". Este algoritmo escrito en pseudocódigo sería:

```
leer N
mientras N > 2 hacer
    N ← N - 2
si N = 2 entonces
        escribe "es par"
sino
        escribe "es impar"
fin
```

Diagrama de flujo: Símbolos

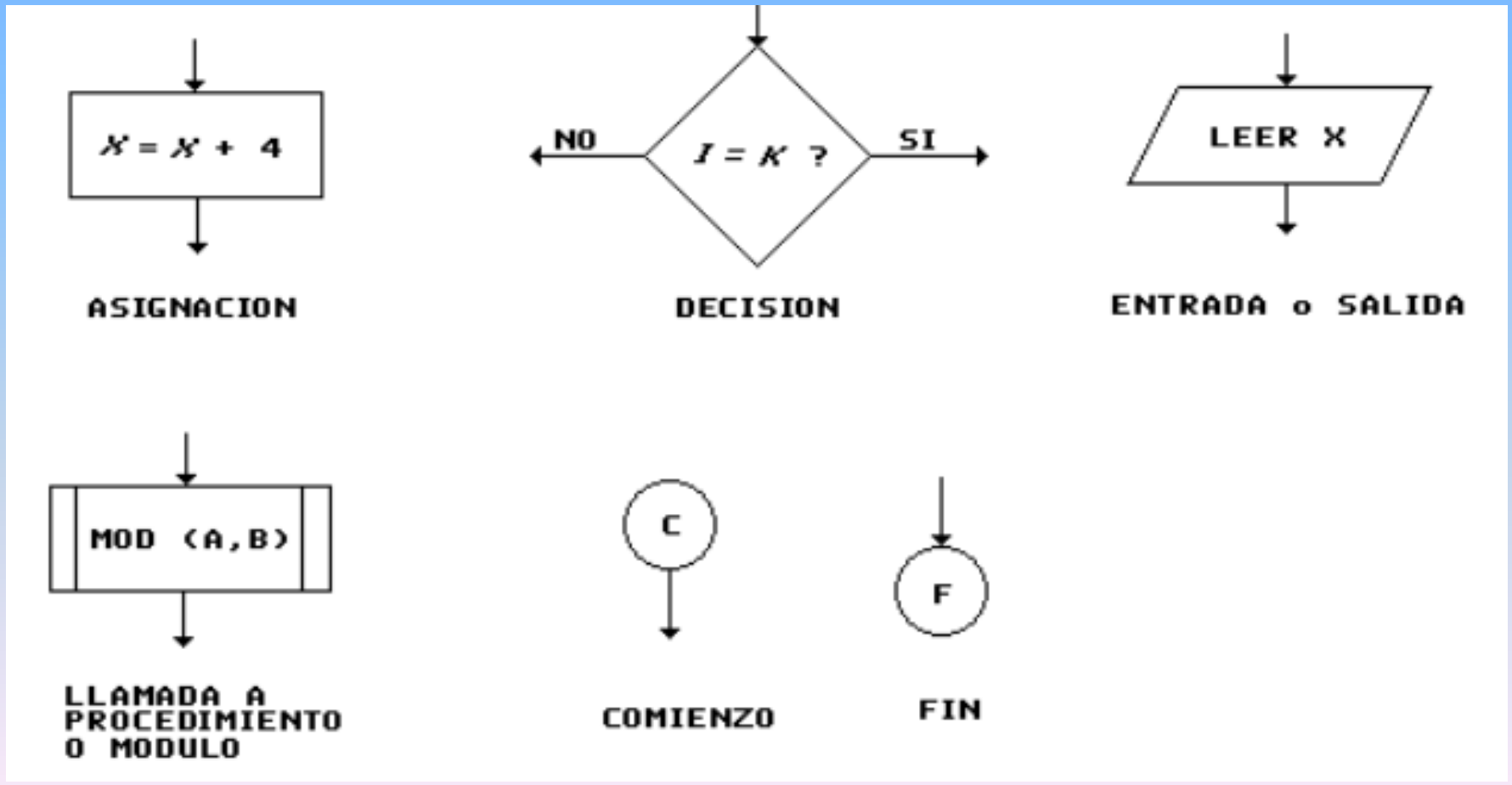
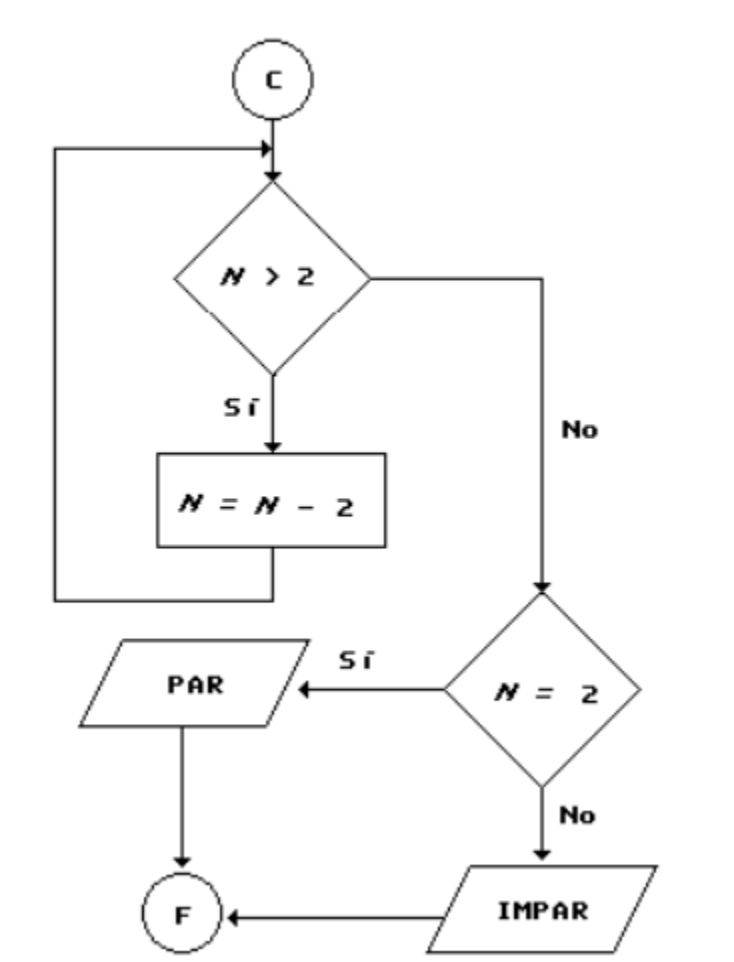
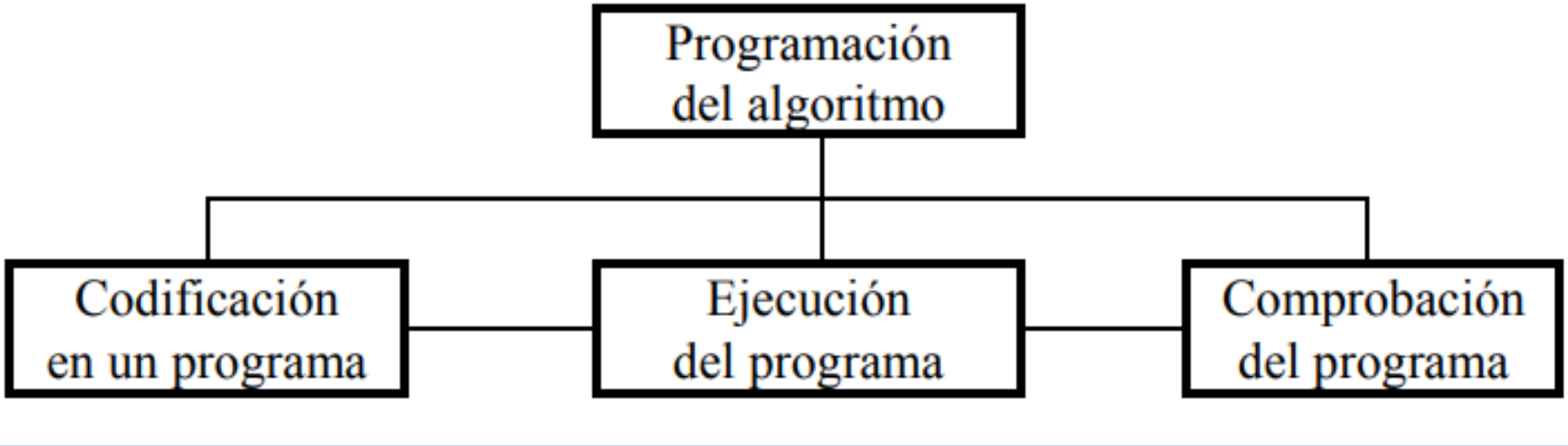


Diagrama de flujo: Ejemplo



Programación del algoritmo



Codificación de un programa

Codificación de un programa

- Lenguaje maquina.
  - 1111s y 0000s

Codificación de un programa

- Lenguaje maquina.
  - 1111s y 0000s
- Lenguajes de programación de alto nivel.
  - Texto similar a lenguaje humano

Traducción de un programa

- Editor de texto -> Programa fuente
- Programa traductor -> Texto a lenguaje maquina
  - Compilador
  - Montador (Linker)
  - Programa ejecutable

Traducción de un programa

- Compilador
  - Análisis lexicográfico
  - Análisis gramatical
    - Sintáctico
    - Semántico
    - Optimización
  - Síntesis

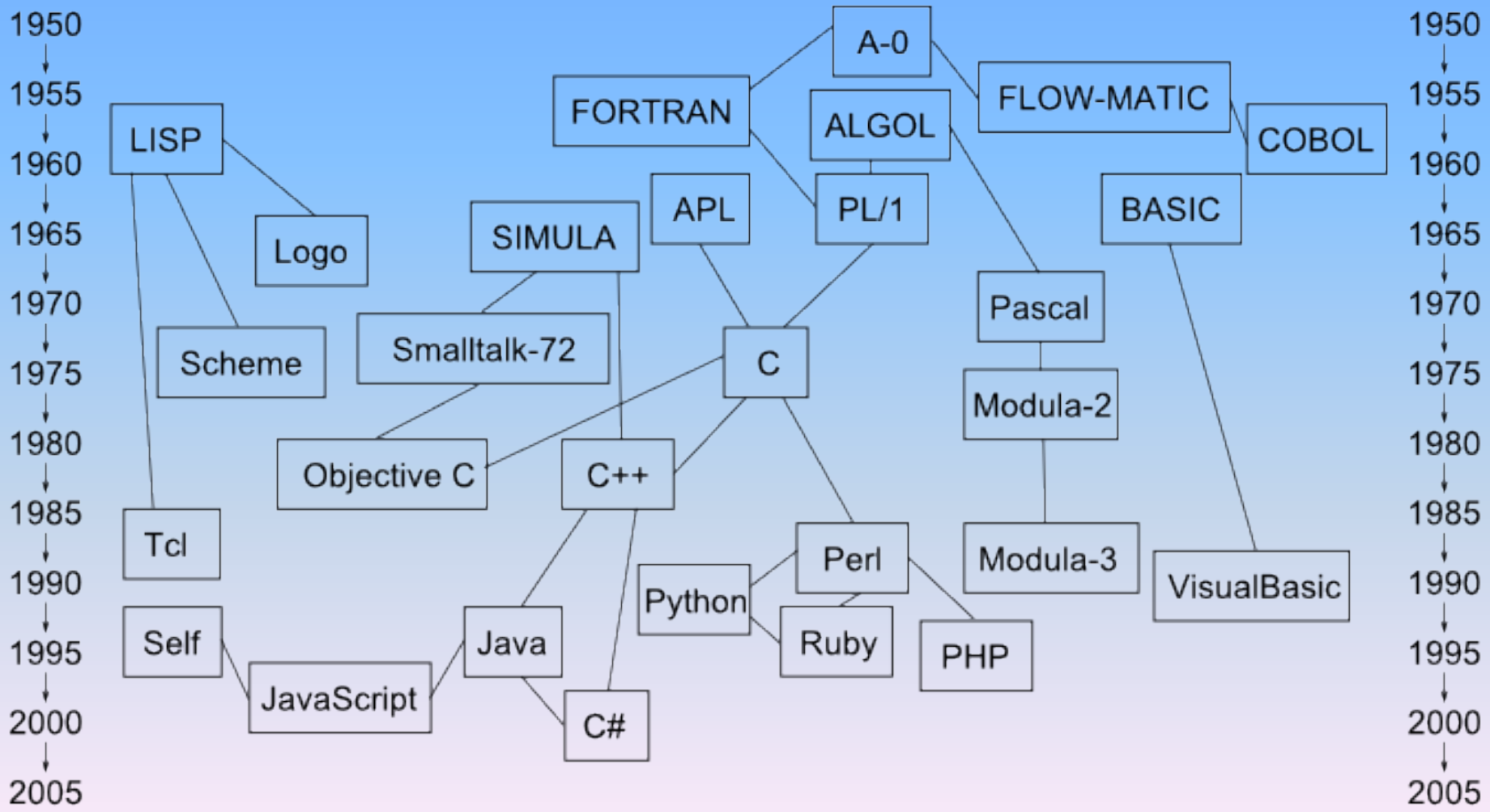
Traducción de un programa

- Interprete
  - Realiza traducción
  - Ejecuta directamente archivo fuente
  - No genera ejecutable

Lenguajes de programación

- Independientes de la maquina
- Similar a lenguaje humano
- Una sola instrucción equivale a varias instrucciones de lenguaje maquina.
- Uso de variables, símbolos y términos usados por los humanos
- No nos preocupamos por organización real en memoria

Evolución de los Lenguajes de Programación



Elementos básicos de lenguajes de programación

- Constantes y variables
- Palabras reservadas
- Expresiones / Operadores
- Entrada / Salida
- Estructura de programa
- Comentarios

Variables

- Contenedores de datos
- Ocupa espacio en RAM
- Nombre / Identificador
  - Letras
  - Números
  - Caracteres especiales
- Tipo

Tipos de datos primitivos

- Representación en la computadora de datos enteros, reales, lógicos, caracteres, etc.

Tipos de datos primitivos

- Representación en la computadora de datos enteros, reales, lógicos, caracteres, etc.
- Interpretación de un patrón de bits
- Usando 1111s y 00000s (base 2)
- Limite en función del tamaño
  - Cantidad de bits asignados
  - Signo
  - Desbordamiento (*Overflow*)

Tipos de datos primitivos

- Al programar, elegimos los tipos de datos a utilizar
  - Define rango acotado (Ahorro de memoria)
  - Operaciones permitidas

Tipos de datos primitivos

Tipo de variable	Bytes que ocupa	Rango de valores
boolean	2	true, false
byte	1	-128 a 127
short	2	-32.768 a 32.767
int	4	-2.147.483.648 a 2.147.483.649
long	8	$-9 \cdot 10^{18}$ a $9 \cdot 10^{18}$
double	8	$-1{,}79 \cdot 10^{308}$ a $1{,}79 \cdot 10^{308}$
float	4	$-3{,}4 \cdot 10^{38}$ a $3{,}4 \cdot 10^{38}$
char	2	Caracteres (en Unicode)

Booleanos

- Boolean
- Lógica booleana
  - Verdadero (True)
  - Falso (False)

Caracteres

- Char
- Tabla de caracteres
  - UNICODE
  - ASCII

https://unicode-table.com/es/

https://elcodigoascii.com.ar/

Enteros

- Byte
- Short
- Int
- Long

- Números con signo, rango posible depende del tamaño de la variable en memoria.

Números reales – Punto flotante

- Float
- Double

- Precisión: Cantidad de cifras decimales
- Similar a notación científica

$5.75 \times 10^{4} = 57\,500$

$6.7 \times 10^{-5} = 0.000\,067$

Palabras reservadas

abstract	do	implements	private	throw
boolean	double	import	protected	throws
break	else	instanceof	public	transient
byte	extends	int	return	true
case	false	interface	short	try
catch	final	long	static	void
char	finally	native	super	volatile
class	float	new	switch	while
continue	for	null	synchronized	
default	if	package	this	

Expresiones / Operadores

- Operar con tipos de datos primitivos
  - Asignación
  - Aritméticos
  - Condicionales
  - Bit

Operadores de asignación

- Cargar valores en variables
  - =
  - <-
  - ++ / --
  - += / -=

Operadores Aritméticos

operador	significado
+	Suma
-	Resta
*	Producto
/	División
%	Módulo (resto)

Operadores condicionales

operador	significado
<	Menor
>	Mayor
>=	Mayor o igual
<=	Menor o igual
==	Igual
!=	Distinto
not	No lógico (NOT)
and	"Y" lógico (AND)
or	"O" lógico (OR)

Entrada / Salida

- Por pantalla / teclado
- Archivos

Estructura de programa

- Indentado
- Uso de ";"
- Sentencias entre llaves "{}"

Comentarios

- Líneas que no son ejecutadas

Estructura de un programa

- Declaración de variables
  - Tipo
  - Nombre
- Sentencias que implementan el algoritmo

```
Identificador del programa o módulo
{sección de declaraciones}
inicio
        {datos de entrada}
        {sentencias imperativas, que ejecutan el algoritmo correspondiente}
        {datos de salida}
fin
```

Que es PYTHON?

- Python es un lenguaje de programación interpretado con una sintaxis que favorece la legibilidad del código.

Lenguaje de programación Python

- Creado a finales de los 80s por Guido Van Rossum
- Multiparadigma: Orientado a objetos, programación funcional y programación imperativa.
- Licencia GNU GPL
- Tipado dinamico
- Codigo legible y transparente

Entornos de desarrollo Python

- Spyder
- Sublime
- Atom
- Thonny
- PyCharm
- Eclipse + PyDev
- Jupyter Notebook

Componentes de entorno de desarrollo

- Panel de Definiciones
  - Coloreado de Palabras según la sintaxis
  - Indentación automática.
  - Emparejamiento de parentesis y llaves.
  - Introducción de comentarios

Consideraciones antes de programar en Python

- En Python hay diferencia entre mayúsculas y minúsculas.
- Las línea de código no deben terminar con "; "
- Los comentarios; si son de una línea debe comenzar con "#" y si ocupan más de una línea van entre ‘ ’ ’
- El indentado es obligatorio, ya que no se usan llaves para separar bloques de codigo

Nuestro primer programa

```
#!/usr/bin/python

print("Hola mundo")
```

Declaración de variables

Nombrevariable = valor

Ejemplo:

```
dias = 2
decision = True
letra = "C"
radio = 25.63

x , y , z = 34 , 25 , 12
x = y = "Hola"
```

Declaración de variables

- Las variables no se declaran definiendo un tipo
- El tipado es automatico
- No se pueden realizar operaciones entre variables de distinto tipo
- Existen funciones para el casteo:
  - int()
  - float()
  - str()

Acá se pueden hacer los primeros 5 ejercicios del TP1

Ingreso de datos por teclado

- Para leer los datos que se introducen en el teclado se utiliza la funcion **input()**

- Se puede utilizar de la siguiente manera:
  - variable = input("Ingrese un numero: ")

Ingreso de otros tipos de datos

- Por defecto, la funcion convierte la entrada a una variable de tipo texto

- Recuerden que es posible castear las variables:

```
variable = int(input("Ingrese un numero entero: "))
```

Estructuras de control

- Orden de realización de los pasos del algoritmo
  - Selectivas
  - Repetitivas

Estructuras de control selectivas

- Decidir que hacer a partir de evaluar una condición.
- Uso de operadores y expresiones lógicas.
- Bifurcaciones en el flujo del programa.

Alternativas simples

- Si – Entonces / If – then.
- Ejecuta una acción solo cuando se cumple una condición

```
si <condición> entonces
    <acciones>
fin_si
```

Alternativas simples

- Si – Entonces / If – then.
- En Python

```
if condicion:
    S1
    S2
    ...
    Sn
```

Alternativas dobles

- Si – Entonces – si_no / If – then – else.

```
si <condición> entonces
    <acciones S1>
si_no
    <acciones S2>
fin_si
```

```
if condicion:
    acciones S1
else:
    acciones S2
```

Alternativas múltiples

- Si – Entonces – si_no – entonces …
- If – then – else if – then …

```
si <condición1> entonces
    <acciones S1>
si_no <condición2> entonces
    <acciones S2>
si_no <condición3> entonces
    <acciones S3>
fin_si
```

```
if condicion1:
    acciones S1
elif condicion2:
    acciones S2
elif  condicion3:
    acciones S3
```

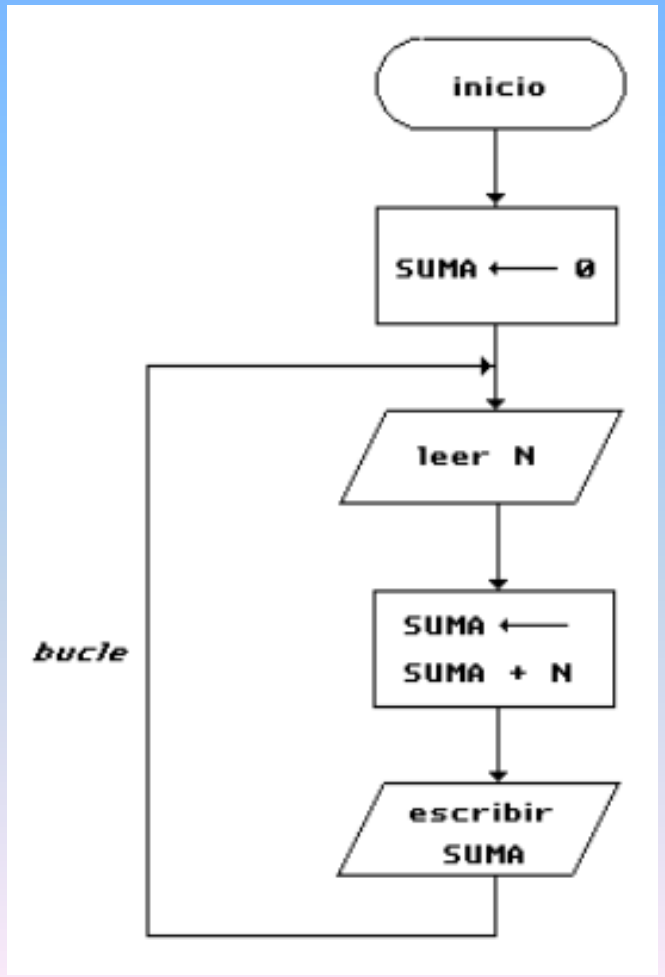
Alternativas múltiples

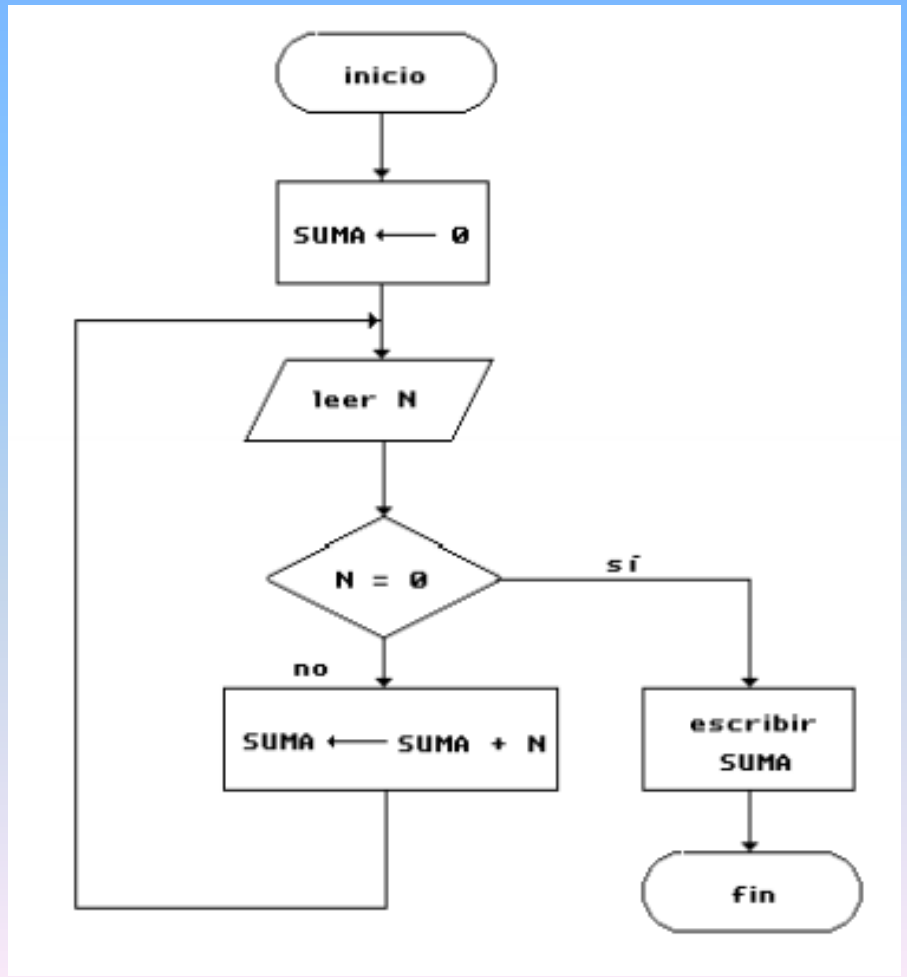
- En Python no existe "switch - case"

Estructuras de control repetitivas

- Conjunto de operaciones que se deben repetir varias veces.
- Ciclo o bucle: Parte de un programa que se repite (iteración) un numero dado de veces o mientras se cumpla una condición.
- Evitar bucles infinitos.

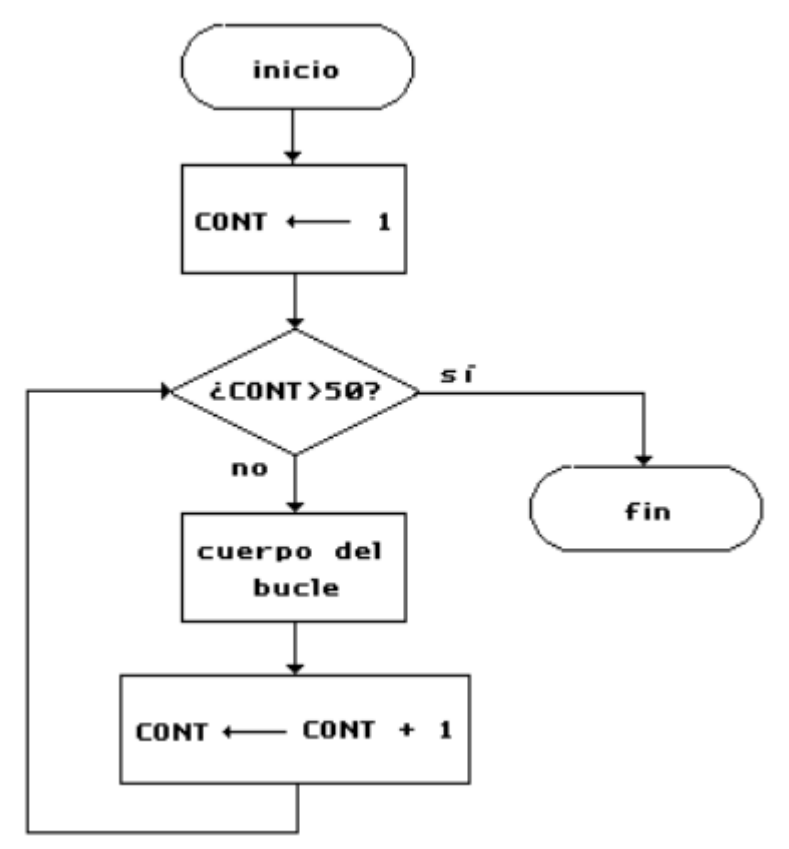
Estructuras de control repetitivas





Estructuras de control repetitivas

- Uso de contadores y acumuladores



Estructuras de control repetitivas

- Tres componentes básicos:
  - Decisión
  - Cuerpo del bucle
  - Salida del bucle

Estructuras de control repetitivas

- Desde – Hasta / For

desde v=vi **hasta** vf **hacer**
  <acciones>
  .
  .
fin_desde

v: variable índice
vi, vf: valores inicial y final de la variable

```
for variable in elemento_iterable:
    <acciones>
```

Estructuras de control repetitivas

- Mientras / While

mientras condición **hacer**
    <acciones>
fin_mientras

```
while condicion:
    <accion>
```

Estructuras de control repetitivas

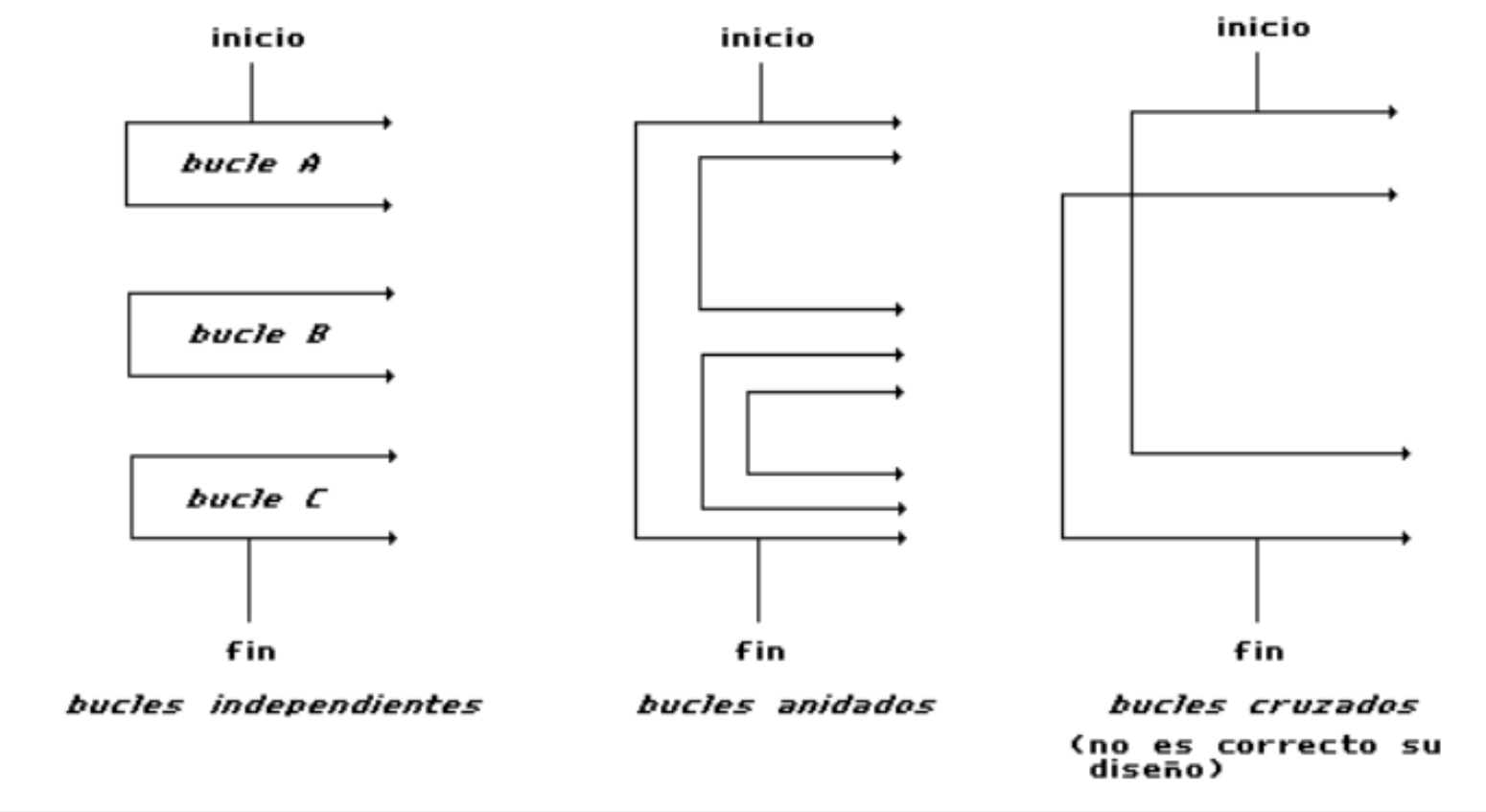
- En Python no existe "do – while"

Estructuras de control repetitivas

- Las dos estructuras de control repetitivas son intercambiables entre si.
- A partir de la declaración de variables accesorias

Estructuras de control repetitivas

- Anidamiento de bucles



Funciones

- Muchas veces, partes del programa se repiten
- Es una buena practica, separar estas partes para crear "Funciones"
- Las funciones realizan tareas especificas
- El programa principal "llama" a las funciones
- Parámetros: Lista de valores de entrada
- Única salida

Funciones

- Parámetros: Lista de valores de entrada
- Única salida
- Nombre identificatorio

función nombre-función(par1,par2,par3...)
  <acciones>
fin función

par1,par2, ,	*lista de parámetros formales o argumentos:* son nombres de identificadores que sólo se utilizan dentro del cuerpo de la función.
nombre-función	nombre asociado con la función, que será un nombre de identificador válido sobre el que se almacenará el resultado.
<acciones>	instrucciones que constituyen la función, y que deben contener al menos una acción que asigne un valor como resultado de la función.

Funciones

- Invocación de funciones
- Utilizando su nombre y los parámetros entre paréntesis, coincidiendo en cantidad y en tipo con los de la definición de la función

nombre-función (lista de parámetros actuales)

nombre función	función que es invocada.
lista de parámetros actuales	constantes, variables, expresiones, valores de funciones, nombres de subprogramas, que se corresponden con los parámetros formales, que hemos visto durante en la declaración de funciones.

Funciones

Escribir la función: $y = x^n$ (potencia n de x, tanto si es positiva, como negativa) y utilizarla para calcular $1/(2.5)^3$

```
función potencia (x, n)
        inicio                          parámetros formales (real y entero)
  y ← 1                                 función interna (valor absoluto)
  desde i = 1 hasta abs(n) hacer
     y ← y * x
  fin_desde
  si n < 0 entonces y ← 1/y
  potencia ← y
fin
```

Su utilización sería:

```
z ← potencia (2.5, -3)
                    parámetros actuales
```

Funciones: Ámbito de variables

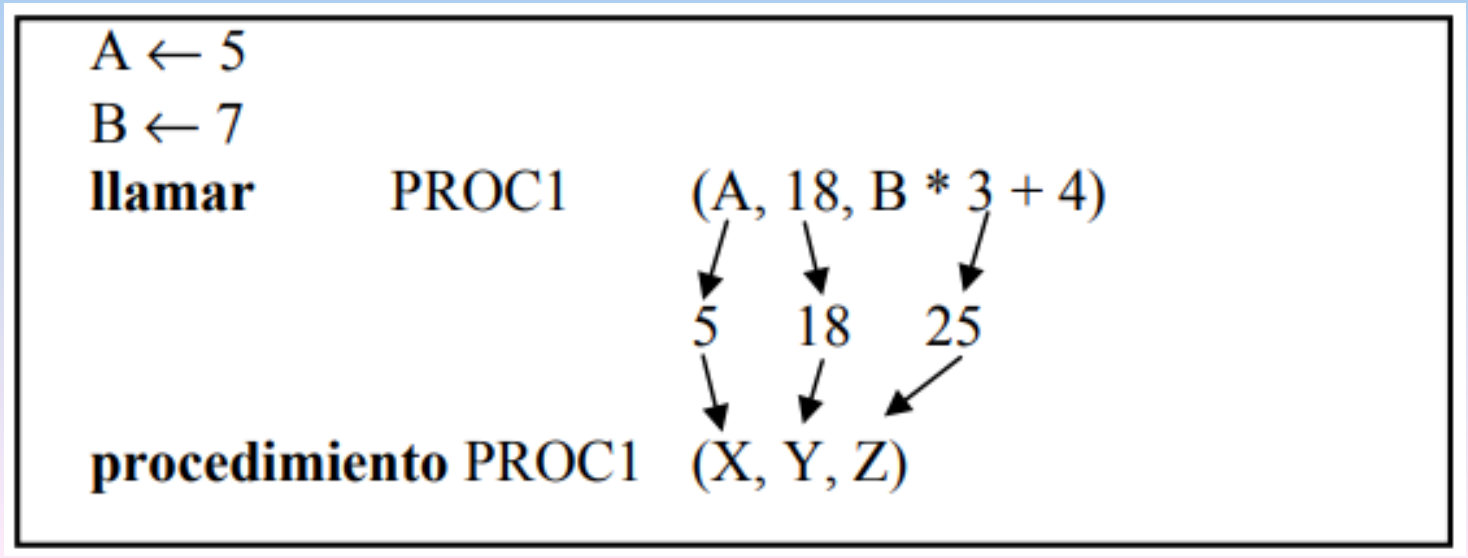
- La ejecución de una función no debe afectar al resto del programa
- Desarrollos independientes
- Variable local
  - Declarada y utilizada solo dentro de la función
  - Compartir nombres
  - Valores no accesibles desde afuera
- Ámbito (*scope*): Parte del programa donde la variable se reconoce como tal

Funciones: Paso de parámetros

- Entrada (Argumentos): Desde el programa principal a la función
- Salida: Resultados de la función, desde la función al programa principal
- Entrada / Salida: Mismo parámetro para mandar argumentos y devolver resultados.
- Tipos:
  - Paso por valor
  - Paso por referencia

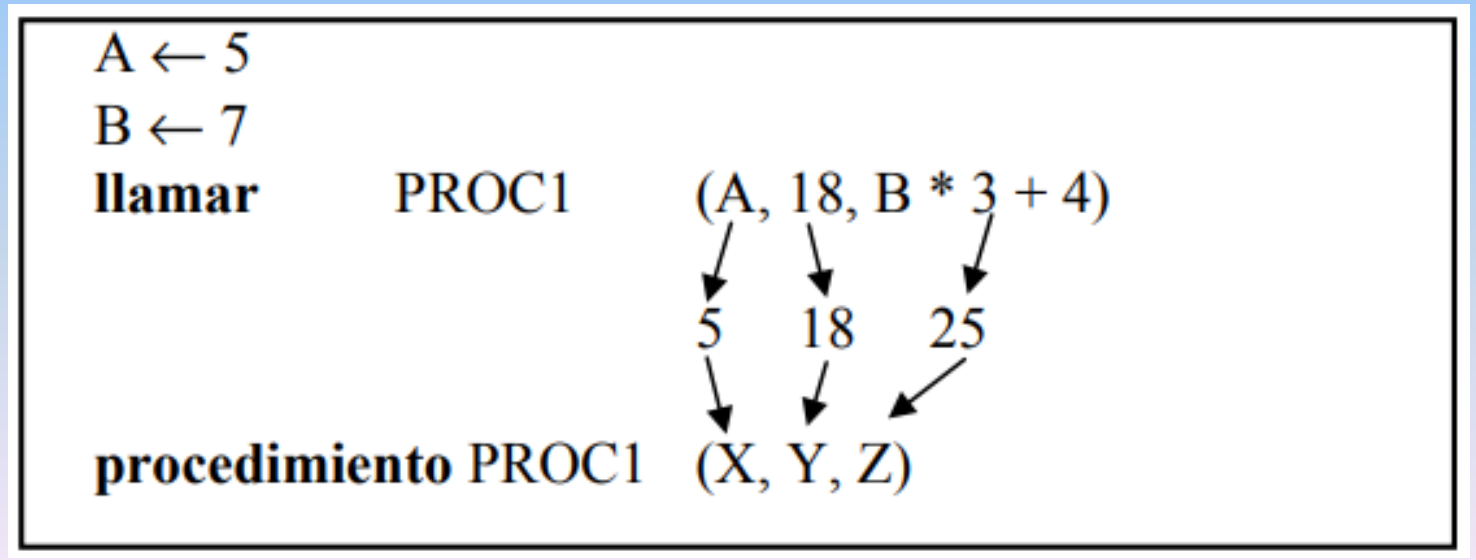
Funciones: Paso por valor

- La función recibe el valor de la variable
- Modificaciones a dicha variable, solo tienen efecto dentro de la función



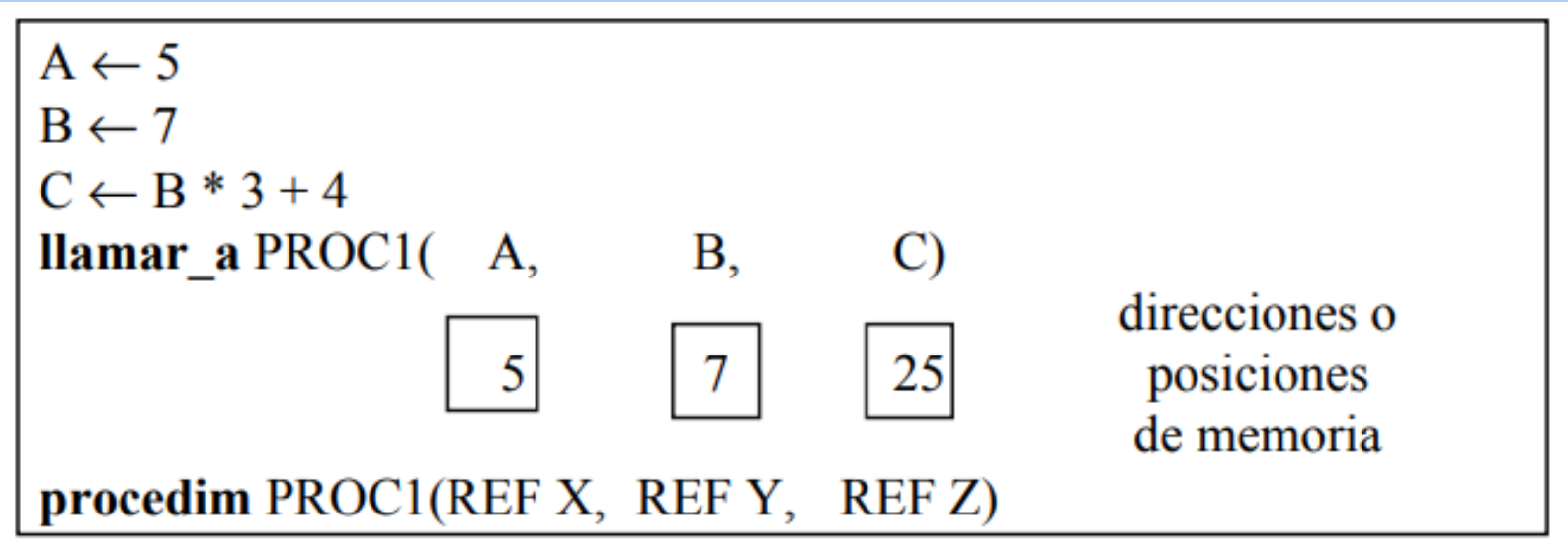
Funciones: Paso por valor

- Todos los parámetros son solo de entrada
- Salida por retorno de la función



Funciones: Paso por referencia

- Parámetros de Entrada / Salida
- Modificaciones a variables dentro de la función se mantienen fuera de ella



Funciones: Paso de parametros

```
algoritmo ABC
inicio
  A ←3
  B ← 5
  C ←17
  llamar_a SUB(A, B, C)
  escribir A,B,C
fin

procedimiento SUB(x,y,z)

  inicio
    x ← x+1
    z ← x+y
  fin
```

(a)

```
algoritmo ABC
inicio
  A ←3
  B ← 5
  C ←17
  llamar_a SUB(A, B, C)
  escribir A,B,C
fin

procedimiento SUB(REF x,REF y,REF
z)
  inicio
    x ← x+1
    z ← x+y
  fin
```

(b)

Funciones: Python

- Tipos primitivos: Solo se permite paso por valor
- No se define tipo de salida
- No se definen tipos de parametros
- Se pueden poner valores por defecto y cambiar el orden de los parametros

```
def nombreFuncion(parametros):
    salida = 0
    <accionesFuncion>
    return salida
```

Funciones: Python

- Función suma de dos números enteros

```
def suma(x , y):
    salida = x + y
    return salida
```

```
def suma(x = 0 , y):
    salida = x + y
    return salida
```

```
n = suma(5 , 4)
n = suma(x = 5 , y = 4)
n = suma(y = 4 , x = 5)
x, y = 5 , 4
n = suma(y = y , x = x)
```

Y las estructuras de datos????

- Programa = Estructura de datos + Algoritmo
  - Estructura: Forma de organizar y representar los datos
  - Algoritmo: Pasos para resolución de problemas