SentenciasDeControlyFunciones

January 14, 2026

1 ITQ

2 PAO 2025-2026

Nombre: *Israel Pabon*

Fecha: *14/01/2026*

3 5.1 Sentencias de Control y Funciones

Israel Pabon

3.1 1 Estructuras de control

3.1.1 1.1 Selección (sentencias condicionales)

- Selección de una de varias alternativas en base a alguna condición.
- Indentación para estructurar el código.
- Importante el símbolo ':'.

[2]:
```
# # Asigno el bvalo uno a la variable x
x = int(input("Ingrese el valor de x"))
#Implemento una condicional para saber si x es mayor a 0
if x > 0:
    print('Valor positivo')
else:
```

```
    print('Valor no positivo')
```

```
Ingrese el valor de x 5

Valor positivo
```

- Las condiciones son expresiones que se evalúan a True o False.
- Operadores de comparación, lógicos, de identidad y de pertenencia vistos anteriormente.

[3]:
```
x = 3
y = [1, 2, 3]

print(x > 0)
print(x > 0 and x < 10)
print(x is not y)
print(x in y)
```

```
True
True
True
True
```

- Se pueden introducir más 'ramas' en sentencias condicionales a través de la palabra clave elif.

[4]:
```
x = -3

if x > 0:
    print('Valor positivo')
elif x == 0:
    print('Valor nulo')
else:
    print('Valor negativo')
```

```
Valor negativo
```

- Anidamiento.

[5]:
```
val = 120
if val > 0:
    if val < 100:
        print('Valor positivo')
    else:
        print('Valor muy positivo')
else:
    print('Valor negativo')
```

```
Valor muy positivo
```

3.2 2 Switch

[7]:
```
a = -6

match a:
    case _ if a > 0:
        print('positivo')
    case _ if a == 0:
        print('zero')
    case _ if a < 0:
        print('negativo')
```

```
negativo
```

[9]:
```
a = 'l' #week day

match a:
    case 'L':
        print("Lunes")
    case 'M':
        print("Martes")
    case 'X':
        print("Miercoles")
    case _:
        print("Wrong Day")
```

```
Wrong Day
```

3.2.1 Expresión ternaria

- Estructura if-else condensado en una línea.
- Se recomienda usar solo en casos sencillos.

[10]:
```
# Ejemplo: cálculo del valor absoluto de un número.
x = -3

resultado = x if x >= 0 else -x
print(resultado)
```

```
3
```

[11]:
```
val = -3

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

print(resultado)
```

```
3
```

3.2.2 Concatenación de comparaciones

- Se pueden concatenar comparaciones en una misma expresión:
  - and: true, si ambos son true.
  - or: true, si al menos uno es true.
  - not: inversión del valor de verdad de una expresión.
- 'elif' para comparar tras un 'if'.

Enlace a Tablas de Verdad.

[12]:
```
genero = 'Drama'
fecha_de_estreno = 1989

if genero == 'Comedia' or genero == 'Acción':
    print('Buena película!')
elif fecha_de_estreno >= 1990 and fecha_de_estreno < 2000:
    print('Buena década!')
else:
    print('Meh')
```

```
Meh
```

3.2.3 Comparación de variables: '==' vs 'is'

- '==' compara si el valor de las variables es el mismo.
- 'is' compara si los objetos en las variables son iguales (referencia).

[13]:
```
a = 1000
b = a

print(a is b)
print(a == b)

c=type(a)
print(c)
d=type(b)
print(d)

print(c==d)

print(id(a))
print(id(b))
```

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

```
1464317825264
1464317825264
```

[14]:
```python
a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(a is b)
print(a is c)
print(a == b)

print(id(a))
print(id(b))
print(id(c))
```

```
False
True
True
1464318060096
1464318060736
1464318060096
```

3.2.4 Valor booleano

- Todos los objetos en Python tiene inherentemente un valor booleano: *True* o *False*.
- Cualquier número distinto de 0 o cualquier objeto no vacío tienen valor *True*.
- El número 0, objetos vacíos y el objeto especial *None* tienen valor *False*.

[15]:
```python
a = -10              #True
b = None             #False

if a:
    print("a es True")
if not b:
    print("b es False")
```

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

[16]:
```python
c = [1,2,3]
                              # a c le reasigno None
c = None                      #0 es igual a vacio a nada

if c:
    print("Lista no vacía")
else:
    print("Lista vacía")
```

```
Lista vacía
```

3.3 3 Iteración (bucles)

- Repetición de un bloque de código.
- La terminación del bucle depende del tipo de bucle.
- Dos tipos: *while* y *for*. ### Bucle 'while'
- Repetición de un bloque de código hasta que se deje cumplir una expresión (es decir, hasta que una condición evalue a *False*).
- Si la condición evalua a *False* desde el principio, el bloque de código nunca se ejecuta.
- Cuidado con los bucles infinitos.

Formato general:

```
while test:          # Mientras se cumple la condición.

    statements       # Instrucciones a ejecutar.
```

[17]:
```
# Ejemplo: Mostrar los primeros 3 objetos de una lista
indice = 0
numeros = [9, 4, 7, 1, 2]

while indice < 3:
    print(numeros[indice])
    indice =indice+1
```

```
9
4
7
```

[18]:
```
# Ejemplo: contar y mostrar los números inferiores a 10.
numeros = [33, 3, 9, 21, 1, 7, 12, 10, 8]
contador = 0
indice = 0

while indice < len(numeros):
    if numeros[indice] < 10:
        print(numeros[indice])
        contador += 1
    indice += 1

print('Contador:', contador)
print('Indice:', indice)
```

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

[19]:
```
nombre = 'Pablo'

while nombre:               # Mientras 'nombre' no sea vacío
    print(nombre)
    nombre = nombre[1:]
```

```
Pablo
ablo
blo
lo
o
```

[20]:
```
# #bucle infinito
i = 0

while i < 10:
    print(i)
    i += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

3.3.1 Bucle 'for'

- Permite recorrer los items de una sequencia o un objeto *iterable*.
- Funciona en strings, listas, tuplas, etc.

Formato general:

```
for item in objeto:          # Asigna los items del objeto a la variable item en cada iteració
    statements               # Instrucciones a ejecutar
```

[21]:
```
peliculas = ['Matrix', 'The purge', 'Avatar', 'Star Wars']

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

```
Matrix
The purge
Avatar
Star Wars
```

- También se usa para iterar un número preestablecido de veces (*counted loops*):

[22]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

- *range* es útil en combinación con len porque permite acceder a los elementos de una *secuencia* por posición.

[24]:
```
nombre = 'Pablo'

for i in range(len(nombre)):
    print(nombre[i])
```

```
P
a
b
l
o
```

[25]:
```
for letra in nombre:
    print(letra)
```

```
P
a
b
l
o
```

- Iteración de tuplas:

[28]:
```
tuplas = [(1, 2,4), (3, 4, 2), (5, 6,1)]
for a, b,c in tuplas:
    print(a, b,c)
```

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

- Iteración de diccionarios. Se iteran las claves:

[29]:
```
diccionario = {'a': 1, 'b': 2, 'c': 3}

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

```
a => 1
b => 2
c => 3
```

- Para iterar los pares (clave-valor) o únicamente los valores, se deben usar los métodos *items* y *values*.

[30]:
```
for key, value in diccionario.items():
    print(f"{key} => {value}")
```

```
a => 1
b => 2
c => 3
```

[31]:
```
for value in diccionario.values():
    print(value)
```

```
1
2
3
```

- La variable *item* en la cabecera del *for* puede ser cualquier expresión que sea válida como parte izquierda de una asignación convencional.

[32]:
```
for a, b, c in [(1, 2, 3), (4, 5, 6)]:
    print(a, b, c)
```

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

3.3.2 3.1 Las sentencias break, continue y else

3.3.3 Break y continue

- Solo tienen sentido dentro de bucles.
- *Break* permite terminar el bucle por completo.
- *Continue* permite saltar a la siguiente iteración, continuando con el bucle.
- Pueden aparecer en cualquier parte de un bucle, pero normalmente aparecen dentro de sentencias condicionales (if).

[33]:
```
# Ejemplo: break
rating_to_find = 4.2
movie_ratings = [4.3, 2.5, 1.7, 4.2, 3.8, 3.3, 4.5]

for rating in movie_ratings:
    print(rating)
```

```
    if rating == rating_to_find:
        print("Found")
        break

print(rating)
```

```
4.3
2.5
1.7
4.2
Found
4.2
```

[35]:
```
for i in range(10):
    if i % 2 == 0:
        continue           # Si el número es par, salta a la siguiente␣
 ↪iteración␣

print(f'Add number {i}')
```

```
Add number 9
```

- Observa como la sentencia continue te puede ayudar a reducir el número de niveles de anidamiento.
- Sin *continue* el anterior ejemplo sería:

[36]:
```
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
```

[37]:
```
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
```

3.3.4 Else

- Los bucles pueden tener una sensencia *else*.
- Resulta poco intuitiva para muchos programadores porque esta sintaxis no existe en otros lenguajes.
- Se ejecuta cuando el bucle termina con normalidad; es decir, cuando no termina a causa de un *break*.

[38]:
```
lista = [60,60,30,60]
element_to_find = 60

for element in lista:
    if element == element_to_find:
        print("found")
        break
else:
    print("not found")
```

```
found
```

[40]:
```
# with flag
lista = [10,30,50,40]
element_to_find = 30
found = False

for element in lista:
    if element == element_to_find:
        found = True
        break
if found:
    print("Dato encontrado")
else:
    print("Dato no encontrado")
```

```
Dato encontrado
```

3.4 4 List Comprehensions

- Permiten construir listas a través de la ejecución repetida (sentencia *for*) de una expressión para cada *item* de un objeto *iterable*.
- Van entre '[' y ']'. Esto es indicativo de que estamos construyendo una lista.

Sintaxis:

```
[<expression> for <item> in <iterable>]/
```

Ejemplo: repetir los carácteres de un string.

[41]:
```
[char*3 for char in "Enrique"]
```

[41]:
```
['EEE', 'nnn', 'rrr', 'iii', 'qqq', 'uuu', 'eee']
```

[42]:
```
lista = []

for char in 'Enrique':
    lista.append(char)

print(lista)
```

```
['E', 'n', 'r', 'i', 'q', 'u', 'e']
```

Comúnmente, el item de la sentencia *for* aparecerá en la expresión principal, pero eso no es obligatorio.

[43]:
```
[2 for _ in 'Enrique']
```

[43]:
```
[2, 2, 2, 2, 2, 2, 2]
```

3.4.1 4.0.1 Versión extendida

Se puede especificar un filtro (sentencia if) para obtener únicamente los elementos que cumplan cierta condición.

Sintaxis:

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

Ejemplo: obtener números pares:

[44]:
```
[y for y in range(9) if y % 2 == 0]
```

[44]:
```
[0, 2, 4, 6, 8]
```

- Las list comprehension no son realmente requeridas, ya que siempre podemos escribir un bucle equivalente.

[45]:
```
pares = []

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

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

3.4.2 4.0.2 Versión completa

La sentencia *if* de una list comprehension también puede contener una expresión alternativa.

Sintaxis:

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

Ejemplo: poner a cero los números pares.

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

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

[ ]:
```
lista = []

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

print(lista)
```

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

[ ]:
```
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)
```

3.4.3 4.0.3 Anidamiento

Las list comprehensions soportan anidamiento en sus expresiones.

Ejemplo: bucle anidado para obtener una lista de tuplas que combinan los elementos de dos listas-dadas.

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

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

3.4.4 4.0.4 Pros y contras

- Principales ventajas de las comprehensions en comparación a un bucle convencional:
  - Expresión compacta y legible, si estás familiarizado con la sintaxis.
  - Mejor rendimiento.

- Desventaja: no escalan bien. Una list comprehension se puede convertir rápidamente en una expresión difícil de entender.

3.4.5 4.0.5 Otros tipos de comprehensions

3.4.6 Dictionary comprehensions

Usando '{' y '}' como delimitadores y una expresión 'clave : valor', se obtiene un diccionario en lugar de una lista.

Ejemplo: diccionario donde cada valor es el cuadrado de la clave.

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

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

[ ]:
```
: items = ['Banana', 'Pear', 'Olives']
price = [1.1, 1.4, 2.4]
shopping = {k:v for k,v in zip(items,price)}

print(shopping)
```

[ ]:
```
items = ['Banana', 'Pear', 'Olives']
price = [1.1, 1.4, 2.4]

for k in zip(items, price):
    print(type(k))
    print(k)
```

3.4.7 Set comprehensions

Usando '{' y '}' como delimitadores y una expresión simple (al igual que en las list comprehensions), se obtiene un conjunto.

Ejemplo: conjunto que incluye los 10 primeros números naturales.

[ ]:
```
c = {x for x in range(10)}

print(c)
print(type(c))
```

3.4.8 Tuple comprehensions

[ ]:
```
tupla = tuple(x for x in range(4))

print(tupla)
```

3.5 5 Excepciones

- Las excepciones son eventos que representan situaciones excepcionales.
- Alteran el flujo de ejecución convencional.
- 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.

3.5.1 Try/except

- Permite capturar excepciones y actuar en consecuencia.

Ejemplo: error de acceso fuera de rango.

[ ]:
```
lista = [6, 1, 0, 5]
lista[4]

print('Código tras el error')
```

[ ]:
```
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)
```

- Normalmente, al capturar excepciones, querremos ser lo más específicos posible, pero también se puedes usar una sentencia *try-except* que capture cualquier error.

[ ]:
```
try:
    4/0
except:
    print('He capturado la división por cero')
```

[ ]:
```
- Recoger la excepción e imprimir el mensaje de error.
```

[ ]:
```
lista = [6, 1, 0, 5]

try:
    print(lista[4])
except Exception as e:
    print("Error = " + str(e))

print('Reacheable Code')
```

3.5.2 Try/finally

- A través de *finally* podemos especificar código que queremos que se ejecute siempre (independientemente de si se produce la excepción o no).
- Se suele usar para liberar recursos.

[ ]:
```
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)
```

3.5.3 Raise

- La sentencia raise nos permite lanzar excepciones de manera explícita.

[ ]:
```
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')
```

3.6 5.1 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) de 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

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'].

https://github.com/Flaquencio/machinelearning2026/blob/main/SentenciasDeControlyFunciones.ipynb

3.7 6 Soluciones

[47]:
```
# Ejercicio 1
lista= [1, 2, 3, 4, 5, 6, 7, 8, 9]

suma = sum(lista)
print(suma)
```

```
45
```

[50]:
```
# Ejercicio 2
lista = [1, 2, 3, 2, 4, 1, 5, 3, 6, 4, 1, 2]
lista_no_duplicados = []

for elemento in lista:
    if elemento not in lista_no_duplicados:
        lista_no_duplicados.append(elemento)

print(lista)
print(lista_no_duplicados)
```

```
[1, 2, 3, 2, 4, 1, 5, 3, 6, 4, 1, 2]
[1, 2, 3, 4, 5, 6]
```

[52]:
```
# Ejercicio 3
n = 6

diccionario = {}
for x in range(1, n + 1):
    diccionario[x] = x * x

print(diccionario)
```

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

[54]:
```
# Ejercicio 4
palabras = ["telefono", "conputadora", "televisor", "tablet", "articulacion",␣
 ↪"mouse", "parlante", "teclado", "armatoste"]

encontrada = False

for palabra in palabras:
    if palabra.startswith("a") and len(palabra) > 9:
        print("Palabra Encontrada es", palabra)
        encontrada = True
        break
else:
    print("Palabra no Encontrada")
```

```
Encontramos la palabra articulacion
```

[59]:
```
# Ejercicio 5
L = [3, 5, 13, 12, 1, 9, 21, 24]
multiplos = []

for i in range(len(L)):
    if L[i] % 3 == 0:
        multiplos.append(i)
```

```
print(multiplos)
```

```
[0, 3, 5, 6, 7]
```

```
[ ]:
```