Estructuras de Control

Enero 13, 2026

Esteban Sánchez

1 Estructuras de Control

1.1 Selección

- Selecciona una dde las varianbles según a una condición
- Indentación para estructurar el código
- El símbol ":" es sumamente importante

In [1]:
```
# # Asigna un valor la variable x
x = int(input("Ingrese el valor de  x"))
if x>0:
    print('Valor positivo')
else :
    print('Valor no positivo')
```

```
Valor positivo
```

- Las condiciones son expresiones que se evaluan a True o False

In [2]:
```
a = 5
b = [5, 6, 7]
print(a > 0)

print(a > 0 and a < 10)
print(a is not b)
print(a in b)
```

```
True
True
True
True
```

- Se puede introducir más "ramas" a través de *elif*

In [3]:
```
c = -6
if c>0:
    print('Valor positivo')
elif c==0:
    print('Valor nulo')
else :
    print('Valor negativo')
```

```
Valor negativo
```

- Anidamiento

In [4]:
```
var = 120
if var > 0:
    if var < 100:
        print('Valor positivo')
    else:
        print('Valor muy positivo')
else:
    print('Valor negativo')
```

```
Valor muy positivo
```

2. Switch

In [5]:
```
x = -7
match x:
    case _ if x > 0:
        print('Positivo')
    case _ if x == 0:
        print('Cero')
    case _ if x < 0:
        print('Negativo')
```

```
Negativo
```

In [7]:
```
dia = "2"

match dia :
    case 'L':
        print("Lunes")
    case 'M':
        print("Martes")
    case 'X':
        print("Miercoles")
    case 'J':
        print("Jueves")
    case 'V':
        print("Viernes")
    case 'S':
        print("Sabado")
    case 'D':
        print("Domingo")
    case _:
        print("Error: Dia Equivovado")
```

```
Error: Dia Equivovado
```

Expresion ternaria

- Estrucutra de if-else en na línea
- Usar en casos sencillos

In [8]:
```
x = -7
resultado = x if x >= 0 else -x
print (resultado)
```

```
7
```

In [9]:
```
var = -4

if var >= 0:
    resultado = var
else :
    resultado = -var

print(resultado)
```

```
4
```

Compracaciones Concatenadas

- Se puden concatenar comparaciones en una misma expresiones:
  - *and*: Es true cuadno ambas son verdadero
  - *or*: Es true si al menos una es verdadero
  - *not*: inversion al valor de una expresión
- *Elif* para comparar tras un *if*

In [10]:
```
genero = 'Comedia'
year = 1990
```

```
if genero == 'Informativo' or genero == 'Accion':
    print ('Buen Libro')
elif year >= 2000 and year < 2010:
    print('Buena Decada')
else :
    print('Meh')
```

```
Meh
```

'==' vs 'is'

- '==' se usa para comprobar si el valor de las variables es igual
- '*is*' se usa para comparar si los objetos de las variables son el mismo

In [12]:
```
x = 1000
y = x

print(x is y)
print(x == y)

z=type(x)
print(z)
a=type(y)
print(a)

print(z==a)

print(id(x))
print(id(y))
```

```
True
True
<class 'int'>
<class 'int'>
True
1988618518416
1988618518416
```

In [14]:
```
p = [4,5,6,7]
q= [4,5,6,7]

r= p
print(p is q)
print(p is r)
print(p == q)

print(id(p))
print(id(q))
print(id(r))
```

```
False
True
True
1988622040576
1988622110912
1988622040576
```

Valor Booleano

- Todos los objetos en Python tienen valor boolena por naturaleza sea: True o False
- Cualquier número aparte del 0 o cualquier otro objeto no nulo tendrá el valor True
- El núemro 0, objetos vacios y el objeto None tienen el valor False

In [16]:
```
x =-10

y = 0

if a:
    print("a es True")
if b:
    print("b es False")
```

```
a es True
b es False
```

In [18]:
```
lsita = [4,5,6]

lista = None

if lista:
    print("Lista completa")
else :
    print("Lista Vacía")
```

```
Lista Vacía
```

3. Bucles

- Repetir un bloque de código
- La terminación del bucle depende del tipo de bucle
- Existen dos tipos: *while* y *for*

'*While*'

- Repetir de un bloque hasta que la condición se deje de cumplir (hasta que sea False)
- Si la condición evalua a False desde un principio, el bloqe no se ejecuta nunca.
- Precaución con los bucles infinitos

Formato general:
while test: - # Mientras se cumpla la condicion
statements - #Instrucciones a ejecutar

In [19]:
```
# Mostart los primeros 3 objetos de una lista

index = 0

num = [9,4,7,1,2]

while index < 3:
    print(num[index])
    index  = index+1
```

```
9
4
7
```

In [22]:
```
# contar y mostrar los números inferiores a 10
num = [33,3,9,21,1,7,12,10,8]

contador = 0
index = 0

while index < len(num):
    if num[index] <10:
        print(num[index])
        contador +=1
    index +=1
print('Contador: ' , contador)
print('Indice: ' , index)
```

```
3
9
1
7
8
Contador:  5
Indice:  9
```

In [23]:
```
nombre = 'Luis'

while nombre:
    print(nombre)
    nombre = nombre[1:]
```

```
Luis
uis
is
s
```

'*For*'

- Permite recorrer items de una sequencia o un ojeto iterable
- Funciona con strings, listas, tuplas, etc.
  Formato General:

for items in objeto: - # Asigna a los item del objeto a la varibale en cada iteración statemtns - #Instrucciones a ejecutar

In [24]:
```
peliculas =['The Godfather' , 'Top Gun' , 'Elf' , 'Die  Hard']

for peliculas in peliculas :
    print(peliculas)
```

```
The Godfather
Top Gun
Elf
Die  Hard
```

- Se puede usar tambien para iterar un numero establecido de veces

In [27]:
```
for i in range(10):
    print(i+1)
```

```
1
2
3
4
5
6
7
8
9
10
```

- *range* es util tambien con *len* ya que permite acceder a elementos de una secuencia por posicion

In [29]:
```
nombre= 'Luis'
for i in range(len(nombre)):
    print(nombre[i])
```

```
L
u
i
s
```

In [30]:
```
for letra in nombre:
    print(letra)
```

```
L
u
i
s
```

- iteracion de tuplas

In [52]:
```
tuplas = [(1, 2,4), (3, 4, 6), (5, 6,7)]
for a, b,c in tuplas:
    print(a, b,c)
```

```
1 2 4
3 4 6
5 6 7
```

- iteracion de diccionarios

In [54]:
```
diccionario = {'x':1, 'y':2, 'z':3}

for key in diccionario:
    print(f"{key} => {diccionario[key]}")
```

```
x => 1
y => 2
z => 3
```

- Para iterar pares clave-valor o solamente valores, se debe usar metodos *items* y *values*

In [57]:
```
for key, value in diccionario.items():
    print(f"{key} => {value}")
```

```
x => 1
y => 2
z => 3
```

In [58]:
```
for value in diccionario.values():
    print(value)
```

```
1
2
3
```

In [60]:
```
for p,q,r in [(4,5,6), (7,8,9)]:
    print(p,q,r)
```

```
4 5 6
7 8 9
```

3.1 Sentencias break, continue y else

Break y continue

- Solo hace sentido dentro de bucles

- *Break* permite terminar el bucle por completo
- *Continue* permite saltar a la siguiente parte de la iteración, siguiendo con el bucle
- Pueden aparecer en cualquier parte de un bucle, principalemente dentr de una sentencia condicional(if)

In [64]:
```
a_encontrar= 4.2
calificacion= [4.3, 2.5, 1.7, 4.2, 3.8, 3.3, 4.5]
for r in calificacion :
    print(r)
    if r == a_encontrar:
        print("Encontrado")
        break
print(r)
```

```
4.3
2.5
1.7
4.2
Encontrado
4.2
```

In [67]:
```
for i in range(10):
    if i%2 ==0 :
        continue
print(f'Numero Impar: {i}')
```

```
Numero Impar: 9
```

- Importante como la sentencia *ojo* puede ayudar a reducir el numero de niveles de anidamiento

In [69]:
```
# Sin continue el ejemplo seria

for i in range (10):
    if i%2!=0:
        print('Numero Impar: ' , end='')
        print(i)
```

```
Numero Impar: 1
Numero Impar: 3
Numero Impar: 5
Numero Impar: 7
Numero Impar: 9
```

In [71]:
```
for i in range (5):
    for a in range (2):
        print(f"{i} - {a}")
        if i == 3 and a ==0:
            break
```

```
0 - 0
0 - 1
1 - 0
1 - 1
2 - 0
2 - 1
3 - 0
4 - 0
4 - 1
```

Else

- Los bucles pueden contener una sentencia *else*
- Resulata poco intuitivo para muchos programadores ya que su sintaxis no esta presente en otros leguajes.
- Se ejecuta cuando el bucle finaliza, es decir cuando no finaliza a causa de un break

In [73]:
```
lista = [40,40,20,40]
por_encontrar=40

for e in lista:
    if e == por_encontrar:
        print("Encontrado")
        break
    else:
        print("No Encontrado")
```

```
Encontrado
```

In [74]:
```
lista = [10,30,50,20]
por_encontrar=20
found = False

for e in lista:
    if e == por_encontrar:
        found = True
        break
if found :
    print("Encontrado")
else :
    print("No Encontrado")
```

```
Encontrado
```

4. Comprensión de listas

- Permite contruir listas a traves de ejecucción rápida usando *for*
- Van entre corchetes, esto indica que se esta armando una lista

In [77]:
```
lista = []

for char in 'Esteban':
```

```
    lista.append(char)
print(lista)
```

```
['E', 's', 't', 'e', 'b', 'a', 'n']
```

- Comunmente el item de de la sentencia *for* aparecera en la expresión principal pero es opcional

In [79]:
```
[2 for _ in 'Esteban']
```

Out[79]:
```
[2, 2, 2, 2, 2, 2, 2]
```

In [ ]:
```

```

4.0.1 Versión Extendida

Se puede especificar un filtro para obtener únicamente los elementos que cumple cierta condición

Sintaxis:

`[<expression> for <item> in <iterable> if <condition>]`

In [80]:
```
# Numeros pares
[y for y in range(9) if y%2 == 0]
```

Out[80]:
```
[0, 2, 4, 6, 8]
```

- Comprensión de listas no son realmente requeridas, se puede escribir un bucle igual

In [81]:
```
pares= []

for x in range (9):
    if x%2==0 :
        pares.append(x)
print(pares)
```

```
[0, 2, 4, 6, 8]
```

4.0.2 Version Completa

La sentencia *if* de una comprensión de listas puede tambien puede obtener una expresión alternativa

Sintaxis: `[<expression_1> for <condition> else <expression_2> for <item> in <iterable>]`

In [82]:
```
[0 if x%2 == 0 else x for x in range(9)]
```

Out[82]: [0, 1, 0, 3, 0, 5, 0, 7, 0]

In [83]:
```
[x**2 if x > 2 else x for x in range(-4, 5) if x>0]
```

Out[83]: [1, 2, 9, 16]

In [84]:
```
lista = []

for x in range (-4,5) :
    if x > 0:
        if x > 2:
            lista.append(x**2)
        else:
            lista.append(x)
print(lista)
```

[1, 2, 9, 16]

In [85]:
```
[x**2 if x > 0 else 100 if x ==0 else x for x in range(-4,5)]
```

Out[85]: [-4, -3, -2, -1, 100, 1, 4, 9, 16]

In [88]:
```
lista = []

for x in range (-4,5):
    if x > 0:
        lista.append(x**2)
    elif x == 0:
        lista.append(100)
    else :
        lista.append(x)
print(lista)
```

[-4, -3, -2, -1, 100, 1, 4, 9, 16]

4.0.3 Anidamiento

Las comprensión de listas soportan anidamiento en sus expresiones

In [89]:
```
lista_1 = [1,2,3]
lista_2= [4,5]

[(x,y) for x in lista_1 for y in lista_2]
```

Out[89]: [(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)]

4.0.4 Pros y Contras

Pros (✅)	Contras (❌)
Reduce bucles anidados a una sola línea.	Más de 2 niveles se vuelve ilegible.

Pros (✅)	Contras (❌)
Clara en estructuras simples (matrices 2D).	Confusa si hay lógica compleja o múltiples condiciones.
Rápido de escribir en scripts exploratorios.	Imposible insertar prints; errores son difíciles de rastrear.
Ideal para flattening o productos cartesianos simples.	No admite elif; limita expresiones condicionales complejas.
Igual de rápido que bucles normales para casos simples.	Puede crear listas intermedias innecesarias si se abusa del anidamiento.
Pythonico cuando se mantiene simple.	Anti-pythonico si sacrifica claridad por brevedad

4.0.5 Otro tipo de compresiones

Comprension de diccionario

Usando llaves y delimitadores y una expresion (clave-valor)

In [90]:
```
d= {x : x*x for x in range(10)}

print(d)
print(type(d))
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
<class 'dict'>
```

In [92]:
```
productos = ['Banana' , 'Pera' , 'Aceitunas']
precios = [1.1, 1.5, 3]

compra = {c:v for c,v in zip (productos,precios)}
print(compra)
```

```
{'Banana': 1.1, 'Pera': 1.5, 'Aceitunas': 3}
```

In [93]:
```
productos = ['Banana' , 'Pera' , 'Aceitunas']
precios = [1.1, 1.5, 3]
for c in zip(productos , precios):
    print(type(c))
    print(c)
```

```
<class 'tuple'>
('Banana', 1.1)
<class 'tuple'>
('Pera', 1.5)
<class 'tuple'>
('Aceitunas', 3)
```

Compresion de conjuntos

Se usa llaves como delimitadores y con una expresión simple como con las comprensión de listas se obtine un set

In [94]:
```
a = {x for x in range(10)}
print(a)
print(type(a))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
<class 'set'>
```

Compresion de tupla

In [97]:
```
tupla = tuple(x for x in range(4))
print(tupla)
print(type(tupla))
```

```
(0, 1, 2, 3)
<class 'tuple'>
```

5. Excepciones

- Las excepciones son eventos que representan situaciones excepcionales e inesperadas.
- Alteran el flujo de ejecución.
- Python lanza excepciones automáticamente cuando se producen errores.
- El programador puede lanzar excepciones de manera explícita y también capturar excepciones para actuar como se crea conveniente.

In [103…
```
lista = [6, 1, 0, 5]
lista[4]
print('Código tras el error')
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[103], line 2
      1 lista = [6, 1, 0, 5]
----> 2 lista[4]
      3 print('Código tras el error')

IndexError: list index out of range
```

In [107…
```
lista = [6, 1, 0, 5]
i = 300
try:
    a = lista[i]
except IndexError:
    print('He capturado la excepción de tipo IndexError')
    a = 0
print('Código tras el bloque try')
print(a)
```

```
He capturado la excepción de tipo IndexError
Código tras el bloque try
0
```

- Normalmente al capturar excepciones, querremos ser lo mas especificos posibles, pero tambien se puede usar una sentencia *try-expect* que capture cualquier tipo de error

In [106…
```
try:
    4/0
except:
    print('He capturado la division por cero')
```

```
He capturado la division por cero
```

- Recoger la exepcion e imprimir el mensaje de error

In [108…
```
lista = [6, 1, 0, 5]
try:
    print(lista[4])
except Exception as e:
    print("Error = " + str(e))
print('Reacheable Code')
```

```
Error = list index out of range
Reacheable Code
```

Try/finally

- Con finally se puede especificar código que siempre se ejecute sin importar que se produzca la excepcion o no.
- Se usa para liberar uso de recursos

In [109…
```
lista = [6, 1, 0, 5]
try:
    a = lista[1]
except IndexError:
    print('Exception IndexError Captured')
finally:
    print('Bloque finally')
    b = lista[2]
print('Código tras el bloque try')
print(b)
```

```
Bloque finally
Código tras el bloque try
0
```

raise

- la sentencia *raise* se usa para lanzar excepciones de manera explicita

In [114…
```
lista = [6, 1, 0, 5]
indice = 4
try:
    if indice >= len(lista):
        raise IndexError('Índice fuera de rango')
```

```
except IndexError:
    print('Excepción de tipo IndexError capturada')
```

```
Excepción de tipo IndexError capturada
```

5. Ejercicios

1. Escribe un programa que calcule la suma de todos los elementos de una lista dada. La lista sólo puede contener elementos numéricos.
2. Dada una lista con elementos duplicados, escribir un programa que muestre una nueva lista con el mismo contenido que la primera pero sin elementos duplicados. Para este ejercicio, no puedes hacer uso de objetos de tipo 'Set'.
3. Escribe un programa que construya un diccionario que contenga un número (entre 1 y n) elementos de esta forma: (x, x*x). Ejemplo: para n = 5, el diccionario resultante sería {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
4. Escribe un programa que, dada una lista de palabras, compruebe si alguna empieza por 'a' y tiene más de 9 caracteres. Si dicha palabra existe, el programa deberá terminar en el momento exacto de encontrarla. El programa también debe mostrar un mensaje apropiado por pantalla que indique el éxito o el fracaso de la búsqueda. En caso de éxito, también se mostrará por pantalla la palabra encontrada.
5. Dada una lista L de números positivos, escribir un programa que muestre otra lista (ordenada) que contenga todo índice i que cumpla la siguiente condición: L[i] es múltiplo de 3. Por 13 ejemplo, dada la lista L = [3,5,13,12,1,9] el programa mostrará la lista [0,3,5] dado que L[0], L[3] y L[5] son, respectivamente, 3, 12 y 9, que son los únicos múltiplos de 3 que hay en L.
6. Dado un diccionario cuyos elementos son pares de tipo string y numérico (es decir, las claves son de tipo 'str' y los valores son de tipo 'int' o 'float'), escribe un programa que muestre por pantalla la clave cuyo valor asociado representa el valor númerico más alto de todo el diccionario. Por ejemplo, para el diccionario {'a': 4.3, 'b': 1, 'c': 7.8, 'd': -5} la respuesta sería 'c', dado que 7.8 es el valor más alto de los números 4.3, 1, 7.8 y -5.
7. Dada la lista a = [2, 4, 6, 8] y la lista b = [7, 11, 15, 22], escribe un programa que itere las listas a y b y multiplique cada elemento de a que sea mayor que 5 por cada elemento de b que sea menor que 14. El programa debe mostrar los resultados por pantalla.
8. Escribir un programa que pida un valor numérico X al usuario. Para ello podéis hacer uso de la función predefinida 'input'. El programa deberá mostrar por pantalla el resultado de la división 10/X. En caso de que el usuario introduzca valores no apropiados, el programa deberá gestionar correctamente las excepciones, por ejemplo, mostrando mensajes informativos por pantalla.
9. Escribir un programa que cree un diccionario cualquiera. Posteriormente, el programa pedirá al usuario (a través de la función predefinida 'input') que introduzca una clave del diccionario. Si la clave introducida es correcta (es decir, existe en el diccionario), el programa mostrará por pantalla el valor asociado a dicha clave. En caso de que la clave no exista, el programa gestionará de manera apropiada el error, por ejemplo, mostrando un mensaje informativo al usuario.

10. Escribe una list comprehension que construya una lista con los números enteros positivos de una lista de números dada. La lista original puede incluir números de tipo float, los cuales deben ser descartados.
11. Escribe una set comprehension que, dada una palabra, construya un conjunto que contenga las vocales de dicha palabra.
12. Escribe una list comprehension que construya una lista con todos los números del 0 al 50 que contengan el dígito 3. El resultado será: [3, 13, 23, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 43].
13. Escribe una dictionary comprehension que construya un diccionario que incluya los tamaños de cada palabra en una frase dada. Ejemplo: el resultado para la frase "Soy un ser humano" será {'Soy': 3, 'un': 2, 'ser': 3, 'humano': 6}
14. Escribe una list comprehension que construya una lista que incluya todos los números del 1 al 10 en orden. La primera mitad se mostrarán en formato numérico; la segunda mitad en texto. Es decir, el resultado será: [1, 2, 3, 4, 5, 'seis', 'siete', 'ocho', 'nueve', 'diez'].

6. Soluciones

1.

In [129…
```
lista = [4,7,5]
suma = 0
for total in lista:
    suma += total

print(suma)
```

```
16
```

2.

In [124…
```
lista = [6,5,2,2,6,4]

lista_sin_repetir = []

for i in lista:
    if i not in lista_sin_repetir:
        lista_sin_repetir.append(i)
print (lista_sin_repetir)
```

```
[6, 5, 2, 4]
```

3.

In [140…
```
n = 7
d = {x: x*x for x in range(1, n+1)}
print(d)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}
```

4.

In [138…
```python
palabras = ['arandano', 'automovilismo', 'anatomia', 'bala']
for p in palabras:
    if p[0] == 'a' and len(p) > 9:
        print(f"¡Encontrada: {p}")
        break
else:
    print("Ninguna cumple la condición.")
```

```
¡Encontrada: automovilismo
```

5.

In [141…
```python
L = [3, 5, 13, 12, 1, 9]
indices = []
i = 0
while i < len(L):
    if L[i] % 3 == 0:
        indices.append(i)
    i += 1
print(sorted(indices))
```

```
[0, 3, 5]
```

6.

In [152…
```python
d = {'a': 4.3, 'b': 1, 'c': 7.8, 'd': -5}
primera = True

for clave in d:
    valor = d[clave]
    if primera:
        clave_max = clave
        valor_max = valor
        primera = False
    elif valor > valor_max:
        clave_max = clave
        valor_max = valor

print(f"Valor Mas Alto: {valor_max}")
```

```
Valor Mas Alto: 7.8
```

7.

In [157…
```python
lista_a = [2,4,6,8]
lista_b = [7,11,15,22]

for x in lista_a :
    if x > 5:
```

```
        for y in lista_b:
            if y < 14:
                print(x*y)
```

```
42
66
56
88
```

8.

In [162…
```
try:
    x = int(input("Ingrese un valor: "))
    print(10 / x)
except ZeroDivisionError:
    print("No se puede dividir por cero.")
except ValueError:
    print("Valor no numérico.")
```

```
1.25
```

9.

In [166…
```
dicc = {'a': 1, 'b' : 4, 'c':22 , 'd': 100}
key = input("Buscar por clave:")
try:
    print("Valor:", dicc[key])
except KeyError:
    print(f"La clave '{key}' no existe.")
```

```
Valor: 1
```

10.

In [168…
```
nums = [3, 4.5, -2, 7, 8.6, 10]
positivos = [n for n in nums if n % 1 == 0 and n > 0]
print(positivos)
```

```
[3, 7, 10]
```

11.

In [171…
```
palabra = "teclado"
vocales = {c for c in palabra.lower() if c in "aeiou"}
print(vocales)  #.lower() se encarga de asegurar que las letras esten en minusculas
```

```
{'a', 'o', 'e'}
```

12.

In [172…
```
nums = [n for n in range(51)
        if 3 in {n % 10, n // 10}]
```

```
print(nums)
```

```
[3, 13, 23, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 43]
```

13.

In [173…
```
frase = "Hola Mundo!"
sizes = {palabra: len(palabra) for palabra in frase.split()}
print(sizes)
```

```
{'Hola': 4, 'Mundo!': 6}
```

14.

In [174…
```
textos = ['seis', 'siete', 'ocho', 'nueve', 'diez']
lista = [i if i <= 5 else textos[i-6] for i in range(1, 11)]
print(lista)
```

```
[1, 2, 3, 4, 5, 'seis', 'siete', 'ocho', 'nueve', 'diez']
```

Repositorio en Git Hub: https://github.com/EstebanSan140106/machine-2.git