

# Informática para la ingeniería



*Campus  
de excelencia internacional*

# Informática para la ingeniería

## Tema 1



# Tipos de datos simples. Variables y operaciones

- ❖ Variables y tipos de datos simples.
- ❖ Instrucciones y operadores: asignación, aritméticos, relacionales, booleanos. Precedencia de los operadores.
- ❖ Cadenas de caracteres y operadores de cadenas (concatenación y comparación).
- ❖ Ejecución y depuración de programas.

## Variable y tipos de dato

**Variable.** Formada por un espacio en el sistema de almacenaje (memoria principal) y un nombre (identificador) que está asociado a dicho espacio. Ese espacio contiene un valor. El nombre de la variable es la forma usual de referirse al valor almacenado. Esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa.

Un **tipo de dato** es un atributo de los datos que indica al ordenador sobre la clase de datos que se va a trabajar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

Los tipos de datos comunes son: números enteros, números con signo (negativos), números de coma flotante (decimales), cadenas alfanuméricas (y unicodes), estados, etc.

## Ámbito y nombrado de la variable

El **ámbito** de una variable es la zona del programa en la que se define la variable, donde existe y puede usarse.

Algunas reglas para nombrarlas:

- Pueden ser de cualquier longitud
- Pueden contener letras con acentos, diéresis, eñes, etc. Tanto minúsculas, como mayúsculas. También dígitos y "\_"
- No pueden comenzar por un dígito
- Diferencia entre mayúsculas y minúsculas

## Ejemplos

número = 5

númeroFlotante = 5.6

cadena = "conjunto caracteres"

Bandera = True

# Variables

Nombres válidos:

```
miVariable  
piña  
duración  
var1  
x  
X  
_x
```

Nombres no válidos:

```
1var  
my#Variable
```

Nombres distintos:

```
miVariable  
mivariable  
Mivariable  
MiVariable  
MiVariáble
```

# Tipos de datos básicos: números

Enteros:

```
>>> edad = 25
>>> edad
25
```

Complejos:

```
>>> x=1+3j
>>> y=1+4j
>>> x+y
(2+7j)
```

Reales o flotantes:

```
>>> precio=34.4
>>> precio
34.4
```

Hexadecimales:

```
>>> n=0xA
>>> n
10
```

Booleano:

```
>>> t=True
>>> y=False
```

Octales:

```
>>> n=0o10
>>> n
8
```

## Tipos de datos básicos: cadenas de caracteres (*strings*)

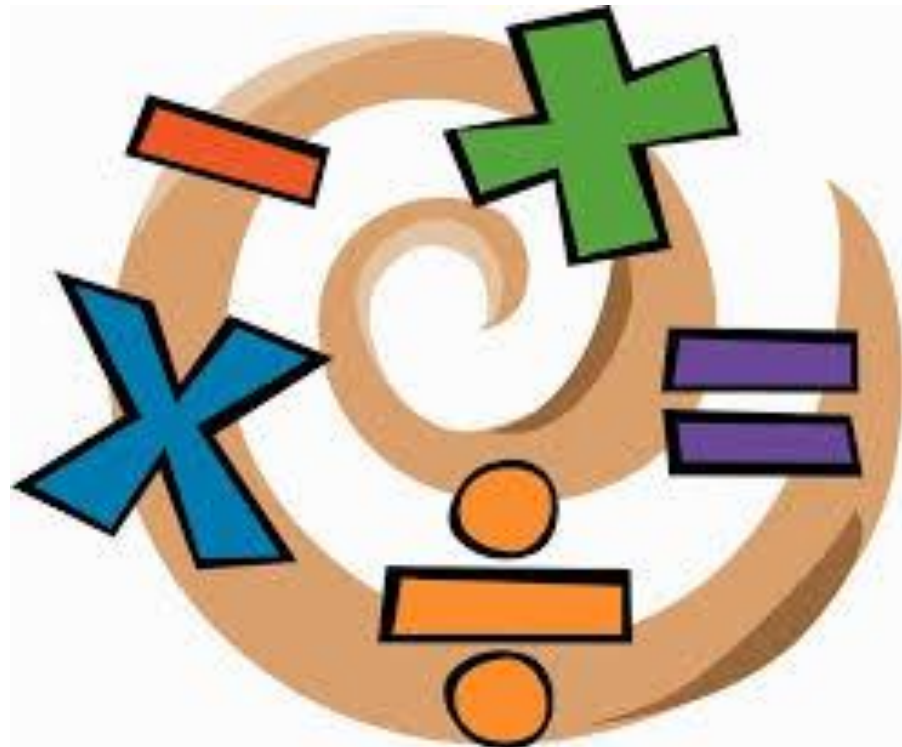
- Secuencias de caracteres delimitadas entre comillas simples (') o dobles (").
- Se accede a sus elementos usando corchetes [].
- Las cadenas son inmutables (no se pueden modificar).
- Delimitar cadenas de múltiples líneas entre triples comillas simples (""") o triples comillas dobles (""").

```
cadena = 'Soy una cadena de caracteres'
cadena_1 = "4"
cadena_2 = "8.937"
cadena_3 = ""
multi = '''Esta es
una cadena que ocupa dos líneas'''
```



# Tipos de operadores

- ☐ Asignación
- ☐ Aritméticos
- ☐ Compuestos
- ☐ Relacionales
- ☐ Booleanos



## Tipos de operadores: asignación

**Asignación:** La variable de la izquierda toma el valor resultante de la constante, variable o expresión de la derecha.

```
a = 3 # a toma valor 3
```

```
r = a # r toma valor de a
```

**Encadenamiento:** el valor de la derecha (variable, valor, llamada a función, etc.) se asigna a todas las variables que están a la izquierda.

```
a = b = c = 3 # a, b y c toman valor 3
```

```
r = s = a # r y s toman el valor de a
```

**Asignación múltiple:**

```
d, e, f = a, b, c # d=a, e=b, f=c
```

```
a, b = b, a # a y b intercambian sus valores
```

## Tipos de operadores: aritméticos

Símbolo	Significado	Ejemplo	Resultado
+	Suma	<code>a = 10 + 5</code>	<code>a es 15</code>
-	Resta	<code>a = 12 - 7</code>	<code>a es 5</code>
-	Negación	<code>a = -5</code>	<code>a es -5</code>
*	Multiplicación	<code>a = 7 * 5</code>	<code>a es 35</code>
**	Exponente	<code>a = 2 ** 3</code>	<code>a es 8</code>
/	División	<code>a = 12.5 / 2</code>	<code>a es 6.25</code>
//	División entera	<code>a = 12.5 // 2</code>	<code>a es 6.0</code>
%	Módulo	<code>a = 27 % 4</code>	<code>a es 3</code>

## Tipos de operadores: compuestos

Combinan un operador aritmético y una asignación.  
Manera compacta de modificar el valor de una variable.

Operador	Ejemplo	Equivale a
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>x -= 5</code>	<code>x = x - 5</code>
<code>*=</code>	<code>x *= 5</code>	<code>x = x * 5</code>
<code>/=</code>	<code>x /= 5</code>	<code>x = x / 5</code>
<code>%=</code>	<code>x %= 5</code>	<code>x = x % 5</code>
<code>//=</code>	<code>x //= 5</code>	<code>x = x // 5</code>
<code>**=</code>	<code>x **= 5</code>	<code>x = x ** 5</code>

## Tipos de operadores: relacionales

Operador	Significado
<	menor estricto
<=	menor o igual
>	mayor estricto
>=	mayor o igual
==	igual a
!=	distinto de

```
>>> 1 < 2
True
>>> 3 > 34
False
>>> 23 == 45
False
>>> 34 != 323
True
```

```
>>> True and (5<4)
False
>>> True or (5<4)
True
>>> not(True or (5<4))
False
```

## Tipos de operadores: booleanos

Expresión	Significado
<code>a and b</code>	El resultado es <code>True</code> solamente si <code>a</code> es <code>True</code> y <code>b</code> es <code>True</code> de lo contrario el resultado es <code>False</code>
<code>a or b</code>	El resultado es <code>True</code> si <code>a</code> es <code>True</code> o <code>b</code> es <code>True</code> de lo contrario el resultado es <code>False</code>
<code>not a</code>	El resultado es <code>True</code> si <code>a</code> es <code>False</code> de lo contrario el resultado es <code>False</code>

`aprobado = True`  
`suspenso = False`

## Tipos de operadores: cadenas de caracteres

### Concatenar (+):

Une caracteres. Operador +

```
cadena='Esto es un' + ' ejemplo de cadenas'  
print (cadena)                Esto es un ejemplo de cadenas
```

### Multiplicar (\*):

Realiza varias copias de una cadena de caracteres. Operador de multiplicación (\*).

```
ejemplo='multiplicación '*3  
print(ejemplo)                multiplicación multiplicación multiplicación
```

## Tipos de operadores: cadenas de caracteres

### Añadir (+=):

Añade caracteres al final de la variable. Operador +=

```
cadena='Esto es un'  
cadena+=' ejemplo de cadenas'  
print (cadena)           Esto es un ejemplo de cadenas
```

### Indexación [posición]

Acceso a cualquier carácter de una cadena. Se ha de escribir la posición entre corchetes. Las posiciones empiezan en 0 desde la izquierda y en -1 desde la derecha.

```
texto_introducido='Esto es un ejemplo'  
print('La letra es', texto_introducido[0])    La letra es E  
print('La letra es', texto_introducido[2])    La letra es t  
print('La letra es', texto_introducido[5])    La letra es e  
print('La letra es', texto_introducido[-2])   La letra es l
```



## Tipos de operadores: cadenas de caracteres

### Segmento [:]

Localiza una subcadena dentro de una cadena de caracteres. El operador [n:m] devuelve la parte de la cadena desde el "n-ésimo" carácter hasta el "m-ésimo-1". Si se omite el primer índice (antes de los dos puntos), la porción comienza al principio de la cadena. Si omite el segundo índice, la porción llega al final de la cadena.

```
texto_introducido='Esto es un ejemplo'
print(texto_introducido[:4])           Esto
print(texto_introducido[1:6])         sto e
```

# Precedencia de operadores

Conjunto de reglas que especifica en qué orden deben ser evaluadas las operaciones de una expresión.

Se evalúa de mayor a menor precedencia. Para la misma precedencia se evalúa de izquierda a derecha.

Operador	Descripción
( )	Paréntesis
**	Exponenciación
~, +, -	Complemento, más y menos unario
*, /, %, //	Multiplicación, división, módulo y división entera
+, -	Suma y resta
>>, <<	Desplazamiento de bits a la derecha y a la izquierda
&	Y lógica de bits
^,	O exclusiva y O lógica de bits
<=, <, >, >=, <, >	Operadores de comparación
==, !=	Operadores de igualdad
=, %=, /=, //, -=, +=, *=, **=	Operadores de asignación (p. ej. $a += 3 \Leftrightarrow a = a + 3$ )
is, is not	Operadores de identidad
in, not in	Operadores de pertenencia
not, or, and	Operadores lógicos

**3+5+1+2**  
**(3+5)+1+2**  
**( 8 +1)+2**  
**( 9 +2)**  
**11**

**3<sup>2<sup>2</sup></sup>**      **3\*\*2\*\*2**  
**3<sup>4</sup>**      **3\*\*(2\*\*2)**  
**3\*\*4**  
**81**

## Ejecución y depuración de programas

**IDE (Interactive Development Enviroment).** Programa que permite editar, ejecutar y depurar programas. Utilizaremos Thonny, disponible en <https://thonny.org/>.

Funcionalidad:

- Ejecutar instrucciones por línea de comandos “Shell”
- Editar programas
- Ejecutar programas (run)
- Depurar programas paso a paso (debug): step over
- Observar el valor de las variables
- Edición avanzada: autocompletar, buscar y reemplazar
- Introspección: step into, step out y resume
- Puntos de interrupción (breakpoints)

## Comentarios

Las líneas que comienzan por almohadilla (#) son ignoradas. Si una línea contiene una almohadilla, se ignora hasta final de línea (si la almohadilla no forma parte de una cadena).

Los grupos de líneas delimitados por triples comillas simples (') o dobles (""") que no son asignados a variables (cadenas) son ignorados.

```
# esto es un comentario en Python
print "Hola mundo" # otro comentario en Python
a = "Aquí '#' no es un comentario"
""" Este es un ejemplo de comentario multilínea
    Antes de las primeras comillas y después de las
    últimas
    no puede ir ningún otro carácter.
"""
'''Este es un ejemplo de
comentario multilínea
'''
s = '''Este es un ejemplo de
un string que abarca varias líneas'''
```

# Informática para la ingeniería

## Tema 2



# Funciones y Módulos Estándar

- ❖ Concepto de función: llamada, argumentos y resultado.
- ❖ Uso de funciones predefinidas (estándar): entrada y salida por consola (input y print), conversión de tipo (str, int y float) y numéricas (abs, round,...).
- ❖ Uso de cadenas.
- ❖ Concepto de módulo: importación, atributos, funciones.
- ❖ Uso de módulos estándar: math, random, datetime, statistics.

## Funciones

Una **función** contiene un conjunto de líneas de código que realizan una tarea específica.

La función tiene un **nombre** y puede recibir **argumentos** que alteran la realización de la tarea y su **resultado**.

Cuando una función devuelve un **resultado**, éste puede ser asignado a una variable o utilizado dentro de una expresión.

- Ejemplo:

```
>>> x=abs(-25) # muestra el valor absoluto de -25
```

25



Python nos permite definir nuestras propias funciones pero de momento aprenderemos a utilizar funciones que ya existen.

## Listado alfabético de las funciones predefinidas

- El intérprete de Python tiene una serie de funciones integradas que siempre están disponibles.

Built-in Functions			
<b>A</b> abs() aiter() all() any() anext() ascii()	<b>E</b> enumerate() eval() exec()	<b>L</b> len() list() locals()	<b>R</b> range() repr() reversed() round()
<b>B</b> bin() bool() breakpoint() bytearray() bytes()	<b>F</b> filter() float() format() frozenset()	<b>M</b> map() max() memoryview() min()	<b>S</b> set() setattr() slice() sorted() staticmethod()
<b>C</b> callable() chr() classmethod() compile() complex()	<b>G</b> getattr() globals()	<b>N</b> next()	<b>T</b> tuple() type()
<b>D</b> delattr() dict() dir() divmod()	<b>H</b> hasattr() hash() help() hex()	<b>O</b> object() oct() open() ord()	<b>V</b> vars()
	<b>I</b> id() input() int() isinstance() issubclass() iter()	<b>P</b> pow() print() property()	<b>Z</b> zip()  __import__()



## Funciones predefinidas

- Algunas de las más usadas son:
  - De entrada y salida: `print`, `input`
  - Numéricas: `abs`, `round`
  - Conversión de tipos: `float`, `int`, `str`
  - Codificación de caracteres: `ord`, `chr`

## Funciones de entrada y salida de datos (I)

- Los programas necesitan habitualmente ser ejecutados con datos introducidos por teclado.
- De la misma manera, los resultados son presentados en la pantalla con un determinado formato.
- Para ello, se utilizan las instrucciones de entrada y salida respectivamente.
- Las funciones de entrada y salida más elementales son: `input()` y `print()`, respectivamente.
- **Ejemplo:**

# Ejemplo de entrada y salida

# Cálculo del área de un cuadrado

`dato = input()` # la función input asigna a una cadena

`lado = float(dato)` # la cadena se convierte en número real

`area = lado * lado`

`print(area)`

## Funciones de entrada y salida de datos (II)

- Al ejecutar el anterior programa la consola se queda bloqueada y no sale ningún mensaje que alerte que se le pide un dato en particular.
- La función `input()` admite un argumento opcional: una cadena con el texto que aparece en pantalla para que el usuario sepa qué introducir.
- De igual manera, `print()` acepta múltiples argumentos, que convertirá a cadenas de caracteres y concatenará, separándolos por espacios.
- Tiene el argumento `sep` para indicar que separador se utiliza cuando el print incluye una coma. Por defecto el separador es un espacio.
- **Ejemplo:**

```
dato = input('Introduce el lado del cuadrado: ')
lado = float(dato) # la cadena se convierte en número real
area = lado * lado
print('El área del cuadrado es: ',area)
```

**Resultado de una ejecución:**

Introduce el lado del cuadrado: 5

El área del cuadrado es: 25.0

# Funciones numéricas

- **abs(n)**

- Devuelve el valor absoluto de un número.
- El argumento puede ser un entero o un número de punto flotante.
- Si el argumento es un número complejo, se devuelve su magnitud.

- Ejemplos:

```
>>> abs(-20.6)
20.6
>>> abs(4+3j)
5.0
```

- **round(n, ndig)**

- Devuelve el número real 'n' redondeado a los dígitos indicados en 'ndig'.
- Si se omite 'ndig', el valor predeterminado es cero. El resultado es un número real.

- Ejemplos:

```
>>> round(34.9645)
35
>>> round(34.9645,3) # el segundo argumento es el n. de decimales
34.965
```

## Funciones de conversión de tipo (I)

- **int(n)**

- Devuelve un entero construido a partir de un número o cadena 'n', o devuelve 0 si no se proporciona un argumento.

- Ejemplos:

```
>>> int(-7.9)
```

```
-7
```

```
int('23') # admite una cadena de caracteres entera (sin decimales)
```

```
23
```

- **float(n)**

- Devuelve un número de punto flotante construido a partir de un número o cadena 'n'.
- Si el argumento es una cadena, debe contener un número de punto flotante o decimal.
- Si no se especifica ningún argumento, devuelve 0.0.

- Ejemplos:

```
>>> float(21)
```

```
21.0
```

```
>>> float('-32.5') # admite una cadena de caracteres numérica
```

```
-32.5
```

## Funciones de cadenas

- `str(n)`

- Convierte un número en una cadena literal imprimible.

- Ejemplos:

```
>>> str(8.50)
```

```
'8.5'
```

```
>>> str(1.5E2)
```

```
'150.0'
```

- `len(s)`

- Devuelve el número de caracteres que contiene la cadena.

- Ejemplos:

```
>>> len('Esto es una cadena')
```

```
18
```

# Funciones de codificación de caracteres

- **ord(c)**

- Devuelve un entero que representa el código Unicode del carácter 'c'.

- Ejemplos:

```
>>> ord('A')
```

```
65
```

```
>>> ord('a') # el código del carácter 'A' es inferior al de 'a'
```

```
97
```

- **chr(n)**

- Devuelve una cadena de un carácter cuyo código Unicode es el entero n.

- Ejemplos:

```
>>> chr(97)
```

```
'a'
```

```
chr(241)
```

```
'ñ'
```

```
chr(8364) # código Unicode del euro
```

```
'€'
```

## Métodos del tipo de datos string

### Minúsculas, mayúsculas y título:

```
>>> cadena='Esto es un ejemplo'
>>> cadena.lower()
'esto es un ejemplo'
>>> cadena.upper()
'ESTO ES UN EJEMPLO'
>>> cadena.title()
'Esto Es Un Ejemplo'
```

### Contenido de texto:

```
>>> nombre='Mario'
>>> nif='32897654X'
>>> telefono='986543123'
>>> precio='3.45'
>>> vacio='   '
```

```
>>> nombre.isalpha()
True
>>> nif.isdigit()
False
>>> telefono.isdigit()
True
>>> nif.isalnum()
True
>>> precio.isdigit()
False
>>> vacio.isspace()
True
```



## Métodos del tipo de datos string

### Encontrar (find):

Localiza una subcadena dentro de una cadena de caracteres. Si no está devuelve -1. Si lo encuentra devuelve la posición.

```
>>> cadena='Esto es un ejemplo'
>>> cadena.find('ejemplo')
11
```

### Reemplazar (replace):

Permite reemplazar una cadena por otra. Devuelve una nueva cadena con el texto reemplazado.

```
>>> cadena.replace(' ', '_')
'Esto_es_un_ejemplo'
```

## Métodos del tipo de datos string

### Lista de palabras (split):

Trocea una cadena utilizando espacio u otro carácter.

```
>>> cadena='Esta cadena tiene varias palabras'
>>> cadena.split()
['Esta', 'cadena', 'tiene', 'varias', 'palabras']
```

### Eliminación de espacios (strip, lstrip y rstrip):

Borra espacios antes y después de la cadena.

```
>>> cadena='    Esta cadena tienes sobras    '
>>> cadena.strip()
'Esta cadena tienes sobras'
>>> cadena.lstrip()
'Esta cadena tienes sobras    '
>>> cadena.rstrip()
'    Esta cadena tienes sobras'
```

## Métodos del tipo de datos string

### Formato (format):

- El método `format()` permite utilizar una cadena como plantilla y sustituir valores para formar una cadena formateada.
- Cada uno de los argumentos de sustitución, llamados marcas de formato, se especifica como `{n}`, siendo `n` el orden en la lista de argumentos. Si no se pone número, sólo `{}`, se supone que el orden es secuencial (`{0},{1}...`).

- **Ejemplo:**

```
>>> cadena = 'La cadena tiene un valor numérico: {0}'
```

```
>>> cadena.format(0.12)
```

```
'La cadena tiene un valor numérico: 0.12'
```

```
# También se podría escribir sin declarar la variable cadena:
```

```
>>> 'La cadena tiene un valor numérico: {0}'.format(0.12)
```

```
'La cadena tiene un valor numérico: 0.12'
```

- Se puede interpolar más de un valor con una sola llamada a `format` :

```
>>> 'Sustituimos dos argumentos: {0} y {1}'.format(12.5,'así')
```

```
'Sustituimos dos argumentos: 12.5 y así'
```

```
# En este caso, uno numérico y el otro cadena
```

# F-String

## F o f:

- Se utiliza para dar formato a cadenas de texto de manera fluida. Puede indicarse con F o f.
- Se siguen utilizando las llaves {} para indicar posición.

- **Ejemplo:**

```
>>> marca = 'DELL'
>>> numProce = 3
>>> precio = 1720.99
>>> print(f 'Equipo {marca} con {numProce} procesadores y precio de salida de {precio}€)
Equipo DELL con 3 procesadores y precio de salida de 1720.99 €
```

```
>>> x = 3
>>> y = 9
>>> marca = 'DELL'
>>> print(F'El resultado es {x*y} ')
El resultado es 27
```

## Módulos

- Un **Módulo** es un fichero que contiene código escrito en Python, básicamente variables y funciones.
- La librería estándar de Python incluye por defecto más de 300 módulos. <https://docs.python.org/3/library/>
- Un **Paquete** es un directorio que agrupa un conjunto de módulos, normalmente relacionados con alguna temática.
- Gran parte del éxito de Python se debe a la existencia de más de 10.000 paquetes de código abierto que permiten resolver todo tipo de problemáticas.
- Cuando se programa en Python lo normal es aprovechar módulos y paquetes existentes, evitando realizar código ya existente.
- Python nos permite definir nuestros propios módulos y paquetes, pero de momento aprenderemos a utilizar los que ya existen.

## Módulos e importación de funciones y variables (I)

- Para utilizar las variables y funciones de un módulo, primero hay que importar el módulo.
- Por ejemplo, `math` es el módulo estándar de funciones matemáticas.
- Para importar y utilizar la función seno:

```
>>> from math import sin
```

```
>>> sin(0)
```

```
0.0
```

```
>>> sin(3.141592/2) # el argumento se especifica en radianes
```

```
0.999999999999999466
```

- Se pueden importar más de una función, separando cada función con una coma.
- Ejemplo:  

```
>>> from math import sin,cos
```

## Módulos e importación de funciones y variables (II)

- Si se utiliza un asterisco, se importan todos los elementos proporcionados por un módulo.
- Así, para importar todos los elementos del módulo `math`:  

```
>>> from math import *
```
- Esta posibilidad no resulta muy aconsejable porque:
  - Al importar elemento a elemento, el programa gana en legibilidad, pues se sabe de dónde proviene cada identificador.
  - Si se define una variable con un nombre determinado y dicho nombre coincide con el de una función definida en un módulo, la variable será sustituida por la función. Si no se conocen todos los elementos que define un módulo, es posible que esta coincidencia de nombre tenga lugar, pase inadvertida inicialmente, y cuando se quieran usar simultáneamente entren en conflicto.

## Módulos e importación de funciones y variables (III)

- Ejemplo:

```
>>> factorial = 2    # asigno factorial a un entero
>>> from math import *
>>> factorial(2)      # llamada a la función factorial del módulo
2
>>> factorial(3)
6
>>> factorial = 2    # asigno factorial de nuevo a un entero
>>> factorial(3)      # intento llamar a la función factorial
                        # pero la variable oculta a la función por ser
                        # la que se declaró en último lugar
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'int' object is not callable
>>> factorial # visualiza factorial que está asignado a un entero
2
```



## Módulos e importación de funciones y variables (IV)

- Existe una manera de evitar estas colisiones en los nombres de identificadores, indicando solo el nombre del módulo (sin asterisco):  
`>>> import math`
- Se puede referenciar la función o variable anteponiendo `math` y un punto.
- De esta manera pueden coexistir ambos identificadores.

- **Ejemplo:**

```
>>> import math
>>> factorial = 2    # asigno factorial a un entero
>>> math.factorial(3) # llama a la función factorial
6
>>> factorial # visualiza el valor de la variable factorial
2
>>> math.pi # visualiza el valor de la variable pi del módulo math
3.141592653589793
```

## Módulo math

Constantes y funciones matemáticas:

```
import math
math.pi           # número pi: 3.141592653589793
math.e            # número e: 2.718281828459045
math.sin(2)       # seno (radianes)
math.cos(2)       # coseno
math.tan(2)       # tangente
math.asin(0.5)    # arcoseno
math.acos(0.5)    # arcocoseno
math.atan(0.5)    # arcotangente
math.pow(2, 4)    # 2 elevado a 4
math.exp(4)       # e elevado a 4
math.sqrt(10)     # raíz cuadrada
math.pow(5, 1/3.0) # raíz cúbica de 5
math.log(3)       # ln; logaritmo natural
math.log(100, 10) # logaritmo base 10
math.ceil(2.3)    # techo (3)
math.floor(2.7)   # suelo (2)
```

## Módulo random

Números aleatorios:

```
import random
random.random()          # float aleatorio >= 0.0 y < 1.0
random.randint(0,10)     # int aleatorio >= 0 y <= 10
random.uniform(0.0,10.0) # float aleatorio >= 0.0 y < 10.0
random.gauss(5.0,1.0)    # float aleatorio media 5.0 y var 1.0
random.choice(('rojo','verde','azul')) # elemento aleatorio
```

# Informática para la ingeniería

## Tema 3



# Estructuras de control de flujo

- ❖ Bifurcación: `if...elif...else`.
- ❖ Excepciones: `try...except`.
- ❖ Bucles: `while...else`.

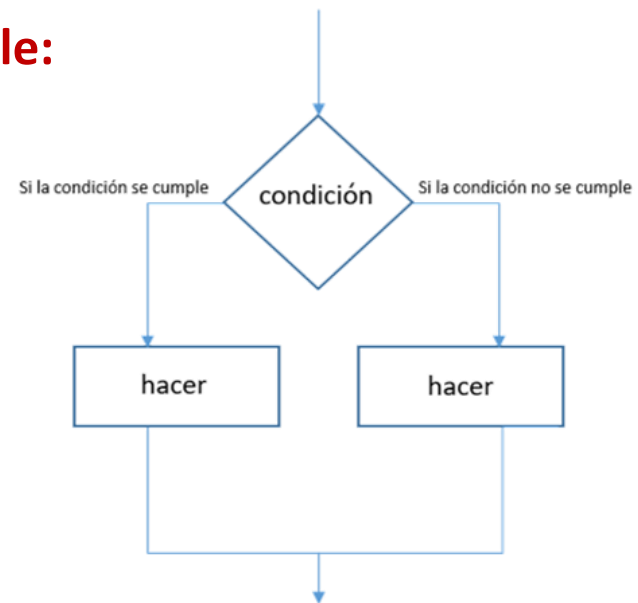
# Estructuras de control I. Estructuras de decisión o condicionales

Permiten alterar la secuencialidad en la línea de ejecución de un programa de instrucciones, bifurcando el camino, en dos o más distintos (en función de que se cumplan, o no, una o más condiciones).

## Sintaxis de una estructura condicional simple:

```
if condición:  
    hacer x  
else:  
    hacer y
```

```
media = 7  
if media < 5:  
    print("materia no superada")  
else:  
    print("materia superada")
```



Materia superada

## Estructuras de control I. Estructuras condicionales múltiples

Anidando estructuras condicionales simples, se forman estructuras condicionales múltiples.

### Sintaxis:

```
if condición 1:  
    hacer x  
else:  
    if condición 2:  
        hacer y  
    else:  
        hacer z
```

```
media = 7  
if media < 5:  
    print("Suspenso")  
else:  
    if media < 7:  
        print("Aprobado")  
    else:  
        if media < 9:  
            print("Notable")  
        else:  
            print("Sobresaliente")
```

Notable

## Estructuras de control I. Sentencia elif

Para la implementación de estructuras condicionales múltiples, puede utilizarse forma compacta elif.

### Sintaxis:

```
if condición 1:  
    hacer x  
elif condición 2:  
    hacer y  
else:  
    hacer z
```

```
media = 7  
if media < 5:  
    print("Suspenso")  
elif media < 7:  
    print("Aprobado")  
elif media < 9:  
    print("Notable")  
else:  
    print("Sobresaliente")
```

Notable



## Tablas de verdad

INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

INPUT	OUTPUT
A	NOT A
0	1
1	0

## Excepciones

Aunque la sentencia sea sintácticamente correcta, puede generar un error cuando se ejecuta. Estos errores se conocen como excepciones, algunas son: divisiones por cero, cálculos matemáticos inadecuados (raíces de valores negativos), operar con tipos incompatibles, etc. Para controlar las excepciones se utiliza **try - except**, que puede entenderse como «ejecuta estas sentencias y, si se comete un error, ejecuta estas otras».

### Algunas excepciones comunes

- **NameError**: no puede encontrar un nombre local o global (variable, función ,etc.).
- **TypeError**: a una función se le pasa un elemento del tipo inapropiado como su argumento.
- **ValueError**: el tipo es correcto pero un valor inapropiado.
- **ZeroDivisionError**: División por cero.
- **FileNotFoundError**: cuando el archivo o diccionario no existe.

# Excepciones

## Sintaxis:

**try:**

*acciones potencialmente erróneas 1..n*

**except:**

*acción para tratar el error 1*

*...*

*acción para tratar el error n*

**else:**

*acción realizada correctamente*

**[finally:**

*acción final]*

```
import math
```

```
try:
```

```
    resultado = math.factorial(2.4)
```

```
except:
```

```
    print("Ha ocurrido algo imprevisto.")
```

```
else:
```

```
    print(f"El factorial es {resultado}")
```

## Excepciones: secuencia

1. Se ejecuta el bloque *try*
  - Si **no ocurre ninguna excepción** dentro de él:
    - i. Los bloques *except* se ignoran.
    - ii. Se ejecuta el bloque *else*, en caso de que exista.
  - Si **ocurre una excepción** durante la ejecución del bloque *try*:
    - i. Se ignoran el resto de las instrucciones del bloque *try* y la ejecución salta a ejecutar *except*. Si hay un bloque *except* que referencia al error que ha ocurrido, ejecuta ese bloque (ver próxima diapositiva). También puede no ponerse ningún bloque. Sólo *except sin más* (ver diapositiva anterior).
2. Se ejecuta el bloque *finally* tanto si se capturó alguna excepción como si no. Incluso si en algún lugar previo se ejecutó un *return*.

## Excepciones

```
try:
    x=30
    y=int("a")
    result = x / y
except ZeroDivisionError:
    print("¡División por cero!")
except NameError:
    print("¡Variable no existe!")
except ValueError:
    print("¡Error en asignación!")
except:
    print("Error")
else:
    print(result)
finally:
    print("ejecutando la clausula finally")
```

## Estructuras de control. while

Repite una serie de instrucciones **mientras** se cumpla la condición (condición sea TRUE).

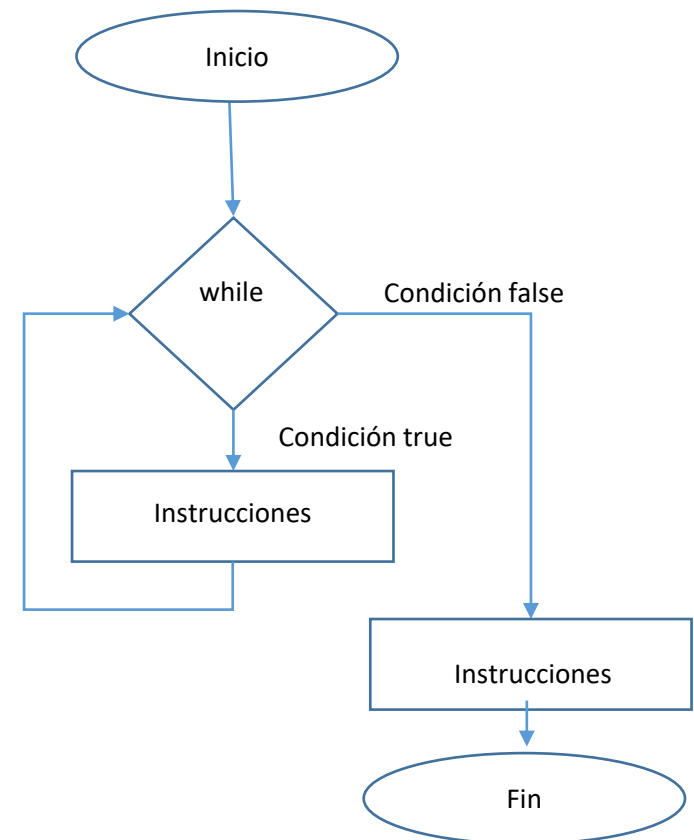
### Sintaxis:

**while** condición:

```
    instrucción_1  
    instrucción_2  
    .....
```

```
i = 1  
while i <= 3:           1  
    print(i)            2  
    i += 1              3  
print("Fin de programa") Fin de programa
```

```
a, b = 0, 1             1  
while b < 5:            1  
    print (b)          2  
    a, b = b, a + b    3
```



# Informática para la ingeniería

## Tema 4

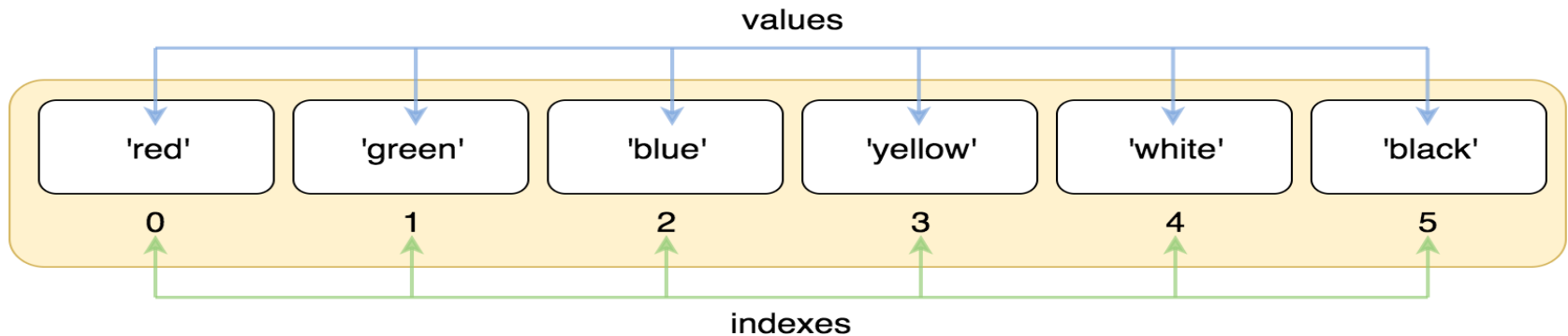


# Secuencias I. Estructuras control de flujo

- ❖ Secuencias: cadenas, listas y tuplas
- ❖ Operadores: +, \*, in
- ❖ Iteración: for...in.
- ❖ Uso de range y enumerate.



# Secuencias en Python



Algunas secuencias:

- ❖ Cadenas de caracteres.
- ❖ Listas
- ❖ Tuplas

## Operators

`x in s`  
`x not in s`  
`s + t`  
`s * n`  
`n * s`

## Slice

`s[i]`  
`s[i:j]`  
`s[i:j:k]`

## Methods

`s.index(x)`  
`s.count(x)`

## Built-in

`len(s)`  
`min(s)`  
`max(s)`

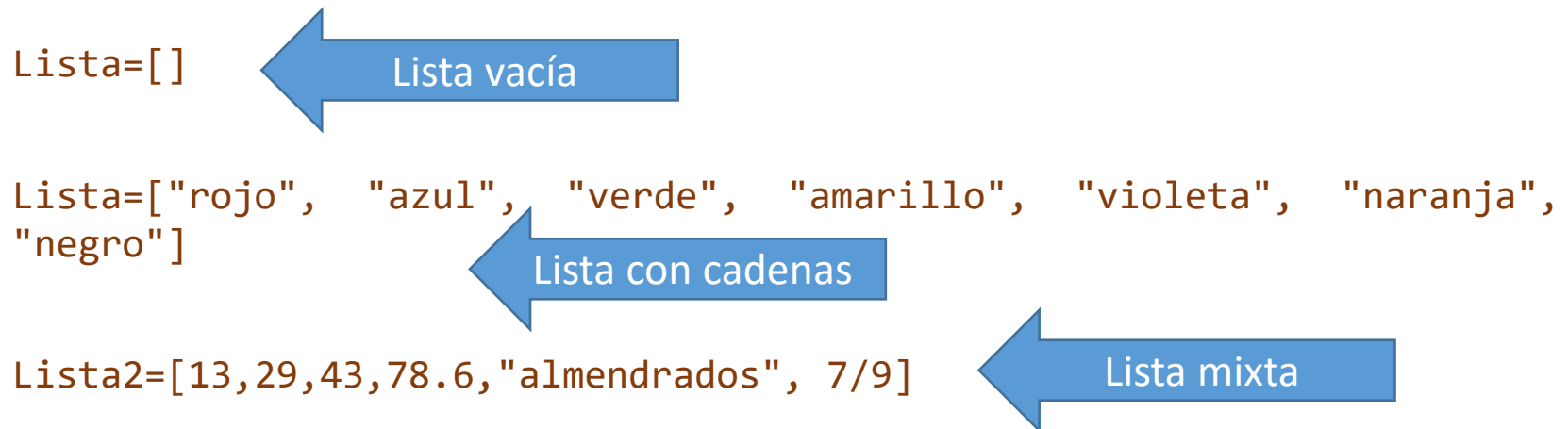
## Secuencias en Python

Es una estructura, serie o colección de datos de cualquier tipo donde es posible identificarlos y acceder a ellos de manera individual a través de su nombre y su índice que, es la posición que ocupa cada dato en esa serie. Es decir, se puede acceder a un elemento individual o un conjunto de elementos (slices). Algunas características:

- ❖ No existe límite en cuanto al número de elementos.
- ❖ Pueden contener cualquier tipo de dato, incluyendo otras secuencias (matrices). Cada elemento puede ser de un tipo diferente.
- ❖ No es necesario conocer a priori los elementos que la integran para poder construirlas.
- ❖ Permite que se vayan añadiendo elementos de manera progresiva.
- ❖ Se le pueden aplicar algunas funciones nativas de Python.
- ❖ Responde a algunos operadores.

## Lista

Es una secuencia **mutable** de valores de cualquier tipo de dato. Se ponen entre corchetes [] y se separan por comas (,). Corresponde al tipo de datos **list**.



*Observa que los números no llevan comillas y las palabras sí.*

## Tupla

Una tupla es una lista **inmutable**, de modo que no puede modificarse de ningún modo después de su creación. Para crearlas se utilizan paréntesis y sus elementos se separan por comas. Tipo de dato **tuple**.

Sus principales características son las siguientes:

- Los elementos de una tupla tienen un orden definido y su índice empieza en cero.
- Pueden tener valores repetidos:

```
ventas_semanales = (32,23,52,64,23)
```

- Pueden tener valores de distintos tipos:

```
datos_cliente = ("Paula", 666123123, 123333.34)
```

## Operadores secuencia

**Concatenación "+"**. Une dos secuencias cualesquiera del mismo tipo.

```
>>> lista_1=[5,7,8,10]
>>> lista_2=['a','b','l','t']
>>> lista_t=lista_1 +lista_2
>>> lista_t
[5, 7, 8, 10, 'a', 'b', 'l', 't']
```

Añadir un elemento a una secuencia.

```
>>> lista_1=[5,7,8,10]
>>> lista_t=lista_1 +[12]
>>> print(lista_t)
[5, 7, 8, 10, 12]
```

El operador genera una nueva secuencia sin modificar la original.

## Operadores secuencia

**Multiplicación "\*".** Repite un número dado de veces la secuencia.

```
lista=[5,6,7,8]
lista=lista * 4
print(lista)
[5, 6, 7, 8, 5, 6, 7, 8, 5, 6, 7, 8, 5, 6, 7, 8]
```

El operador genera una nueva secuencia sin modificar la original.

## Operadores secuencia

**"in".** Comprueba si un elemento está dentro de una secuencia.

```
>>> lista_1=[5,7,8,10]
>>> if 7 in lista_1:
>>>     print("El 7 está contenido en la secuencia")
>>> else:
>>>     print("El 7 NO está contenido en la secuencia")
```

El 7 está contenido en la secuencia

```
>>> lista=[5,6,7,8]
>>> if not 5 in lista:
>>>     print("El elemento no está")
>>> else:
>>>     print("Está contenido")
```

Está contenido

## Comprobar elementos de la secuencia

Las secuencias vacías dan como resultado `False` y si tienen algún contenido equivalen a `True`, de esta forma se pueden usar en sentencias como los *if*.

Para comprobar si nuestra secuencia está vacía podemos escribir la instrucción:

```
if lista:
    print("secuencia con contenido")
else:
    print(" secuencia vacía")
```

## Borrado de secuencia

Para borrar una secuencia se usa el operador *del*:

`del lista`

`del tupla`



## Iteración sobre secuencias. for...in

Itera sobre los elementos de cualquier secuencia (cadena de texto, listas, etc.), en el mismo orden en que aparecen en la secuencia. Puede leerse como «para todo elemento de una secuencia, hacer. .»

### Sintaxis:

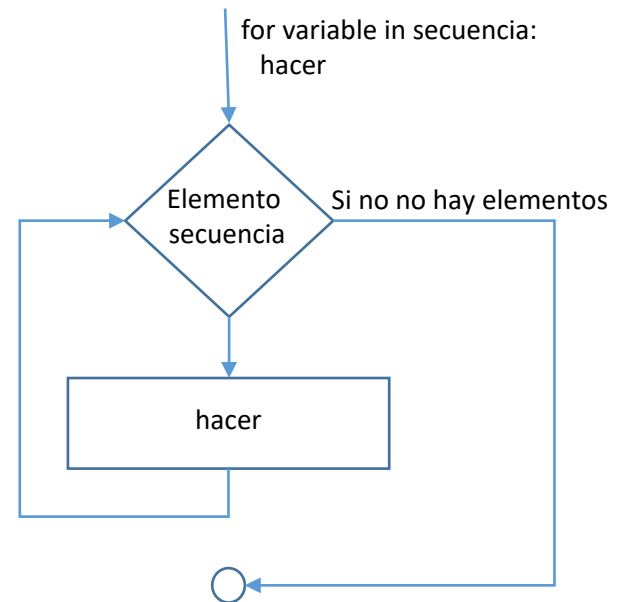
```
for variable in secuencia:  
    hacer 1  
    hacer 2  
    hacer n
```

```
for i in [2,4,6]:  
    print(i)
```

→ 2  
4  
6

```
for i in ["adiós","como","tal"]:  
    print(str(i)+" tiene "+str(len(i)))
```

→ adiós tiene 5  
como tiene 4  
tal tiene 3



## Iteración sobre secuencias. for...in

Pueden anidarse:

```
for i in [10,20]:  
    for j in [2,4]:  
        print(i*j)
```

20  
40  
40  
80

## Break

La sentencia **break** permite abortar o romper la ejecución del bucle **for** desde cualquier punto del mismo.

## Continue

Omite la ejecución de las sentencias desde que aparece en el bucle hasta la siguiente iteración del mismo.

## Iteración sobre secuencias. Función range

La función ***range*** devuelve un elemento de tipo *range*, que es un generador de secuencia de valores.

### Sintaxis:

**range**(valorFinal)

**range**(valorInicial, valorFinal)

**range**(valorInicial, valorFinal, incremento)

**range**(5) → [0,1,2,3,4]

**range**(1,5) → [1,2,3,4]

**range**(0,5,2) → [0,2,4]

## Iteración sobre secuencias. for...in enumerate

**enumerate**, es una función que permite obtener el índice junto a su valor correspondiente.

### Sintaxis:

**For indice, valor in enumerate(variable):**

instruccion\_1

instrucción\_2

.....

```
texto_introducido='ejemplo'  
for i,letra in enumerate(texto_introducido):  
    print (i, letra)
```

```
0 e  
1 j  
2 e  
3 m  
4 p  
5 l  
6 o
```

# Informática para la ingeniería

## Tema 5

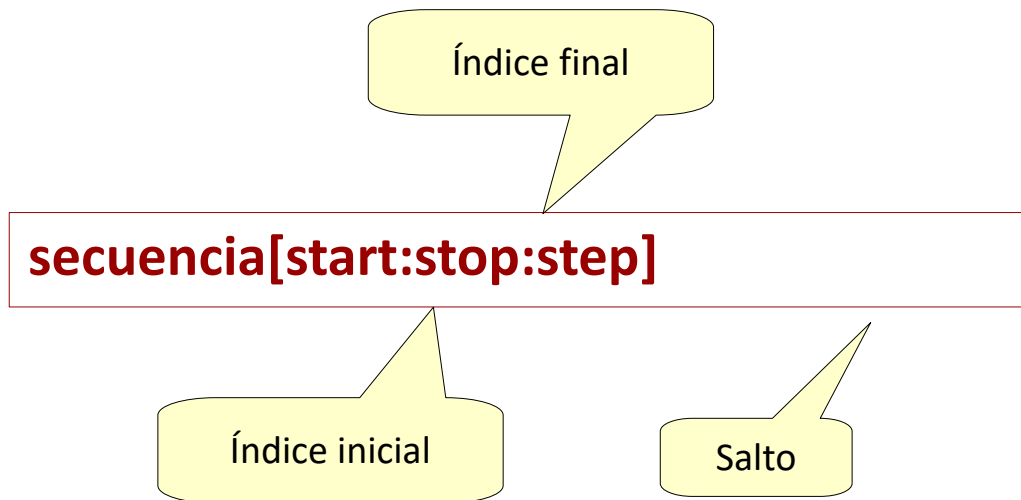


# Secuencias II

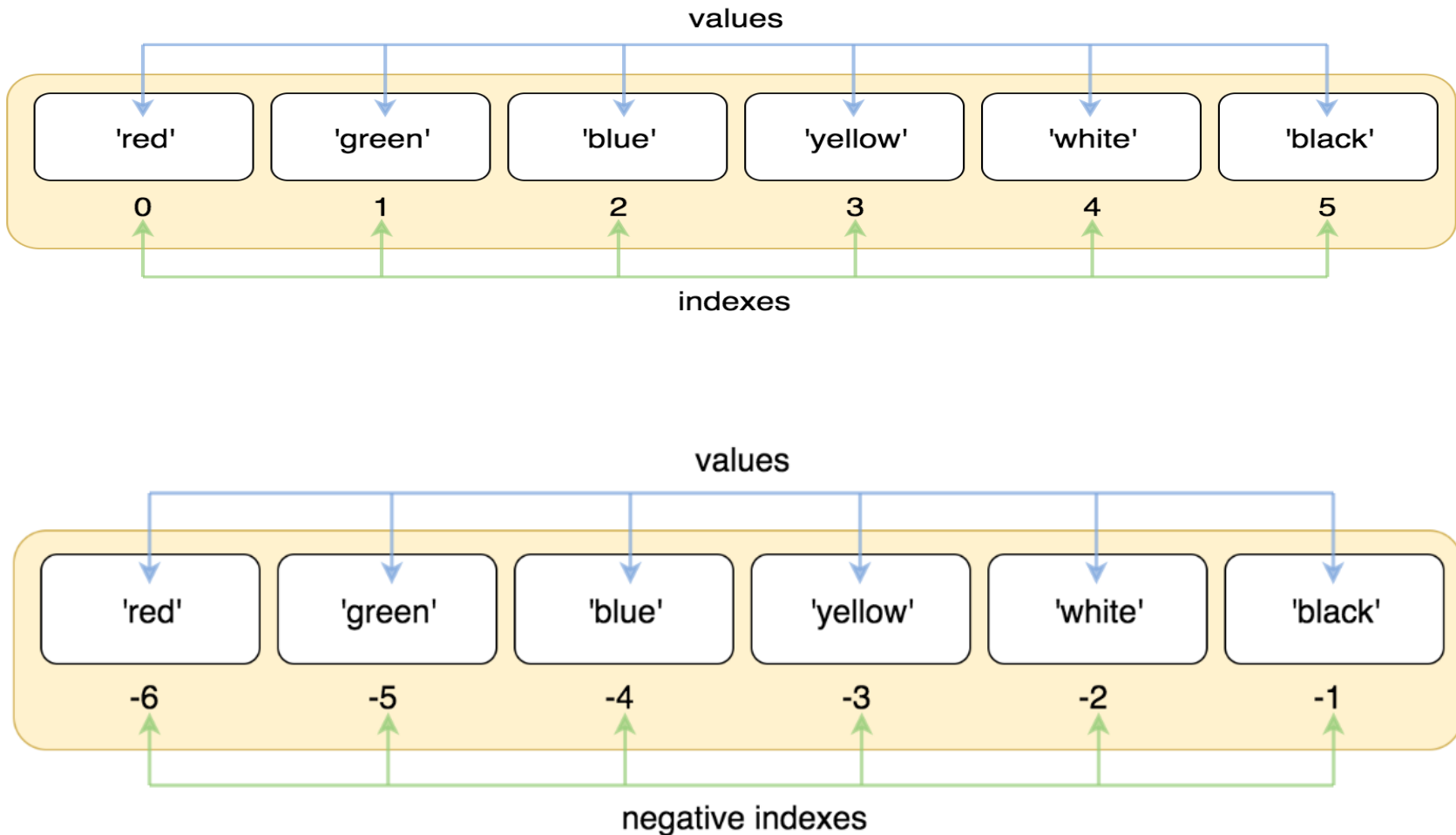
- ❖ Slicing.
- ❖ Concepto de valor y referencia.
- ❖ Métodos de secuencias: append, insert, remove, index,...
- ❖ Funciones de secuencias: len, sum, max, min,...
- ❖ Resumen métodos listas

## Vistas o segmentos (slicing)

Se puede obtener una parte de la secuencia original usando `secuencia[start:stop:step]`, con `start`, `stop`, `step` enteros. Lo anterior devuelve una nueva secuencia, del mismo tipo que la original (un segmento de una lista es una lista, etc.). La nueva secuencia contiene los elementos desde `secuencia[start]` hasta `secuencia[stop-1]` (no incluye a `secuencia[stop]`), salteándose `step` elementos cada vez. Los valores por defecto son `start=0`, `stop=último índice` y `step=1`.



## Vistas o segmentos (slicing)





# Vistas o segmentos (slicing)

```
lista=['a','b','c','d','e','f']
```

```
# Dos elementos centrales
```

```
print(lista[2:4])
```

```
# Todo menos dos últimos elementos
```

```
print(lista[:-2])
```

```
# Dos últimos elementos
```

```
print(lista[-2:])
```

```
# Elementos en posición impar menos el último
```

```
print(lista[1:-1:2])
```

```
# Lista invertida
```

```
print(lista[::-1])
```

```
# Toda la lista
```

```
print(lista)
```

```
print(lista[:])
```

```
print(lista[::])
```

```
['c', 'd']
```

```
['a', 'b', 'c', 'd']
```

```
['e', 'f']
```

```
['b', 'd']
```

```
['f', 'e', 'd', 'c', 'b', 'a']
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

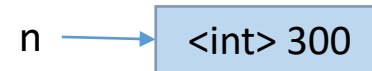
```
['a', 'b', 'c', 'd', 'e', 'f']
```

## Conceptos de referencia y valor

Todas las variables tienen una zona de memoria donde se almacenan sus datos.

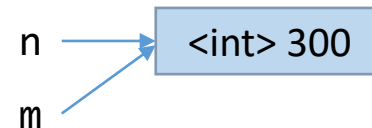
```
>>> n=300 # crea una zona de memoria para almacenar un tipo de dato int y almacena en ella el valor 300
```

La variable `n` no es más que un nombre que **referencia** a una zona de memoria donde se puede almacenar un tipo de dato *int*, y que en este caso contiene un **valor** de 300.



Si asigno `n` a otra variable, no se crea una nueva zona de memoria, sino una nueva referencia a la misma zona de memoria:

```
>>> m=n
```



## Conceptos de referencia y valor

Las listas no son inmutables y nos ofrecen muchas maneras de cambiar sus valores.

```
>>> notas=[5.4,7.8,6.2,8.9]
>>> revision=notas
```



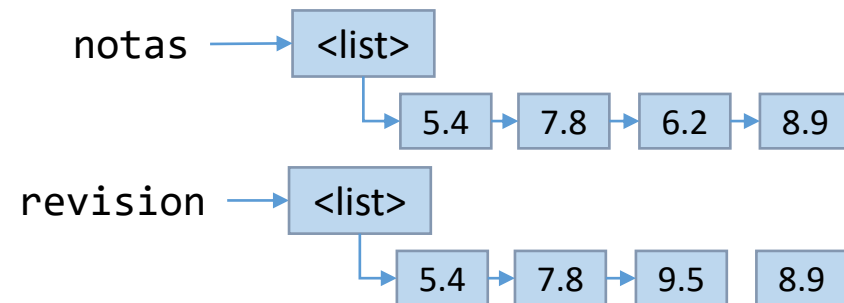
Las variables revision y notas apuntan a la misma zona de memoria, de manera que si hago un cambio en uno de los elementos de la lista, afecta a ambas.

```
>>> revision[2]=9.5
>>> print(notas,revision)
[5.4, 7.8, 9.5, 8.9] [5.4, 7.8, 9.5, 8.9]
```



Si realmente quiero copiar una lista, debo usar el método “copy”.

```
>>> notas=[5.4,7.8,6.2,8.9]
>>> revision=notas.copy()
>>> revision[2]=9.5
>>> print(notas,revision)
[5.4, 7.8, 6.2, 8.9] [5.4, 7.8, 9.5, 8.9]
```



## Funciones para secuencias

**len(secuencia).** Devuelve el número de elemento de la secuencia.

**min(secuencia).** Devuelve el valor menor de la secuencia.

**max(secuencia).** Devuelve el valor mayor de la secuencia.

```
>>> lista_1=[5,7,8,10]
```

```
>>> len(lista_1)           4
```

```
>>> min(lista_1)          5
```

```
>>> max(lista_1)          10
```

## Funciones para secuencias

### **sum(secuencia)**

- devuelve la suma de todos los elementos de la secuencia.

### **sorted(secuencia)**

- devuelve una lista con los valores de la secuencia ordenados.

```
valores = [2,1,3]  
print(sum(valores)) # muestra: 6  
print(sorted(valores)) # muestra: [1, 2, 3]
```

## Métodos para secuencia

**secuencia.count(x).** Cuenta el número de veces que está el elemento en la secuencia.

**secuencia.index(x).** Devuelve la posición del elemento en la secuencia, si el elemento no existe genera error.

```
>>> lista=(5,6,7,8)
```

```
>>> lista.index(8) 3
```

```
>>> lista_1=[5,7,8,10,7,3,4]
```

```
>>> lista_1.count(7) 2
```

## Resumen métodos listas

<code>list.append(x)</code>	Agrega un ítem al final de la lista. Equivale a <code>a[len(a):] = [x]</code> .
<code>list.extend(iterable)</code>	Extiende la lista agregándole todos los ítems del iterable. Equivale a <code>a[len(a):] = iterable</code> .
<code>list.insert(i, x)</code>	Inserta un ítem en una posición dada. El primer argumento es el índice del ítem delante del cual se insertará, por lo tanto, <code>a.insert(0, x)</code> inserta al principio de la lista, y <code>a.insert(len(a), x)</code> equivale a <code>a.append(x)</code> .
<code>list.remove(x)</code>	Quita el primer ítem de la lista cuyo valor sea <code>x</code> . Es un error si no existe tal ítem.
<code>list.pop([i])</code>	Quita el ítem en la posición dada de la lista, y lo devuelve. Si no se especifica un índice, <code>a.pop()</code> quita y devuelve el último ítem de la lista.
<code>list.clear()</code>	Quita todos los elementos de la lista. Equivalente a <code>del a[:]</code> .

## Resumen métodos listas

<code>list.reverse()</code>	Invierte los elementos de la lista in situ.
<code>list.copy()</code>	Devuelve una copia superficial de la lista. Equivalente a <code>a[:]</code> .
<code>list.index(x[, start[, end]])</code>	Devuelve un índice basado en cero en la lista del primer ítem cuyo valor sea x.
<code>list.count(x)</code>	Devuelve el número de veces que x aparece en la lista.
<code>list.sort(key=None, reverse=False)</code>	Ordena los ítems de la lista in situ.

<http://docs.python.org.ar/tutorial/3/datastructures.html>



# Informática para la ingeniería

## Tema 6



# Matrices. Secuencia III

- ❖ Secuencias de múltiples dimensiones: matrices.
- ❖ Iteración de secuencias de múltiples dimensiones.

## Definición de matriz

Las matrices son disposiciones multidimensionales de valores, organizados en un número arbitrario de dimensiones.

La siguiente matriz tiene 4 filas y 3 columnas (4x3):  $m = \begin{pmatrix} 1 & 3 & 5 \\ 7 & 8 & 0 \\ 5 & 6 & 9 \\ 0 & 0 & 1 \end{pmatrix}$

## Construcción

En Python las matrices se pueden construir como listas de listas:

```
m0 = []          # matriz vacía
```

```
m1 = [[1, 3, 5], [7, 8, 0], [5, 6, 9], [0, 0, 1]]
```

A continuación vemos cómo definir una matriz 3x2x2:

```
m2 = [[[3, 2], [4, 2]], [[7, 2], [3, 6]], [[6, 8], [6, 7]]]
```

Este esquema se puede generalizar para definir matrices de cualquier dimensión.

Los elementos de una matriz pueden ser de tipos distintos.

## Matrices irregulares

Es posible crear matrices donde por ejemplo, cada fila tiene un número distinto de elementos:

```
m3 = [[3, 5], [4, 3, 2]] # 1ª fila: 2 elementos. 2ª: 3 elem.
```

## Acceso

Para acceder a un elemento se indica su posición:

```
print (m3[0][1])    # muestra 5
print (m3[0])        # muestra [3, 5]
m3[1][2] = 0         # pone a 0 el último elemento de la última fila
```

## Función Len

La función *len* indica cuántos elementos tiene una dimensión determinada de la matriz.

```
print (len(m3))      # muestra el nº de filas (1ª dimensión): 2
print (len(m3[1]))    # muestra el nº de elementos de la 2ª fila: 3
```

## Añadir elementos con `append`

La función *append* de las listas añade elementos al final:

```
cliente1 = ["María", 666333111, 123565.09]
cliente2 = ["Pedro", 666000111, 109832.22]
clientes = [] # matriz vacía
clientes.append(cliente1) # añade cliente1 como 1ª fila
clientes.append(cliente2) # añade cliente2 como 2ª fila
clientes.append(["Laura", 678000777, 293302.31]) # 3ª fila
print(clientes)
```

Para añadir una fila con el mismo elemento en todas las columnas, usamos *append* con el operador `*` para indicar el número de elementos:

```
clientes.append([0]*3) # añade fila de 3 columnas con valor 0
```

## Añadir elementos con insert

La función *insert* de las listas permite añadir nuevos elementos en posiciones concretas (*append* sólo permitía añadir al final):

```
cliente1 = ["María", 666333111, 123565.09]
cliente2 = ["Pedro", 666000111, 109832.22]
clientes = []                                # matriz vacía
clientes.append(cliente1) # añade cliente1 como 1ª fila
clientes.append(["Laura", 678000777, 293302.31]) # 2ª
fila clientes.insert(1, cliente2) # añade cliente2 como 2ª
fila print(clientes)
```

La función *insert* no elimina ninguna de las filas existentes, sino que inserta la nueva en la posición *x* solicitada, desplazando en una posición la que estaba hasta ese momento en esa posición y todas las siguientes

## Eliminar elementos con pop

La función *pop* elimina y devuelve la fila indicada por medio de su único argumento:

```
m = [[1, 2], [3, 3], [4, 5]]
eliminada = m.pop(1) # elimina la 2ª fila
print(m)             # muestra: [[1, 2], [4, 5]]
print(eliminada)     # muestra: [3, 3]
```

Si se quisiera eliminar una columna, habría que eliminar el elemento correspondiente en cada fila:

```
m = [[1, 2], [3, 3], [4, 5]]
for i in range(len(m)):
    m[i].pop(0) # elimina el primer elemento de la fila i
print(m)       # muestra: [[2], [3], [5]]
```

## Recorrido con for

Se puede recorrer cualquier matriz usando tantos bucles for anidados como dimensiones tiene:

```
for i in range(len(m1)):          # recorre filas
    for j in range(len(m1[i])):    # recorre columnas
        m1[i][j] = int(input(f"[{i}-{j}] :"))
```

- *len(m1)* devuelve cuántos elementos tiene la 1ª dimensión (filas), y *len(m1(i))* cuántos en la 2ª dimensión del elemento i (columnas).
- El orden de los bucles del ejemplo recorre la matriz por filas:  
 $m1[0,0] \rightarrow m1[0,1] \rightarrow m1[0,2] \rightarrow m1[1,0] \rightarrow m1[1,1] \rightarrow m1[1,2] \dots$
- Para recorrer por columnas en lugar de por filas tan solo habría que intercambiar la posición de los bucles.
- Para recorrer en orden decreciente se reescribe la función range:  

```
for i in range(len(m1)-1, -1, -1) # desde len(m1) hasta 0
```



## Ejemplo final: trasposición de una matriz

```
m = []                                # crea la matriz vacía
n = int(input("dimensión: ")) # lee dimensión (nº filas = nº cols)

# lectura del contenido de la matriz:
for i in range(n):
    m.append([0] * n)                # añade fila por fila con ceros
    for j in range(n):                # recorre los elementos de la fila
        m[i][j] = int(input "["+str(i)+", "+str(j)+"]: ")

# trasposición
for i in range(n):                    # recorre las filas
    for j in range(i):                # recorre columnas de la 0 a la i-1
        m[i][j], m[j][i] = m[j][i], m[i][j]
print(m)
```

# Informática para la ingeniería

## Tema 7



# Funciones y Módulos Propios

- ❖ Definición de funciones: nombre, parámetros y resultado.
- ❖ Paso de argumentos por valor o referencia.
- ❖ Ámbito de las variables.
- ❖ Definición de módulos propios.

## Definición de Funciones (propias)

Ya hemos aprendido a usar funciones existentes.

Python nos permite crear nuestras propias funciones.

### Sintaxis definición de una función:

```
def nombreFunción([parámetros]):  
    """ La función hace... """ # documentación función  
    bloque de instrucciones y definiciones  
    [return valor de retorno]
```

### Sintaxis de la llamada a la función:

```
[variable =] nombreFunción([argumentos])
```

## Definición de Funciones - Ejemplo

Nombre      Parámetros

```
def mayor(a,b):  
    if a>b:  
        return a  
    else:  
        return b
```

Definición función

Cuerpo función

Resultado      Argumentos

```
num_mayor=mayor(6,2)  
print(num_mayor)
```

Llamada función

6

The diagram illustrates the components of a Python function. The function definition 'def mayor(a,b):' is shown with 'def' labeled as the 'Nombre' (name) and '(a,b)' as the 'Parámetros' (parameters). The indented code block 'if a>b: return a; else: return b' is labeled as the 'Cuerpo función' (function body). The function call 'num\_mayor=mayor(6,2); print(num\_mayor)' is shown with 'num\_mayor' labeled as the 'Resultado' (result) and '(6,2)' as the 'Argumentos' (arguments). A bracket groups the definition and body, and another bracket groups the call and the result '6'.

```
def Saludos(nombre):  
    print("Buenos días", nombre)
```

Buenos días Juan ← Saludos("Juan")

## Definición de Funciones - Ejemplo

### Definición:

```
def mensajePremio(tratamiento, nombre, premio):  
    ''' Devuelve un mensaje formateado relativo a un premio  
        tratamiento y nombre son cadenas  
        premio es un float que representa un importe en €  
    ...  
  
    plantilla='Estimado {} {}, le han tocado {} €'  
    return(plantilla.format(tratamiento, nombre, round(premio,2)))
```

### Uso:

```
print(mensajePremio('Dr. ', 'Méndez', 234.4567))
```

Estimado Dr. Méndez, le han tocado 234.46 €

La función “esconde” la complejidad de formatear el texto.  
Una vez resuelto el problema, podemos reutilizar la solución.  
Si queremos hacer un cambio, solo lo hacemos en la función.

## Definición de Funciones - Generalidades

- La funcionalidad de cada función debe ser clara y sencilla.
- El objetivo es dividir la lógica de nuestro programa en bloques para reducir la complejidad y reutilizar código.
- Una función puede llamarse varias veces durante la ejecución de un programa.
- La función consta de un cuerpo, que no es más que todas las sentencias y definiciones que se necesitan para realizar la acción para la que se crea.
- En el cuerpo puede haber llamadas a otras funciones predefinidas (len, str, etc.).
- También puede haber llamadas a otras funciones definidas por el usuario.

## Definición de Funciones - Parámetros y variables

- Una función puede tener cero o más parámetros.
- Los parámetros permiten alterar el comportamiento de la función y pueden ser de cualquier tipo de datos.
- Dentro de la función los parámetros actúan como si fuesen variables, solo que nos ofrecen una copia de las variables o valores literales que fueron pasados a la función como argumentos.
- Dentro de las funciones pueden crearse variables, llamadas variables locales porque solo pueden usarse dentro de la misma.
- Las funciones tienen acceso a las variables globales, que son las definidas en el archivo fuera de las funciones.



## Definición de Funciones - Return

Marca el fin de la función. Puede ir acompañado del valor que se quiere regresar (que puede ser de cualquier tipo). Puede ir en solitario para indicar que la función no retorna nada, y es lo que se conoce como "procedimiento". E incluso puede omitirse al final de la función y se entiende, al igual que en el caso anterior que la función no regresa ningún valor o regresa **None**.

```
def mayor(a,b):  
    if a>b:  
        return b,a  
    else:  
        return a,b
```

```
a,b=mayor(10,5)
```



Return

```
def imprime(mensaje):  
    print(mensaje)  
  
imprime("Hola Mundo")
```



Sin Return

## Paso de argumentos por valor o referencia

Dependiendo del tipo del dato que pasemos como argumento a una función se realizará:

- Paso por valor: Se crea una copia local de la variable dentro de la función. Si modificamos la copia, la variable original mantiene su valor. Tipos de datos simples: enteros, cadenas...
- Paso por referencia: Se crea una copia local de la dirección de la variable dentro de la función. Si modificamos la copia, también estamos haciendo cambios sobre la variable original. Tipos de datos compuestos: listas,...



Una manera de devolver múltiples resultados es devolver un tipo de dato compuesto (lista, tupla...) en el return.

Otra manera es pasar un tipo de datos compuesto como argumento y que la función haga cambios sobre el mismo.


## Alcance de las variables

No todas las variables del programa son accesibles desde todos los lugares del mismo. Depende donde estén definidas:

- Locales: están definidas dentro del cuerpo de una función, y solo pueden ser accesibles dentro de esa función.
- Globales: definidas en el programa principal y son accesibles dentro de todo el programa incluidas las funciones.

```
total = 0                                      Global
def sum( num1, num2 ):
    total = num1 + num2                        Local
    print (f"Dentro el total es : {total}")
    return total
```

```
sum( 5, 10 )
print (f"Fuera el total es : {total}")
```

 Dentro el total es : 15  
Fuera el total es : 0

## Definición de Módulos (propios)

Ya hemos aprendido a usar módulos existentes.

Python nos permite crear nuestros propios módulos.

Un módulo no es más que un fichero .py y lo normal es que contenga instrucciones, funciones, etc. relacionadas alrededor de algún tipo de problemática.

Al importar un módulo, lo primero que ocurre es que se ejecuta todo el código contenido en el fichero, aunque no importe todos sus elementos, de manera que si asignamos valores a variables, éstas ocuparan espacio en memoria. Igualmente, si llamamos a *input* o *print* pedirá datos al usuario o mostrará información por la pantalla en el momento de ejecutar el *import*.

Por lo tanto, si lo que buscamos es la reutilización de funcionalidad, lo normal es que el código se limite a asignar valores a variables, definir funcione, etc.

A diferencia de las funciones, las variables de un módulo no tienen su propio espacio de memoria, de manera que cualquier cambio es global.

# Informática para la ingeniería

## Tema 8



# Persistencia

❖ Ficheros de texto

## Ficheros. Definición

En informática, un fichero o archivo es un grupo de datos estructurados que son almacenados en algún medio de almacenamiento persistente (disco duro, disco óptico, pendrive, etc.) y pueden ser usados por las aplicaciones.

La forma en que una computadora organiza, da nombre, almacena y manipula los archivos se denomina sistema de archivos y suele depender del sistema operativo y del medio de almacenamiento.

En Windows existen, principalmente, dos tipos de ficheros:

Ficheros ASCII o ficheros de texto. Contienen caracteres codificados según el código ASCII y se pueden leer con cualquier editor de texto. Suelen tener extensión \*.txt o \*.bat, pero también otras como \*.m (Matlab), \*.c (lenguaje C), \*.cpp (C++), \*.java (Java) o como en el caso de Python \*.py

Ficheros binarios: Almacenan la información tal como están en la memoria del ordenador. No son legibles directamente por el usuario. Ocupan menos espacio en disco y no se pierde tiempo cambiándolos a formato ASCII al escribirlos y al leerlos en el disco.

## Ficheros de texto

Los ficheros de texto son de acceso secuencial.

En un fichero de acceso secuencial se leen y escriben los datos del mismo modo que se hace en una antigua cinta de video. Si se quiere acceder a un dato que está hacia la mitad de un fichero, habrá que pasar primero por todos los datos anteriores.

El trabajo con ficheros obliga a seguir siempre un protocolo de tres pasos:

- Abrir el fichero indicando su ruta (relativa o absoluta) y el modo de trabajo. Hay varios modos de trabajo:
  - Lectura: es posible leer información del fichero, pero no modificarla ni añadir nueva información.
  - Escritura: solo es posible escribir información en el fichero. Sobreescribe el archivo si existe. Crea el archivo si no existe.
  - Adición: permite añadir nueva información al fichero, pero no modificar la ya existente. Crea el archivo si no existe.
- Leer o escribir la información que deseas.
- Cerrar el fichero.



## Ficheros de texto

Veamos las órdenes de Python para los pasos anteriores:

**fichero=open(nombredelfichero, modo)**

**nombredelfichero** es el nombre del fichero a abrir (Variable String o nombre del fichero entre comillas)

**modo** es el modo de apertura del fichero. Para indicar el modo usaremos:

- Lectura: 'r' (ejemplo: **fichero=open('c:/misdocumentos/ejemplo.txt', 'r')**)
- Escritura: 'w' (ejemplo: **fichero=open('c:/misdocumentos/ejemplo.txt', 'w')**)
- Adición: 'a' (ejemplo: **fichero=open('c:/misdocumentos/ejemplo.txt', 'a')**)

**fichero.close()**

Cierra el fichero. Si se quiere volver a utilizar, habría que volver a abrirlo.

Si no se cierra el fichero, en algunos casos podría producirse pérdida de información en fichero.

## Ficheros de texto. Lectura

**fichero.readline()** Lee una línea de un fichero de texto. Lee todos los caracteres hasta encontrar un salto de línea.

```
ficheroN="C:/Users/Documents/Prueba.txt"
fichero=open(ficheroN,"r")
linea=fichero.readline()
while linea!="":
    print(linea)
    a_lista=linea.split()
    print(a_lista)
    linea=fichero.readline()
fichero.close()
```

### Fichero Prueba.txt

```
lunes 12 123.87 alta a
martes 13 56.74 baja b
miercoles 14 101.12 media m
```

```
ficheroN="C:/Users/Documents/Prueba.txt"
fichero=open(ficheroN,"r")
for linea in fichero:
    print(linea)
    a_lista=linea.split()
    print(a_lista)
fichero.close()
```

```
lunes 12 123.87 alta a
['lunes', '12', '123.87', 'alta', 'a']
```

```
martes 13 56.74 baja b
['martes', '13', '56.74', 'baja', 'b']
```

```
miercoles 14 101.12 media m
['miercoles', '14', '101.12', 'media', 'm']
```

## Ficheros de texto. Lectura

`fichero.read(n)` Lee "n" caracteres de un fichero de texto. Si no se indica n se lee todo el contenido.

Fichero Prueba.txt

lunes 12 1  
23.87 alta  
a  
martes  
13 56.74 b  
aja b  
mier  
coles 14 1  
01.12 medi  
a m

lunes 12 123.87 alta a  
martes 13 56.74 baja b  
miercoles 14 101.12 media m

```
ficheroN="C:/Users/Documents/Prueba.txt"  
fichero=open(ficheroN,"r")  
linea=fichero.read(10)  
while linea!="":  
    print(linea)  
    linea=fichero.read(10)  
fichero.close()
```

## Ficheros de texto. Lectura.

Una de las formas más prácticas de leer un fichero de texto es usando el bucle *for* como se muestra en el ejemplo. En cada iteración, se almacena una línea nueva en la variable del bucle:

```
fichero = open("archivo.txt","r")  
for linea in fichero:  
    print(linea)  
fichero.close()
```

En cada iteración del bucle, variable *línea* recibe una nueva línea de texto, en el orden en el que se han escrito en el fichero.

No es necesario comprobar cuándo se llega al final del fichero, ya que el *for* termina cuando esto ocurre.

## Ficheros de texto

Método de escritura:

- `fichero.write(cadena)`

Escribe el contenido de la variable cadena en un fichero de texto

En los siguientes ejemplos la única diferencia es el modo de apertura del fichero.

En el primer ejemplo se abre en modo escritura y por tanto los datos se escriben desde el inicio del fichero.

En el segundo se abre en modo adición y por tanto los datos se escriben a continuación de los que ya haya en el fichero.

Si se quiere añadir un salto de línea se pone: “\n”.

```
fichero=open("c:/tmp/dígitos.txt",'w')
for i in range(0,10):
    fichero.write(str(i))
fichero.close()
```

```
fichero=open("c:/tmp/dígitos.txt",'a')
for i in range(0,10):
    fichero.write(str(i))
fichero.close()
```

# Informática para la Ingeniería

## Tema 9



# Interfaz Gráfica de Usuario (IGU)

- ❖ Introducción
- ❖ Elementos clave de una IGU
- ❖ Ventana
- ❖ *Widgets* comunes
- ❖ Administrador de diseño: paquete y cuadrícula
- ❖ Eventos
- ❖ Otros *widgets*

## Interfaz gráfica de usuario. Introducción

Una interfaz gráfica de usuario (IGU, o GUI por sus siglas en Inglés (*Graphical User Interface*)) provee una experiencia de usuario basada en un paradigma llamado WIMP (*windows, icons, menus, pointer*), esto es, (VIMP (ventanas, íconos, menús, puntero)).

Los sistemas operativos modernos dan preferencia a las IGU sobre las IBLC (Interfaces basadas en línea de comando) porque han demostrado que reducen la curva de aprendizaje de los usuarios.

Los sistemas operativos dan guías y paquetes que simplifican y estandarizan el desarrollo de las IGU.

El paquete estándar para la IGU en Python es **tkinter**.

Para utilizarlo, primero necesita importar el módulo:

```
import tkinter # accede a sus elementos usando tkinter.x
```

o

```
from tkinter import * # accede a sus elementos directamente
```



## Elementos clave de una IGU

Los elementos principales de una IGU son los siguientes:

- *Window* (Ventana): Superficie donde la IGU o ventana se dibujará.
- *Widgets*: Estas son las cosas con las que el usuario ve e interactúa: *buttons* (botones), *input fields* (campos de entrada), *images* (imágenes)...
- *Frames* (Marcos): Tipo especial de *widget* utilizado para contener otros *widgets* o marcos (*frames*) formando una estructura jerárquica.
- Administrador de diseño: Controla el diseño de los *widgets*, v.gr. cómo se posicionan o cambian su tamaño en base al espacio disponible.
- Comandos: Son funciones que se llamarán cuando se genera un evento de usuario, tal como pulsar un botón.

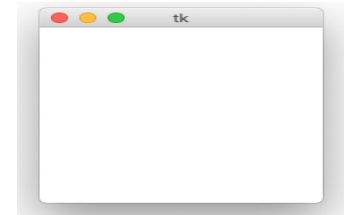
A las ventanas y los marcos se les llama “contenedores” (*containers*), porque son utilizados para colocar otros marcos (*frames*) o *widgets*.

Esto nos permite crear un árbol jerárquico de componentes.

# Ventana

Para crear una ventana, todo lo que tenemos que hacer es lo siguiente:

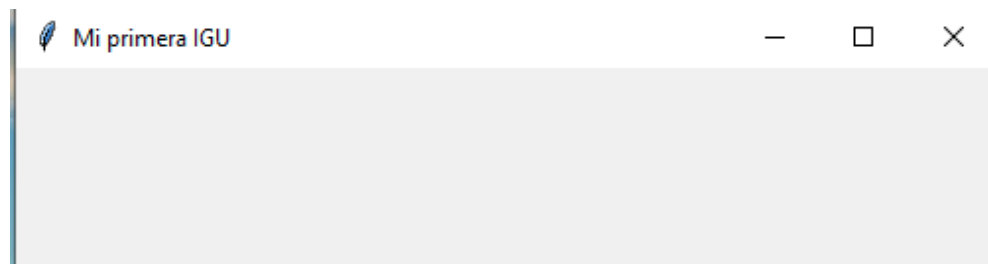
```
from tkinter import *  
ventana = Tk() # crea una ventana  
ventana.mainloop() # abre la ventana y  
                  # comienza la gestión de eventos
```



El elemento “ventana” representa una ventana.

Varios métodos permiten cambiar su apariencia:

```
from tkinter import *  
ventana = Tk() # crea una ventana  
ventana.geometry('500x100') # Tamaño de la ventana por defecto  
ventana.title('Mi primera IGU') # texto en la cabecera  
ventana.mainloop() # abre la ventana y comienza la gestión de  
eventos
```

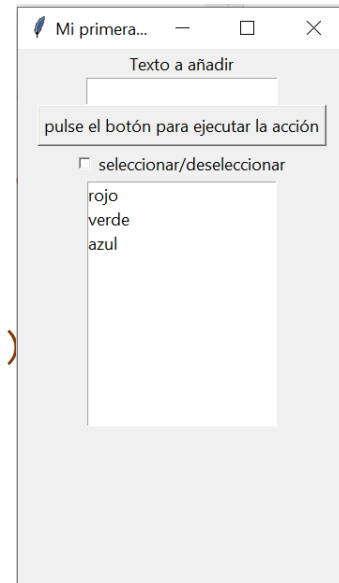


## Widgets comunes

Permite a los usuarios interactuar con la IGU:

- *Label* (Etiqueta): salida de texto
- *Entry* (Entrada): entrada de texto
- *Button* (Botón): pulsar para ejecutar una acción
- *Checkbutton* (Botón de verificación): Seleccionar/deseleccionar una opción
- *Listbox* (lista): visualiza varias líneas de texto

```
from tkinter import *
ventana = Tk() # crea una ventana
ventana.geometry('350x575') # tamaño por defecto de la ventana
ventana.title('Mi primera ventana') # texto en la cabecera
etiquetatexto = Label(ventana,text='Texto a añadir')
entradadetexto = Entry(ventana)
boton = Button(ventana,text='pulse el botón para ejecutar la acción')
seleccion = Checkbutton(ventana,text='seleccionar/deseleccionar')
lista = Listbox(ventana)
etiquetatexto.pack() # diseño (paquete) de la widget en al ventana
entradadetexto.pack()
boton.pack()
seleccion.pack()
lista.pack()
lista.insert(END,'rojo') # añade el texto “rojo” al final de la lista
lista.insert(END,'verde')
lista.insert(END,'azul')
ventana.mainloop() # abrir ventana y gestionar eventos desde tkinter
```



## Administración de diseño – paquete

Los *Widgets* pueden organizarse de manera relativa utilizando *pack*

Sí no se dan argumentos al paquete (*pack*) los componentes ocupan su tamaño por defecto y se organizan de arriba para abajo y centrados

Las opciones permiten poner los componentes sobre la izquierda, derecha o abajo, rellenar todo el espacio disponible, etc.

```
from tkinter import *

ventana = Tk()
ventana.geometry('200x300')
ventana.title('GUI pack')
etiqueta1 = Label(ventana,text='Arriba',bg='red') # los nombres
etiqueta2 = Label(ventana,text='Izquierda',bg='green') # de
etiqueta3 = Label(ventana,text='Derecha',bg='blue') # los colores
etiqueta4 = Label(ventana,text='Abajo',bg='yellow') # en inglés
etiqueta5 = Label(ventana,text='Centro',bg='orange')
etiqueta1.pack(fill=X)
etiqueta2.pack(side=LEFT,fill=Y)
etiqueta3.pack(side=RIGHT,fill=Y)
etiqueta4.pack(side=BOTTOM,fill=X)
etiqueta5.pack(side=TOP,fill=X)
ventana.mainloop()
```



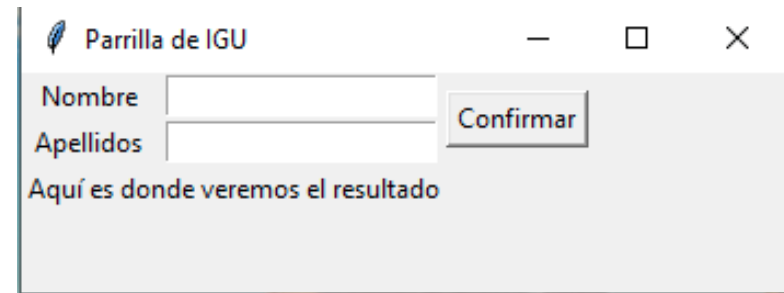
## Administración de diseño – parrilla (*grid*)

Para muchas aplicaciones, se prefiere el componente de diseño en parrilla

El método en parrilla permite ordenar los componentes por renglones y columnas

Las opciones *rowspan* y *columnspan* indican la ocupación múltiple de celdas

```
from tkinter import *
ventana = Tk()
ventana.geometry('450x200')
ventana.title('Parrilla de IGU')
etiquetanombre = Label(ventana, text='Nombre')
entradatextonombre = Entry(ventana)
etiquetaapellidos = Label(ventana, text='Apellidos')
entradatextoapellido = Entry(ventana)
botondeconfirmar = Button(ventana, text='Confirmar')
etiquetaresultado = Label(ventana, text='Aquí es donde veremos el resultado')
etiquetanombre.grid(row=0, column=0)
entradatextonombre.grid(row=0, column=1)
etiquetaapellidos.grid(row=1, column=0)
entradatextoapellido.grid(row=1, column=1)
botondeconfirmar.grid(row=0, column=2, rowspan=2)
etiquetaresultado.grid(row=2, column=0, columnspan=2)
ventana.mainloop()
```

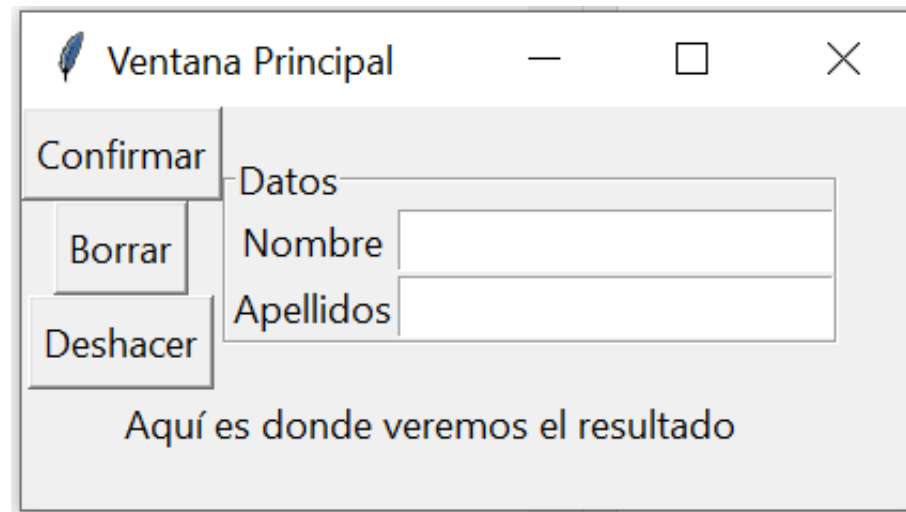


## Marcos (Frames)

Un marco (*Frame*) es un tipo especial de *widget* que puede contener otros *widgets*

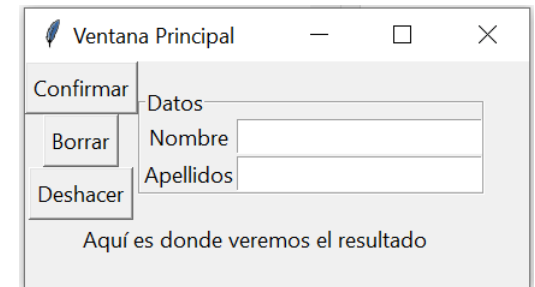
*LabelFrame* es un marco con una cabecera de texto y una caja rodeándolo

Cada marco tiene su propio diseño y otros marcos pueden anidarse (árbol)



## Marcos (Frames). ejemplo

```
from tkinter import *
ventana = Tk()
ventana.geometry('560x520')
ventana.title('Ventana Principal')
marcoizquierdo=Frame(ventana)
botonconfirmar = Button(marcoizquierdo,text='Confirmar')
botonborrar = Button(marcoizquierdo,text='Borrar')
botondeshacer = Button(marcoizquierdo,text='Deshacer')
marcoizquierdo.grid(row=0,column=0)
botonconfirmar.grid(row=0,column=0)
botonborrar.grid(row=1,column=0)
botondeshacer.grid(row=2,column=0)
marcoderecho=LabelFrame(ventana,text='Datos')
etiquetanombre = Label(marcoderecho,text='Nombre')
entradatextonombre = Entry(marcoderecho)
etiquetaapellidos = Label(marcoderecho,text='Apellidos')
entradatextoapellidos = Entry(marcoderecho)
marcoderecho.grid(row=0,column=1)
etiquetanombre.grid(row=0,column=0)
entradatextonombre.grid(row=0,column=1)
etiquetaapellidos.grid(row=1,column=0)
entradatextoapellidos.grid(row=1,column=1)
etiquetaresultado = Label(ventana,text='Aquí es donde veremos el resultado')
etiquetaresultado.grid(row=1,column=0,columnspan=2)
ventana.mainloop()
```



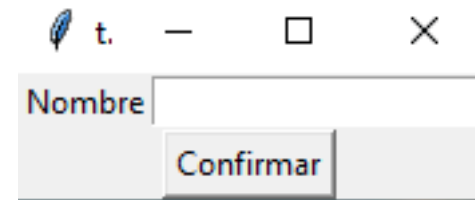
## Eventos

Algunos componentes capturan eventos de usuario (v.gr. botones)

*command* indica la función a ser llamada cuándo el botón es pulsado

```
from tkinter import *
```

```
def funcionconfirmar():  
    print('alguien pulsó el botón')
```



```
ventana = Tk() # crea la ventana  
etiquetadenombre = Label(ventana, text='Nombre') # crea una etiqueta de texto  
widget  
entradadetextodenombre = Entry(ventana) # crea una entrada de texto widget  
# crea un botón mostrando el texto "Confirmar"  
botondeconfirmar = Button(ventana, text='Confirmar', command=  
funcionconfirmar)  
# llama a confirmFunction cuando el botón es pulsado  
# define las preferencias del paquete widget utilizando el diseño de parrilla  
etiquetadenombre.grid(row=0,column=0)  
entradadetextodenombre.grid(row=0,column=1)  
botondeconfirmar.grid(row=1,columnspan=2)  
ventana.mainloop() # abre la ventana y comienza la gestión de eventos
```

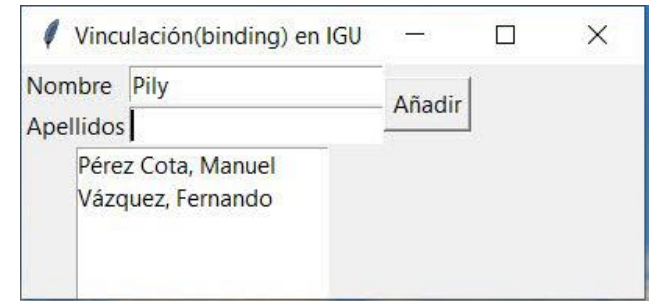


## Vinculación (*Binding*)

Interacciona con los *widgets* desde el código para vincularlo a las variables  
Los cambios realizados en un *widget* automáticamente cambian el valor de las variables y viceversa

```
from tkinter import *
```

```
def meterdatosenlista():
    cajalista1.insert(END, Apellidos.get()+' '+ Nombre.get())
    Nombre.set("") # utilizar get and set para leer y cambiar valores
    Apellidos.set("")
    entradaNombre.focus() # posiciona el cursor sobre entradaNombre
ventana = Tk()
ventana.geometry('400x150')
ventana.title('Vinculación(binding) en IGU')
etiquetaNombre = Label(ventana, text= 'Nombre ')
Nombre = StringVar() # Nombre se unirá a entradaNombre
entradaNombre = Entry(ventana, textvariable = Nombre)
etiquetaApellidos = Label(ventana, text='Apellidos')
Apellidos = StringVar() # Apellido se unirá a entradaApellidos
entradaApellidos = Entry(ventana, textvariable = Apellidos)
botonConfirmar = Button(ventana, text='Añadir', command = meterdatosenlista)
cajalista1 = Listbox(ventana)
etiquetaNombre.grid (row=0,column=0)
entradaNombre.grid(row=0,column=1)
etiquetaApellidos.grid(row=1,column=0)
entradaApellidos.grid(row=1,column=1)
botonConfirmar.grid(row=0,column=2, rowspan=2)
cajalista1.grid(row=2,column=0, columnspan=2)
ventana.mainloop()
```



*StringVar, BooleanVar, IntVar, FloatVar...*

Son variables especiales de tkinter utilizadas para este propósito

## Otros *widgets*

Tkinter nos provee otros muchos *widgets* útiles, incluyendo:

- *Message* (mensaje): como *Label* pero para texto multilínea
- *Text* (texto): como *Entry* con capacidad de edición multilínea
- *Radiobutton*: selección múltiple *checkbox*
- *Spinbox*: seleccionar valores de un grupo dado
- *Scale* (escala): control deslizante para establecer un valor numérico
- *Progressbar*: posición relativa de un valor numérico dentro de un rango
- *Menu*, *Menubutton* y *OptionMenu*: menús desplegables
- *Canvas*: dibujo de imágenes y figuras geométricas básicas (círculos ...)
- *PanedWindow*: como un marco (*frame*) pero el usuario puede cambiar su tamaño
- *Scrollbar* (barra de desplazamiento): control deslizante para gestionar áreas grandes de visualización
- *Notebook* (cuaderno): permite al usuario seleccionar múltiples páginas de contenido
- *Treeview*: representación jerárquica de valores

# Informática para la ingeniería

## Tema 10



# Conceptos Básicos de Informática

- ❖ Información
- ❖ Ordenador Personal
- ❖ Hardware y Software
- ❖ Sistemas Operativos
- ❖ Bases de datos

**Informática:** conjunto de conocimientos técnicos que se ocupan del tratamiento automático de la información por medio del ordenador.

**Información:** conjunto organizado de datos procesados, que constituyen un mensaje que cambia el estado de conocimiento del sujeto o sistema que recibe dicho mensaje.

**Datos:** Representación simbólica de un atributo o variable cuantitativa o cualitativa. Los datos describen hechos empíricos, sucesos y entidades.

**Codificación binaria:** los datos/información que maneja un ordenador se representa como 1 y 0.

**BIT** (Binary digiT): unidad de información más pequeña que puede procesar un ordenador.

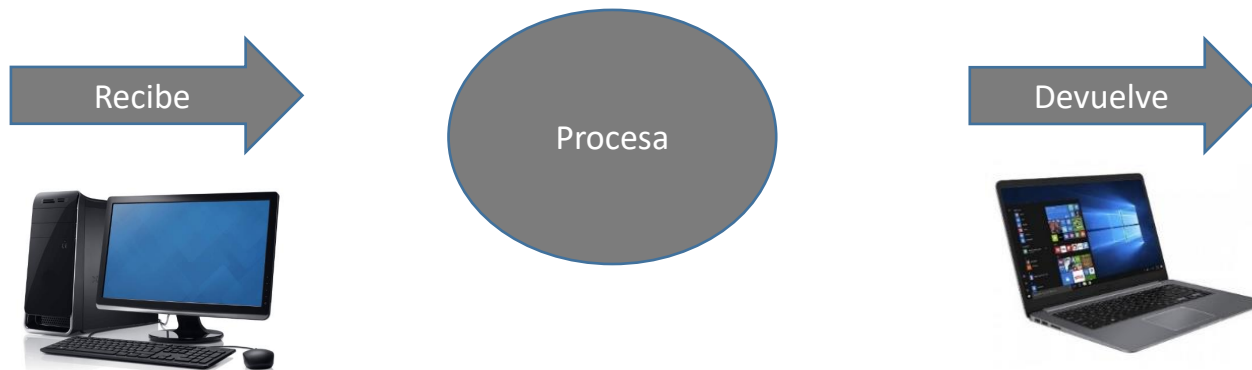
**Byte:** número de bits necesarios para almacenar un carácter. Generalmente: 1 byte = 8 bits

Decimal	Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010

## Un PC u ordenador personal

PC, ordenador o computador es una máquina electrónica que recibe y procesa datos con la misión de transformarlos en información útil y devolverlos. Se encuentra compuesto por:

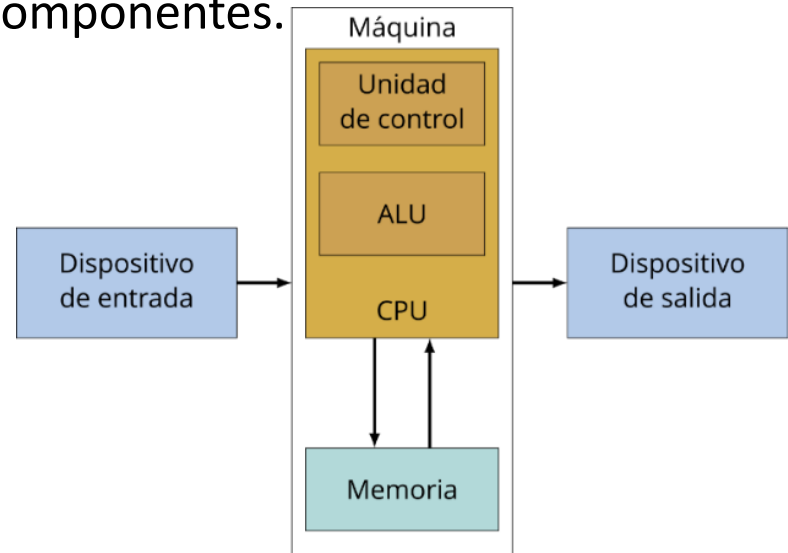
- **Hardware:** conjunto de circuitos electrónicos, cables, dispositivos electromecánicos, y otros elementos físicos que forman la parte física del ordenador.
- **Software:** conjunto de directrices o programas ejecutables por el ordenador. Parte lógica del ordenador.



# Hardware

La tecnología ha evolucionado significativamente pero a grandes rasgos la arquitectura de Von Neumann sirve para explicar las partes principales que conforman un ordenador.

- ❖ **Unidad Central de Proceso:** ejecuta los programas.
- ❖ **Memoria Principal:** contiene los programas en ejecución y sus datos.
- ❖ **Dispositivos de Entrada y Salida:** permiten la interacción del ordenador con el exterior.
- ❖ **Bus del sistema:** comunica todos los componentes.



## Hardware. Unidad central de proceso o CPU

Llamada también procesador o microprocesador. Es el “cerebro” de la máquina. Su misión es supervisar todas las funciones que realiza el ordenador, comparar el resultado de las operaciones, incluyendo las matemáticas, y seguir las instrucciones contenidas en los programas que se ejecutan. Es decir:

- ❖ Detecta señales de estado procedentes de las distintas unidades.
- ❖ Obtiene de la memoria las instrucciones máquina del programa de manera secuencial.
- ❖ Genera señales de control dirigidas a todas las unidades monitorizando las operaciones que implican la ejecución de la instrucción.
- ❖ Contiene un reloj que sincroniza todas las operaciones elementales de la computadora (periodo de reloj → tiempo de ciclo) GHz.





## Hardware. Unidad central de proceso o CPU

Tres elementos básicos:

- ❖ **Memoria principal:** almacena el programa que determinará la actuación y los datos que serán manejados por la CPU.
- ❖ **Unidad de control:** coordina y controla las operaciones que se hagan con los datos. Lee los datos necesarios de la memoria y activa los circuitos necesarios de la ALU.
- ❖ **Unidad lógico-aritmética (ALU):** realiza las operaciones aritméticas y lógicas con los datos que recibe de la unidad de control; procedentes de la memoria principal.

## Hardware. Núcleo

Un núcleo es una unidad de procesamiento en sí misma; lee instrucciones y ejecuta las acciones específicas. Puede ser capaz de hacer ciclos de procesamiento de forma independiente: lectura, decodificación, ejecución y escritura.

Con dos núcleos se puede paralelizar y llevar a cabo dos ciclos de instrucción independientes, como si tuviéramos dos procesadores, y así sucesivamente. Más núcleos implica mayor capacidad de instrucción simultánea. La velocidad de los ciclos de instrucciones **NO** depende de la cantidad de núcleos.

La frecuencia de reloj es lo que define la velocidad con que se llevan a cabo estos ciclos de instrucciones. Se mide en GHz; cuanto mayor sea, más rápido será capaz cada uno de estos núcleos en completar esas cuatro partes que componen un ciclo de instrucción completo. Elementos que influyen en el rendimiento del procesador:

- ❖ Núcleos
- ❖ Frecuencia del reloj
- ❖ Arquitectura CPU
- ❖ Memoria y más

Se puede consultar en Internet test de Rendimiento de los procesadores actuales

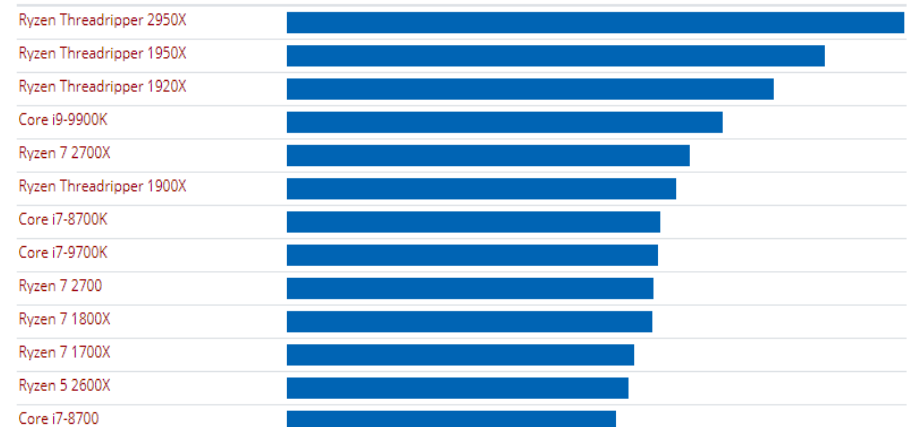
<https://novabench.com/>

<https://www.userbenchmark.com/Software>

<https://www.sisoftware.co.uk/> y muchos más.

### Rendimiento procesadores

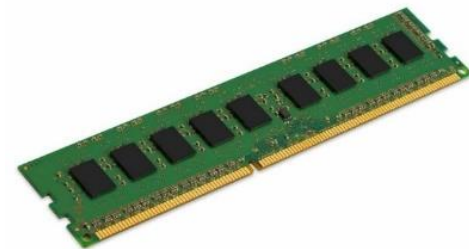
#### multinúcleo



## Memoria principal o RAM

Es una memoria rápida de acceso aleatorio: RAM (Random Access Memory). Se puede leer y escribir en cualquier celda y el tiempo necesario para ello es el mismo. En ella se carga el sistema operativo, los programas que estemos usando, variables, archivos de trabajo, etc. Sus características más importantes son su capacidad de almacenamiento (se mide en Bytes) y su velocidad.

Es una memoria volátil, su contenido se borra al apagar el ordenador.



# Hardware. Bus del Sistema

Comunica los distintos componentes del ordenador. Formado por conductores metálicos los cuales transmiten señales eléctricas que son enviadas y recibidas por los distintos componentes. Se caracteriza:

- ❖ Cantidad de información que transmite (bits), número de líneas físicas mediante las cuales se envía la información. Un cable de 32 hilos permite la transmisión de 32 bits en paralelo.
- ❖ Velocidad se define a través de su frecuencia (Hz), número de paquetes de datos que pueden ser enviados o recibidos por segundo.
- ❖ El ancho de banda es la cantidad de información que puede transferir por unidad de tiempo.

## CABLE BUS



Por el BUS viaja la información en forma de 0 y 1 (bits).  
Cuando por un cable del bus hay corriente es un estado digital de 1 (1bit)  
Cuando por un cable del bus no hay corriente es un estado digital 0 (1bit)  
8 bits viajando en un momento determinado es un byte de información

## TIPOS DE BUSES

ATA o IDE



Conector IDE

Conexión en paralelo, el envío de muchos bits a la vez a través de un conector ancho, plano, con una velocidad de transferencia de datos máxima de 133 MB/s

SATA

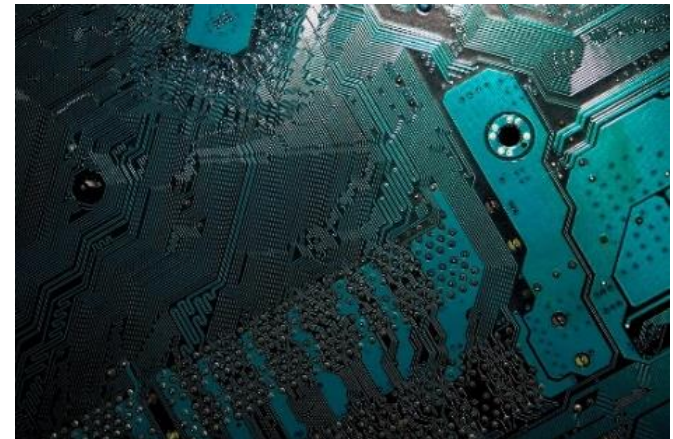
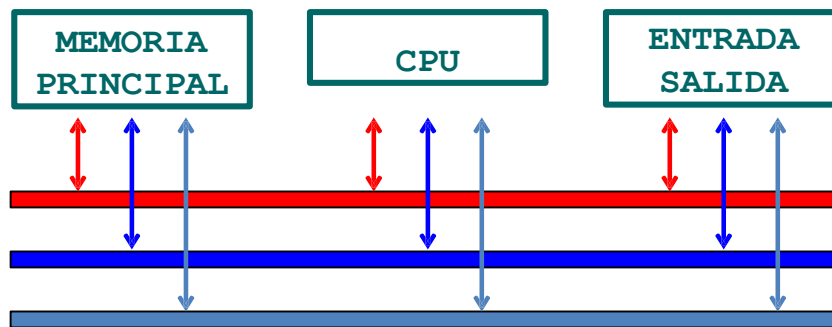


Conector SATA

Conexión Serie, cable y conector más pequeño. Velocidades de transferencia de datos de 600MB/s

# Hardware. Bus del Sistema

- ❖ Bus de **datos**: transmite la información.
- ❖ Bus de **direcciones**: identifica el origen o destino de los datos.
- ❖ Bus de **control**: indica la operación que se debe realizar con la información (p. ej. escribirla en memoria).



Bus de datos

Bus de dirección

Bus de control

<https://youtu.be/THCPH3HmWD0>

## Hardware. Periféricos I/O

**Dispositivos de entrada:** Permiten ingresar información a la unidad de procesamiento; ya sea del usuario o de otro soporte físico. Traducen los datos en impulsos eléctricos, que luego son transmitidos al ordenador para su proceso y almacenamiento. Son vitales para permitir la comunicación entre el ordenador y el exterior.

- ❖ Teclados: tablero de botones a los que les asigna valores conforme a un lenguaje específico.
- ❖ Ratón: ingresan información a sistemas de representación visual traduciendo los movimientos que el usuario realiza con el dispositivo a instrucciones.
- ❖ Micrófonos: capturan el sonido (ondas sonoras en el aire) y lo traducen a impulsos eléctricos que luego pueden ser codificados y almacenados o transmitidos y reproducidos.
- ❖ Cámaras: capturan la imagen empleando un sistema de lentes y componentes fotosensibles, para almacenar digitalmente la imagen y movimiento reales.
- ❖ Escáneres: capaces de “leer” la imagen dispuesta en su bandeja para transmitir una copia digital de la misma.
- ❖ Lectores de código de barras: lectores ópticos que reconocen un código de barras (líneas negras sobre un fondo blanco) en el cual está contenida la información del producto.
- ❖ Joysticks: traslada al ordenador los comandos que el usuario desea transmitir al videojuego, permitiéndole controlar lo que ocurre e interactuar con el sistema.

## Hardware. Periféricos I/O



## Hardware. Periféricos I/O

**Dispositivos de salida:** Permiten la extracción o recuperación de información proveniente del ordenador o de otro soporte físico.

- ❖ Impresoras: capaz de convertir en un documento impreso el contenido digital del computador. Pueden ser de inyección de tinta o de láser.
- ❖ Parlantes: convierten los impulsos eléctricos en señales sonoras que los usuarios pueden escuchar.
- ❖ Videobeams y proyectores: reciben información y la representan gráficamente, proyectándola como haces de luz en una superficie destinada para ello.





## Hardware. Periféricos I/O

Existen dispositivos mixtos o de entrada/salida (E/S)- input/output (I/O), los cuales pueden cumplir con ambas funciones:

- ❖ Monitor: convierte las señales digitales en información visual, representada gráficamente. Varían en su capacidad de calidad visual, algunos permiten el ingreso de información a través de pantallas táctiles.
- ❖ Disco Compacto (CD) y Disco de Video Digital (DVD): grabadas mediante un rayo óptico (láser), permiten almacenar y/o sobre todo recuperar información.
- ❖ Disco duro: almacenamiento externo regrabable de alta capacidad (Gigabytes o Terabytes) y velocidad. Una cabeza magnética escribe o lee la información modificando o detectando las propiedades magnéticas del material. Pueden ser portátiles.
- ❖ La tarjeta de red, adaptador de red o adaptador LAN conecta un ordenador a una red informática y posibilita compartir recursos, tales como archivos, discos duros, impresoras e internet, etc., entre dos o más ordenadores.

## Hardware. Placa Base

Placa física sobre la que se conectan los distintos componentes internos del ordenador. Es la columna vertebral del ordenador y en ella están algunos de los componentes o partes más importantes del equipo. **Se conectan todos los componentes internos del ordenador**, desde el procesador hasta los discos duros, la memoria RAM o la tarjeta gráfica. Cada uno de estos componentes tiene su propia ranura para conectarlo.

En la estructura básica de la placa base destaca la **placa de circuito impreso**, también conocida como PCB (*Printed Circuit Board*). Es un sustrato no conductor de la carga eléctrica sobre el que se colocan los demás componentes.

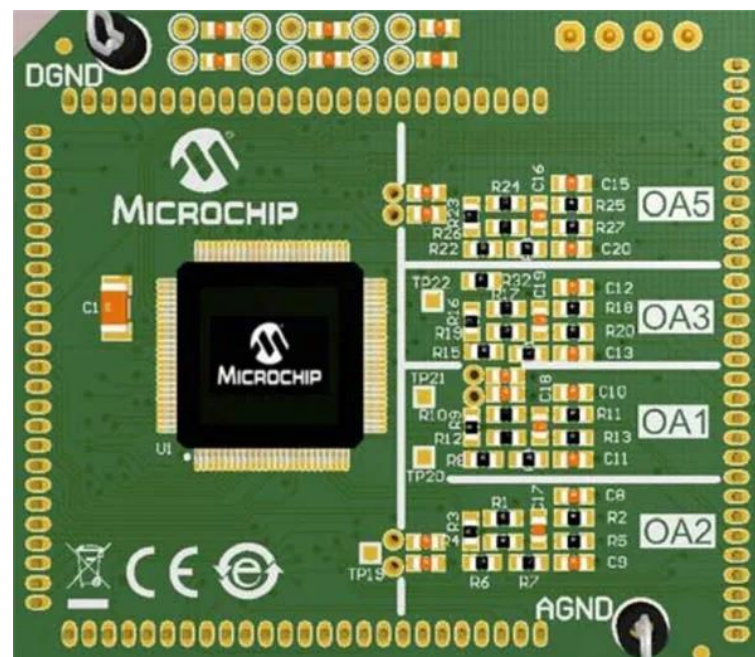
Incorpora elementos propios imprescindibles para el funcionamiento del ordenador, entre los que destaca la **BIOS**.



<https://hardzone.es/2018/02/18/conectores-placa-base/>

## Hardware. Placa Base: chipset

- ❑ El chipset es el auténtico cerebro de la placa base.
- ❑ Suele ser el circuito integrado más grande de todos. Su nombre significa literalmente conjunto de chips, y su función es controlar el flujo de datos entre diferentes componentes clave del ordenador, como el procesador, la memoria y los diferentes periféricos que haya conectados.
- ❑ El tipo de chipset que tenga la placa base determinará otras características que se puede o no tener. Por ejemplo, no todos los chipsets son compatibles con todos los procesadores de cada marca.
- ❑ El chipset también condiciona las cantidades máximas de conectores como los PCI Express, puertos SATA o USB que puede haber: establece un máximo de módulos como las tarjetas gráficas, la memoria RAM o conectores USB y discos duros.



## Hardware. Placa Base: sistema de alimentación eléctrica

- ❑ El **sistema de alimentación eléctrica** es el conjunto interno de componentes que se encargan de la regulación del voltaje interno de la placa base, así como sus fases de alimentación eléctrica.
- ❑ La placa tiene muchos componentes y circuitos, y algunos de ellos pueden llegar a generar mucho calor, y si algunos componentes alcanzan o superan ciertos umbrales máximos de temperatura durante mucho tiempo podrían dañarse para siempre. Por eso, la mayoría de placas base tienen **disipadores y ventiladores** que ayudan a evacuar el calor residual.



## Hardware. Placa Base: BIOS

**BIOS** (Basic Input–Output System) Sistema Básico de Entrada - Salida: contiene el software que se ejecuta durante el arranque:

- ❖ Verificación de los componentes vitales: memoria, microprocesador, discos duros, teclado, etc.
- ❖ Comprueba las claves de acceso.
- ❖ Especifica los dispositivos en los que buscará el S.O. al arrancar.
- ❖ Permite deshabilitar ciertos elementos (infrarrojos, por ejemplo)
- ❖ Reloj: marca los instantes en los que se realizan las acciones.

<https://hardzone.es/2018/02/18/conectores-placa-base/>

## Hardware. Placa Base

Los diferentes tipos de **conectores** permiten conectar periféricos y componentes. Además de la cantidad de conectores, el chipset también determina la versión de cada uno de ellos. Conectores habituales:

- **PCI Express:** Para conectar tarjetas. Generalmente se utilizan para las tarjetas gráficas, pero puede haber otras tarjetas enfocadas a aspectos más técnicos como tarjetas de comunicaciones.
- **USB**
- **SATA:** Para los discos duros
- **M.2:** Para las unidades de estado sólido o SSD.



Los periféricos suelen estar formados por dos partes: una mecánica (motores, electroimanes, etc.) y otra formada por circuitos electrónicos, mucho más rápida que la anterior.



## Hardware. Placa Base: audio y wifi

- ❑ Las placas base de los ordenadores también suelen tener :
  - ❑ **chips de sonido**, que determinan la calidad del sonido que va a tener el PC.
  - ❑ módulos **WiFi** y controladoras **Ethernet**, que van a determinar la velocidad máxima de conexión de ambos.
- ❑ En todos los casos, es posible usar los conectores de la placa base para añadir cualquiera de estos módulos.

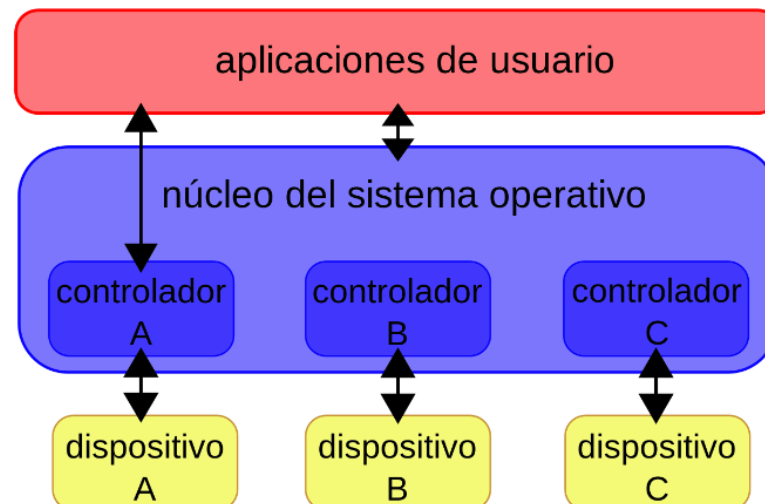


## Controladores del dispositivo

**Software** que conecta el sistema operativo (SO) directamente con los componentes del hardware. El controlador de dispositivo (driver) le da instrucciones al sistema operativo sobre cómo debe funcionar determinado hardware y de que forma el sistema debe trabajar en conjunto para suministrarte los mejores resultados.

Es común el uso de **controladores genéricos**, pero para obtener el máximo rendimiento se recomienda usar los drivers del fabricante del hardware.

Los drivers dependen del sistema operativo y de la versión del mismo.



<https://tecnologia-informatica.com/que-son-drivers-controladores/>



# Software

**Software** Conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación (IEEE). En resumen, programas que integran un ordenador y le permiten realizar tareas. El software da instrucciones al hardware de la forma como debe realizar una tarea.

Ejemplos: sistemas operativos, navegadores web (Internet Explorer, Google Chrome, Firefox), antivirus, LibreOffice, Microsoft Word, Adobe Acrobat, lenguajes programación, etc.

**El software es la parte intangible de la máquina,  
Pero con la que el usuario interactúa.**



## Software propietario y software libre

**Software propietario** los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones), su código no está disponible o el acceso se encuentra restringido. Se requiere una autorización. Una determinada persona o entidad tiene la titularidad de los derechos de autor.

**Software libre** capacidad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software que se ha puesto en sus manos; esto implica gran ventaja porque el código fuente es colocado a disposición del usuario. La primera característica es que para utilizar o descargar esta clase de software no es necesario realizar ningún pago.



[http://www.scielo.org.mx/scielo.php?script=sci\\_arttext&pid=S0187-358X2011000200003](http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S0187-358X2011000200003)

# Software

**Software de sistema:** Desvincula al usuario de los detalles del sistema informático, aislándolo especialmente de las características internas de: memoria, discos, puertos y dispositivos de comunicaciones, etc. Incluye: **Sistemas operativos (SO)**, Controladores de dispositivos, Herramientas de diagnóstico, y más.

**Software de programación:** Conjunto de herramientas que permiten desarrollar programas de informática, usando diferentes lenguajes de programación. Incluye: Compiladores, Intérpretes, Enlazadores, Depuradores, Entornos de desarrollo integrados (IDE), etc.

**Software de aplicación:** Permite a los usuarios llevar a cabo tareas específicas, en cualquier campo de actividad susceptible de ser automatizado o asistido. Incluye: Aplicaciones para Control de sistemas y automatización industrial, Aplicaciones ofimáticas, Software educativo, Software empresarial, **Bases de datos**, Videojuegos, Software de cálculo numérico y simbólico, Software de diseño asistido (CAD), Software de control numérico (CAM), y mucho más.

*Posible clasificación del software (wikipedia)*

# Sistema Operativo

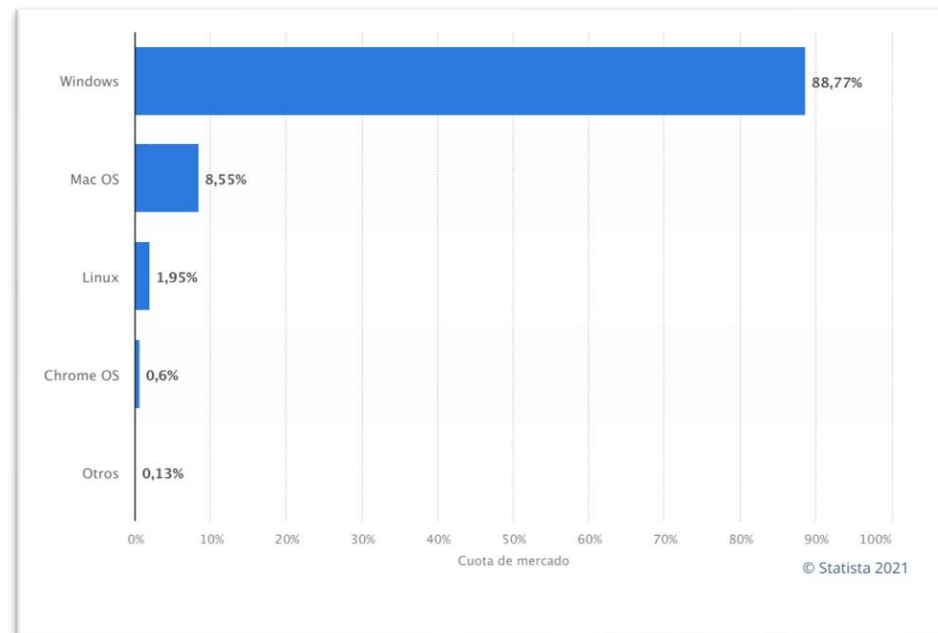
Es software (programa o conjunto de ellos) que controla el funcionamiento del equipo físico, ocultando los detalles de la máquina y haciendo más simple y eficiente el uso del ordenador. Interfaz entre los dispositivos de hardware y los programas usados. Se encarga de:

- ❖ Gestión de procesos: permite que varios programas se ejecuten a la vez.
- ❖ Abstracción hardware: trabaja con los diferentes modelos de hardware (p.ej: distintos discos duros) de forma que los programas no tengan que preocuparse de los detalles. Cada fabricante programa un “driver”.
- ❖ Gestión de la Entrada/Salida: permite leer y escribir información en los dispositivos de E/S. Incluye la gestión de archivos.
- ❖ Gestión de memoria: permite que los programas usen la memoria RAM sin notar que hay otros programas usándola (piensan que tienen toda la memoria para ellos).

# Sistemas Operativos. Clasificación

- ❖ En función de las tareas:
  - ❖ Monotarea: Solamente puede ejecutar un proceso (aparte de los procesos del propio S.O.) en un momento dado.
  - ❖ Multitarea: Capaz de ejecutar varios procesos al mismo tiempo. Este tipo de S.O. normalmente asigna los recursos disponibles (CPU, memoria, periféricos) de forma alternada a los procesos que los solicitan.
- ❖ En función de los usuarios:
  - ❖ Monousuario: Si sólo permite ejecutar los programas de un usuario al mismo tiempo.
  - ❖ Multiusuario: Si permite que varios usuarios ejecuten simultáneamente sus programas, accediendo a la vez a los recursos de la computadora.
- ❖ En función del manejo de recursos:
  - ❖ Centralizado: Permite utilizar los recursos de una sola computadora.
  - ❖ Distribuido: Permite utilizar los recursos (memoria, CPU, disco, periféricos...) de más de un ordenador al mismo tiempo.

# Sistemas Operativos. Ejemplos



# Sistemas Operativos Móvil

Controla un dispositivo móvil al igual que las PC que utilizan Windows o Linux, los dispositivos móviles tienen sus sistemas operativos como Android, IOS entre otros. Los sistemas operativos móviles son mucho más simples y están más orientados a la conectividad inalámbrica, los formatos multimedia para móviles y las diferentes maneras de introducir información en ellos.

Algunos de los sistemas operativos utilizados en los dispositivos móviles están basados en el modelo de capas.

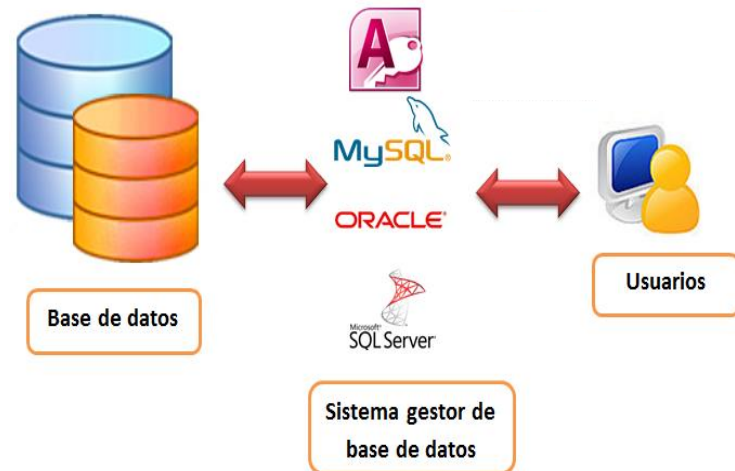


# Bases de Datos

Conjunto de información perteneciente a un mismo contexto, ordenada de modo sistemático para su posterior recuperación, análisis y/o transmisión. Van desde una biblioteca hasta los datos de usuarios de una empresa de telecomunicaciones. Muchas webs tienen detrás un gestor de contenidos que, a su vez, utiliza un **SGBD** para almacenar la información de la web.

Algunas características son:

- ❖ Independencia lógica y física de los datos.
- ❖ Redundancia mínima.
- ❖ Acceso concurrente por parte de múltiples usuarios.
- ❖ Integridad de los datos.
- ❖ Consultas complejas optimizadas.
- ❖ Seguridad de acceso y auditoría.
- ❖ Respaldo y recuperación y más.
- ❖ Acceso a través de lenguajes de programación.





# Bases de Datos

Existen bases de datos libres y propietarias, algunos ejemplos son: Oracle, IBM DB2, Microsoft SQL Server, Informix, MySQL y PostgreSQL.

Los Sistemas de Gestión de Base de Datos (**DataBase Management System DBMS**) son software dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta.

**El lenguaje SQL** es el más universal en los sistemas de base de datos. Permite realizar consultas a nuestras bases de datos para mostrar, insertar, actualizar y borrar datos.



# Informática para la ingeniería

FIN