

entero, flotante, booleano, cadena, octal, hexadecimal

```
int 783 0 -192 0b010 0o642 0xF3
      cero binario octal hexadec.
float 9.23 0.0 -1.7e-6
bool True False
str 'Uno\nDosx'
Retorno de línea 'escapado' '""X\tY\tZ 1
'I\''ame'
'escapado'
bytes b"pepe\xfe\775"
      hexadecimal octal
```

DOCUMENTACION:  
DESACTIVAR COD:  
'''print('Hola')'''

Tipos de base

Tipos Contenedores

secuencias ordenadas, acceso a índices rápidos, ...

```
list [1,5,9] ['x',11,8.9] ['palabra']
tuple (1,5,9) 11,'y',7.4 ("palabra",)
```

Valores no modificables (inmutables) expresión separadas por coma → tuple

contenedores de claves, sin orden a priori, acceso rápido a claves, claves únicas

diccionario dict {'clave': 'valor'} dict(a=3,b=4,k='v')  
(parejas clave/valor) {1: 'uno', 3: 'tres', 2: 'dos', 3.14: 'pi'}

conjuntos set {'clave1', 'clave2'} {1,9,3,0} set()  
claves = valores hash (tipos básicos, inmutables ...) frozenset conjunto inmutable vacío

Identificadores

Para nombres de variables, funciones, módulos, clases ...

a...zA...Z seguido de a...zA...Z\_0...9

- los acentos son posibles pero mejor evitar
- palabras reservadas de lenguaje prohibidas
- distingue entre minúsculas/MAJUSCULAS

⊙ a pepe x7 y\_max BigOne  
⊙ my and for

Conversiones

```
int('15') → 15 type (expresión)
int('3f',16) → 63 especificación de la base del número entero en el 2º parámetro
int(15.56) → 15 truncamiento de la parte decimal
float('-11.24e8') → -1124000000.0
round(15.56,1) → 15.6 redondeado al primer decimal
bool(x) False para x cero, x contenedor vacío, x None o False ; True para otros x
str(x) → '...' cadena de representación de x a mostrar (ver formato posteriormente)
chr(64) → '@' ord('@') → 64 código ↔ carácter
repr(x) → '...' cadena de representación literal de x
bytes([72,9,64]) → b'H\t@'
list('abc') → ['a','b','c']
dict([(3,"trois"),(1,"un")]) → {1:'un',3:'tres'}
set(["un","deux"]) → {'uno','dos'}
str union y secuenciación str → str unido
'.join(['pepe','12','pswd']) → 'pepe:12:pswd'
str cortar con espacio en blanco → list de str
"palabras espaciadas".split() → ['palabras','espaciadas']
str cortar con str separador → list de str
'1,4,8,2'.split(",") → ['1','4','8','2']
secuencia de un tipo → list de otro tipo
[int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Variables y Asignación

=

asignación ↔ asociación de un nombre a un valor  
1) evaluación del valor de la expresión de la derecha  
2) asignación ordenada con los nombres de la izquierda.

```
x=1.2+8+sin(y)
a=b=c=0 Asignación a un mismo valor
y,z,r=9.2,-7.6,0 asignaciones múltiples
a,b=b,a Intercambio de valores
a,*b=seq } secuencia de desempaquetado en
*a,b=seq } elemento y lista
x+=3 incremento ⇔ x=x+3 también *=
x-=2 decremento ⇔ x=x-2 /=
x=None valor constante « no definido » %=
del x borrando el nombre x ...
```

Indices para las secuencias

para las listas, tuplas, cadenas de caracteres, bytes...

índices negativos	-5	-4	-3	-2	-1
índices positivos	0	1	2	3	4

```
lst=[10,20,30,40,50]
corte positivo 0 1 2 3 4 5
corte negativo -5 -4 -3 -2 -1
```

Acceso a las sub-secuencias de lst [corte inicial : corte final : incremento]

```
lst[:-1] → [10,20,30,40] lst[:: -1] → [50,40,30,20,10] lst[1:3] → [20,30] lst[:3] → [10,20,30]
lst[1:-1] → [20,30,40] lst[:: -2] → [50,30,10] lst[-3:-1] → [30,40] lst[3:] → [40,50]
lst[::2] → [10,30,50] lst[:] → [10,20,30,40,50] copia de la secuencia
```

Omitiendo algún parámetro del corte → desde el principio/ hasta el final.

En secuencias mutables (list), se puede eliminar elementos con del lst[3:5] y modificar asignando lst[1:4]=[15,25]

Logica booleana

Comparadores: < > <= >= == != (resultado booleano) ≤ ≥ = ≠

a and b y lógico ambos simultáneamente

a or b o lógico uno, el otro, o ambos

⊘ trampa: and y or devuelve el valor de a o b (el que sea más corto) ⇒ asegúrese de que a y b sean booleanos.

not a no lógico

True False } constantes Verdadero/Falso

Bloques de sentencias

sentencia control:

bloque de instrucciones 1...

sentencia control:

bloque instrucc 2...

Sangría!

sentencia siguiente a bloque 1

⊘ configura el editor para insertar 4 espacios cuando pulsemos un tabulador como sangría.

module truc ⇔ fichier truc.py Importar módulos/nombres

```
from monmod import nom1,nom2 as fct
→ acceso directo a nombres, renombrando con as
import monmod → acceso a través de monmod.nom1 ...
⊘ Módulos y paquetes buscados en el 'path' de python. (ej. sys.path)
```

Sentencias Condicionales

un bloque de instrucciones es ejecutado, solo si su condicion es verdadera

if condicion logica:

bloque de instrucc.

Combinable con sino si, sino si... y un solo sino final. Solo se ejecuta el bloque de la primera condición encontrada verdadera.

No usar ⇔ SI USAR:

```
if bool(x)==True: ⇔ if x:
if bool(x)==False: ⇔ if not x:
```

if edad<=18: estado="Jugar"
elif edad>65: estado="Retirado"
else: estado="Activo"

Matemáticas

Los ángulos son en radianes

```
from math import sin,pi...
sin(pi/4) → 0.707...
cos(2*pi/3) → -0.4999...
sqrt(81) → 9.0
log(e**2) → 2.0
ceil(12.5) → 13
floor(12.5) → 12
modulos math, statistics, random, decimal, fractions, numpy, etc.
```

Operadores: + - \* / // % \*\*

Prioridades (...)

@ → × matriz python 3.5+ numpy

```
(1+5.3)*2 → 12.6
abs(-3.2) → 3.2
round(3.57,1) → 3.6
pow(4,3) → 64.0
⊘ prioridades habituales
```

Excepciones de error

Señales: raise

ExcClass(...) Tratamiento:

```
try: bloque tratamiento intentar
except ExcClass as e:
    bloc tratamiento erroneo
eslg: bloque sin peligro
```

tratamiento raise X() normal

tratamiento error raise error

⊘ bloc finally para tratamientos finales en todos los casos..

⊘ buena practica: si dato a convertir descontrolado → Gestión excepciones

**Sentencia de bucle condicional**

bloque de instrucciones ejecutado siempre que la condición sea verdadera

**while** *condición lógica:* → *bloque de instrucciones*

*Cuidado con los bucles sin fin.*

```
s = 0
i = 1
while i <= 100:
    s = s + i**2
    i = i + 1
print('suma:', s)
```

Inicialización antes del bucle.  
Condición con al menos un valor variable (aquí i)  
variar la variable de control!!

**Control de bloque**

**Break** salida inmediata  
**continue** sig. iteración

PROHIBIDAS: Teorema de la prog. estructurada

**Algoritmo:**  $s = \sum_{i=1}^{100} i^2$

**Sentencia Bucle Iterador**

bloque de instrucciones ejecutadas para cada elemento de un contenedor o iterador

**for var in secuencia:** → *bloque de instrucciones*

Recorrido por los valores de un contenedor

```
s = "pepe"
cpt = 0
for c in s:
    if c == "e":
        cpt = cpt + 1
print("encontradas ", cpt, "e")
```

Inicialización antes del bucle

Algoritmo: contar el número de 'e' en la cadena.

Recorrido sobre un dict/set ⇔ recorrer la secuencia de claves

**Visualización**

```
print('v=', 3, 'cm:', x, 'y+4')
```

Elementos a visualizar: valores literales, variables, expresiones

Opciones de **print**:

- `sep=' '` separador de elementos, default espacio fin
- `end='\n'` línea, default fin de línea
- `file=sys.stdout` imprimir en un archivo, default salida estándar

**Ejecución**

`s = input("Directivas:")`

`input` devuelve siempre una **cadena**, convertirlo al tipo deseado (ver cuadro de las conversiones)

Recorrer los **índices** de un contenedor secuencialmente

- modificar el elementos correspondiente a la posición
- acceder a los elementos alrededor del índice (antes/después)

VALOR = (2, 1, 0.5, 0.2, 0.1)  
cantidad = [2, 0, 0, 0, 4]

**Algoritmo:** determinación de cantidad disponible en monedero.

```
t = 0.0
for pos in range(len(VALOR)):
    t += VALOR[pos] * cantidad[pos]
print('total monedero=', t)
```

Por supuesto simultánea **índices** y **valores** de la secuencia:

```
for idx, val in enumerate(lst):
```

**Operaciones genéricas sobre contenedores**

`len(c)` num. de elemnt.  
`min(c)` `max(c)` `sum(c)`

`sorted(c)` → **list** copia ordenada

`val in c` → booleano, operador **in** busca la presencia (**not in** ausencia)

`enumerate(c)` → **iterador** (índice, valor)

`zip(c1, c2...)` → **iterador** sobre tuplas que contiene los elementos del índice de c

`all(c)` → **True** si los elementos de **c** son verdad, de lo contrario **False**

`any(c)` → **True** si al menos un elemento de **c** es verdadero, sino **False**

`c.clear()` Borra los contenidos de los diccionarios, conjuntos, listas.

Específico para contenedores de secuencias ordenados (listas, tuplas, cadenas, ...)

`reversed(c)` → **iterdor** inverso    `c*5` → duplicación    `c+c2` → concatenación

`c.index(val)` → posición    `c.count(val)` → número de ocurrencias

`import copy`

`copy.copy(c)` → Copia superficial del contenedor.

`copy.deepcopy(c)` → Copia profunda del contenedor.

**Secuencias de enteros**

**range([inicio,] final [,paso])**

`# inicio` default 0, `final` no incluido en la secuencia, `paso` sig. default 1

`range(5)` → 0 1 2 3 4    `range(2, 12, 3)` → 2 5 8 11

`range(3, 8)` → 3 4 5 6 7    `range(20, 5, -5)` → 20 15 10

`range(len(seq))` → Secuencia de índices de valores en **seq**

`# range` proporciona una secuencia de enteros contruidos según sea necesario

**Operaciones sobre Listas**

`# modificar lista original`

`lst.append(val)` añadir elemento al final

`lst.extend(seq)` añadir secuencia de elementos al final

`lst.insert(idx, val)` insertar elemento en una determinada posición

`lst.remove(val)` elimina el primer ítem con determinado valor **val**

`lst.pop([idx])` → **valor** elimina determinado ítem y retorna su valor

`lst.sort()` `lst.reverse()` ordena / invierte la lista original

**Definición de Funciones**

Nombre de la función (identificador) ↓ parámetros nombrados

```
def fct(x, y, z):
    """documentacion"""
    # bloque instrucciones, calculo de resultado, etc.
    return res
```

`return res` ← valor resultado. Si no hay resultado, se retorna: **return None**

`# parámetros y variables sólo existen dentro del bloque y durante la llamada a la función ("caja negra")`

Avanzado: `def fct(x, y, z, *args, a=3, b=5, **kwargs):`

`*args` número de argumentos posicionales (→ **tuple**), valores por defecto.

`**kwargs` número de argumentos nombrados (→ **dict**)

**Operaciones en Diccionarios**

`d[clave]=valor` `del d[clave]`

`d[clave]` → **valor**

`d.update(d2)` { actualiza/añade asociaciones

`d.keys()` `d.values()` → ver clave/valor o ambos

`d.items()`

`d.pop(clave[, default])` → **valor**

`d.popitem()` → (clave, valor)

`d.get(clave[, default])` → **valor**

`d.setdefault(clave[, default])` → **va**

**Operaciones sobre Conjuntos**

Operadores:

- | → unión (carácter barra vertical)
- & → intersección
- ^ → diferencia/diferencia simétrica
- < <= > >= → relaciones de inclusión

Los operadores también existen como métodos..

`s.update(s2)` `s.copy()`

`s.add(clave)` `s.remove(clave)`

`s.discard(clave)` `s.pop()`

**Llamadas a funciones**

`r = fct(3, i+2, 2*i)`

Almacenamiento o usa el valor de retorno

un valor de argumento por parametro

`# Es el uso del nombre de la función entre paréntesis lo que hace la llamada.`

avanzado: `*secuencia` `**diccionario`

**Ficheros**

Almacenamiento de datos en disco y re-lectura

```
f = open("fic.txt", "w", encoding="utf8")
```

**variable** Fichero para su operación y manejo

**nombre del fichero** (+path...)

**modos de apertura**

- 'r' lectura (read)
- 'w' escritura (write)
- 'a' añadir (append)

**codificación de caracteres en el fichero:** `Utf8`, `ascii`, `latin1` ...

**en escritura**

`f.write("coucou")`

`f.writelines(list de lignes)`

`f.close()` No olvides cerrar el archivo después de usarlo!

**en lectura**

`f.read([n])` → **siguientes caracteres** si **n** sin especificar, leer hasta el final!

`f.readlines([n])` → **list** siguientes líneas

`f.readline()` → siguiente línea

`f.flush()` escritura de la caché

`f.truncate([tamaño])` recortado lectura / escritura avanza de forma secuencial en el archivo, modificable con:

`f.tell()` → posición `f.seek(posicion[, origen])`

Muy frecuentes: apertura en el bloque protegido **with open(...)** as **f:** (cierra automático) y el bucle para leer las líneas de un archivo de texto.

# tratamiento de **línea**

**Operaciones sobre cadenas**

`s.startswith(prefixo[, inic[, fin]])`

`s.endswith(suf[, inic[, fin]])` `s.strip([caractères])`

`s.count(sub[, inic[, fin]])` `s.partition(sec)` → (antes, sec, después)

`s.index(sub[, inic[, fin]])` `s.find(sub[, inic[, fin]])`

`s.is...()` test sobre la categoría del contenido (ex. `s.isalpha()`)

`s.upper()` `s.lower()` `s.title()` `s.swapcase()`

`s.casefold()` `s.capitalize()` `s.center([ancho, rempl])`

`s.ljust([anch, rempl])` `s.rjust([anch, rempl])` `s.zfill([anch])`

`s.encode(codificación)` `s.split([sep])` `s.join(seq)`

**directivas de formateo** **valores a formatear** **Formateos**

"modele{ } { } { }".format(x, y, r) → **str**

"{selección:formateo!conversión}"

**Selección:**

- 2
- nom
- 0.nombre
- 4[clave]
- 0[2]

**Formato:**

`car-relleno.` `alineación` `signo` `ancho.mini` `precision` `ancho.max` `tipo`

`<>^ = + - espace 0` al nicio, para rellenar con 0

enteros: **b** binario, **c** caracter, **d** decimal (default), **o** octal, **x** ou **X** hexa...

reales: **e** ou **E** exponencial, **f** ou **F** punto fijo, **g** ou **G** genera (default), cadena: **s** ... % porcentaje

**Conversion:** **s** (texto legible) o **r** (representación litera)