Estructuras_Control

January 14, 2026

INSTITUTO SUPERIOR TECNOLÓGICO QUITO

Formamos tu PROPÓSITO DE VIDA

Estructuras de Control en Python

Nombre: Alejandro Bedoya

0.1 Estructuras de Control

0.1.1 Selección (Sentencias Condicionales)

- Se puede seleccionar una o varias alternativas en base a la condición
- Necesitas tener la identación para estructurar el código
- Es importante colocar los dos puntos ":" al final de la sentencia

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

[2]:
```
#Instrucción
#1) Crear una variable que pueda almacenar el número del usuario
#2) Implemento una condicional con if para saber si el número del usuario
#es mayor a 0, igual a 0 o menor a 0 con un mensaje

#Parseamos a entero porque el input da por default un string
num_user = int(input("Escribe tu número para la validación"))
#Bucle if que dirá si es positivo o no
if num_user > 0:
    #mensaje
    print("El número es positivo")
#En caso que el número sea cero
elif num_user == 0:
    #mensaje
    print("El número es cero")
#En caso que el número no sea positivo o cero
else:
    #mensaje
    print("El número es negativo")
```

```
Escribe tu número para la validación 0

El número es cero
```

[4]:
```
#True o False
numero = 3
lista = [1,2,3]
print(numero>0) # numero es mayor a 0? Verdadero
print(numero > 0 and numero < 10) #numero es mayor a 0 y numero es menor a 10?␣
 ↪Verdadero
print(numero is not lista)# numero no es la lista? Verdadero
print(numero in lista) #numero está dentro de la lista? Verdadero
```

```
True
True
True
True
```

0.1.2 Anidamiento

[5]:
```
numero_anidado = 120
#Si el número es mayor a 0, entramos al siguiente bucle
if numero_anidado > 0:
    #Si el número es menor a 100, entonces será positivo
    if numero_anidado < 100:
        print("El número es positivo")
    #Si no es menor a 100, entonces el número va a sobrepasarlo
```

```
    else:
        print("El número sobrepasa el 100, POSITIVO WOAAA")
#Si el número no es mayor a 0, entonces el número es negativo
else:
    print("El número es negativo")
```

```
El número sobrepasa el 100, POSITIVO WOAAA
```

0.2 Switch

Para el switch nosotros utilizaremos la palabra "match" y case. Si queremos recibir cualquier valor utilizamos el _ acompañado de un condicional como if.

[8]:
```
valor_switch = int(input("Escribe un número"))

match valor_switch:
    #El valor dado (cualquiera del usuario) es menor a 0
    case _ if valor_switch > 0:
        #Mensaje de positivo
        print("positivo")
    #El valor dado es igual a cero
    case _ if valor_switch == 0:
        #Mensaje de cero
        print("cero")
    #El valor dado es menor a cero
    case _ if valor_switch < 0:
        #Mensaje de negativo
        print("negativo")
```

```
Escribe un número 0

cero
```

Por otro lado, si tenemos una condición en específico como "L" y si ese caso sucede entonces imprime "hola"

[10]:
```
#valor_switch2 = "Monday" -- saldrá Lunes
#valor_switch2 = "Friday" -- saldrá Viernes
#valor_switch2 = "Saturday" -- saldrá Sábado
valor_switch2 = "Wednesday"

match valor_switch2 :
    case "Friday":
        print("Viernes")
    case "Monday":
        print("Lunes")
    case "Saturday" :
        print("Sábado")
    case _ :
        print("Día no contado")
```

```
Día no contado
```

0.2.1 Expresiones ternarias (acotación de condicional)

Es un if else condensado en una sola linea de código y se recomienda usar solo en casos sencillos

[15]:
```
#INSTRUCCIÓN
#Calcular el valor absoluto de un número

numero_no_absoluto = -12

#Condicional Normal
if numero_no_absoluto >= 0:
    print(numero_no_absoluto)
else:
    numero_no_absoluto*= -1
    print(numero_no_absoluto)

#Condicional ternaria
#variable = resultado if condicional else condicional
transformacion_absoluto = numero_no_absoluto if numero_no_absoluto >= 0 else␣
 ↪-numero_no_absoluto
print (transformacion_absoluto)
```

```
12
12
```

0.2.2 Concatenación de Comparadores

- Se pueden concatenar comparaciones en una misma expresion
  - *and* : Verdadero en caso que ambos sean **True**
  - *or* : Al menos uno debe ser verdadero para que salga **True**
  - *not* : Inversión del valor de verdad de una expresión
- "elif" para comprarar nuevamente con una condición después de un if

Enlaces de Tablas de Verdad

[17]:
```
genero = "Drama"
fecha_estreno = 2015
#Si genero es Comedia o Drama
if genero == "Comedia" or genero == "Acción":
    #Imprime el mensaje
    print("Es una buena peli")
#entonces si la fecha de estreno es mayor o igual al 2000 y menor o igual a␣
 ↪2014
elif fecha_estreno >= 2000 and fecha_estreno <= 2014:
    #Imprime el mensaje
    print("Buen año para la peli")
#Si no se cumple ninguna condición
```

```
else:
    #Sale este mensaje
    print("Ya nada")
```

```
Ya nada
```

0.2.3 Comparación de variables: "==" vs "is"

El **"=="** sirve para comparar si el valor de las variables es igual, pero el **"is"** compara si los objetos de las variables son iguales según su referencia.

Es decir, el == se usa para comparar valores y el **is** se usa para comparar el id

[19]:
```
#Variables
numero_uno = 1000
numero_dos = numero_uno
#Veamos el id, debería ser el mismo
print(id(numero_uno))
print(id(numero_dos))

#Como el is lo usamos para ver si los id son iguales, debe salir True
print(numero_uno is numero_dos)
#Como el == lo usamos para ver si los números son iguales, también debe ser True
print(numero_uno == numero_dos)
```

```
1232602949488
1232602949488
True
True
```

[20]:
```
#Variables
a = 12
b = a
#Decimos que c será el mismo tipo de dato que a
c=type(a)
print(c)
#Decimos que d será el mismo tipo de dato que b
d=type(b)
print(d)

#entero es igual a entero? verdadero
#Como ambos son entero, entonces son iguales en dato
print(c==d)
```

```
<class 'int'>
<class 'int'>
True
140708772623624
140708772623624
```

0.2.4 Valor Booleano

- Todos los objetos en Python tienen de manera default un valor booleano: *True* o *False*
- Cualquier número distinto de 0 o cualquier objeto no vacío tienen valor de *True*
- El número 0, objetos vacíos y el objeto "None" tienen por defecto *False*

[23]:
```
positivo = -50 #Saldrá true porque tiene un valor
negativo = None #Saldrá false porque es None

#Si es la variable positivo entonces
if positivo :
    #Saldrá el mensaje
    print("Positivo es True")

#Si no es la variable negativo entonces
#Si no es None entonces
if not negativo:
    #Saldrá el mensaje
    print("Negativo es False")

valor_lista = [1,2,3]

#a la variable "valor_lista" le reasigno el objeto "None"
valor_lista = None
#Si la lista tiene un valor entonces
if valor_lista:
    #mensaje
    print("Esta lista no está vacía")
#Si no tiene un mensaje (Como 0, None o valor vacío)
else:
    #mensaje
    print("La lista está vacía")
```

```
Positivo es True
Negativo es False
La lista está vacía
```

0.3 Iteración (Bucles)

Es la **repetición de un bloque de código**, la finalización del bloque depende de su tipo pero generalmente es cuando su condición es **False**.

Existen dos tipos: ***while*** y ***for***

0.3.1 Bucle while

Su función basa en la **repetición del código hasta que se deje de cumplir una condición**, si la condición se hace *False* desde el inicio, el while nunca se va a ejecutar. Es necesario tener **cuidado con los bucles infinitos**, siempre cerrarlos o siempre tener alguna expresión para alcanzar la condición

[27]:
```
#Formato
#while condicional:   --- mientras se cumple la condición, el bucle se repetirá
 #expresiones  ---- Instrucciones para ejecutar

#INSTRUCCIÓN: Mostrar los 3 primeros objetos de una lista
#Lista
lista_bucle = [4,5,6,7,1,2,3,10]
#Indice para poder ir recorriendolo
indice = 0
#Mientras que el índice sea menor que 3, seguirá funcionando
while indice < 3:
    #Imprimimos el número de la lista donde se encuentre el índice
    print(lista_bucle[indice])
    #Creamos la expresión para poder sumar el indice y así no se haga infinito
    indice += 1
    #Cuando el índice llegue a 3, el bucle cortará y no dará el 4to porque pasa␣
 ↪su condición
```

```
4
5
6
```

[35]:
```
#INSTRUCCIÓN: Contar y mostrar los números inferiores a 10

#Creamos la lista
numeros = [33,23,4,5,6,7,12,144,255,1,2]
#Creamos un contador porque necesitamos contar de uno a uno los números
contador = 0
#Creamos el índice porque necesitamos recorrer la lista
indice = 0

#Mientras que el índice sea menor a la cantidad de números que hay en la lista
while indice < len(numeros):
    # Si el número que está en el índice es menor a 10 entonces:
    if numeros[indice] < 10:
        #Mostrará el número
        print(numeros[indice])
        #Aumentará el contador
        contador += 1
    #El indice está fuera del if porque necesita ir aumentando para cerrar el␣
 ↪while
    indice += 1

#El contador es solo msotrar cuántos números hay con esas condiciones
print('Contador:', contador)
#Muestra cuántos números había en el total de la lista
print('Indice:', indice)
```

```
4
5
6
7
1
2
Contador: 6
Indice: 11
```

[34]:
```python
#INSTRUCCIÓN: Mostrar los números inferiores a 30
#EJERCICIO PARA ENTENDER

numeros_ejercicio = [1,2,3,4,5,6,7,8,22,33,44,55,66,77,88,99,1]
indice = 0
contador= 0
#Mientras que el indice sea menor a la cantidad total de números de la lista
while indice < len(numeros_ejercicio):
    #Si el número dentro del índice es menor a 30
    if numeros_ejercicio[indice] < 30:
        #se mostrará el número
        print(numeros_ejercicio[indice])
    #Suma el índice para que recorra todos y no sea un infinite loop
    indice+=1
```

```
1
2
3
4
5
6
7
8
22
1
```

[37]:
```python
nombre = "Adriel"
#Mientras exista un nombre
while nombre:
    #Imprime el nombre
    print(nombre)
    #Toma desde la posición uno hasta el final
    #A medida que el bucle vaya generandose, el índice va creciendo
    #Evitamos el infinite loop con esto
    nombre = nombre[1:]
```

```
Adriel
driel
riel
iel
```

```
el
l
```

0.3.2 Bucle for

- Permite recorrer los items de una secuencia o un objeto iterable
- Funciona con strings, tuplas, listas, sets, etc

El formato de este bucle es: `for variable_que_almacena in condicional:`

```
statements
```

[38]:
```
peliculas = ["Matrix", "La purga", "Avatar" , "SAO"]

#Recorrera toda la lista y almacenará los valores en "pelicula_variable"
for pelicula_variable in peliculas:
    print(pelicula_variable)
```

```
Matrix
La purga
Avatar
SAO
```

[42]:
```
#También se puede iterar un número pre-establecido de veces (se los conoce como counted loops)
#Recorrera los números que estén dentro del rango del 0 al 3 (recuerda que el último es exclusivo)
#y los almacena en la variable "numeros"
for numeros in range(4):
    #imprime los números dentro del rango
    print(numeros)
```

```
0
1
2
3
```

[45]:
```
"""
"range()" es útil en combinación con "len()" porque permite acceder
a los elementos de una secuencia
"""
nombre = "Jonnath"
#Recorreremos todo el string mediante el número total de letras en el nombre
for letra_nombre in range(len(nombre)):
    #imprimiremos letra por letra del nombre
    print(nombre[letra_nombre])


#Es solo para separar los ejercicios xd
```

```
print("-------------------------------------")

#Si la letra está en el nombre, entonces
for letra in nombre:
    #imprimira las letras
    print(letra)
```

```
J
o
n
n
a
t
h
-------------------------------------
J
o
n
n
a
t
h
```

[52]:
```
#Iteración de tuplas

#Creamos la tupla con 3 valores
tupla_ejercicio = [(1,2,4), (3,4,7) , (5,6,1)]
#mediante índice, ira juntando y necesitan las tuplas tener el mismo valor
for a, b, c in tupla_ejercicio:
    #imprime el primero de cada uno
    print(a, b, c)
```

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

[54]:
```
diccionario = {'a': 1, 'b': 2, 'c': 3}

for key in diccionario:
    print(f" para {key} hay {diccionario[key]}")
```

```
 para a hay 1
 para b hay 2
 para c hay 3
```

[59]:
```
#Para iterar los pares o únicamente los valores, se deben usar métodos items
#Se imprime la clave y el valor
for key, value in diccionario.items():
    print(f"{key} => {value}")
```

```
print("----------------------")

#Se imprime solo el valor
for value in diccionario.values():
    print(value)

print("----------------------")

#Se puede iterar tranquilamente por índice 1,4 ; 2,5 ; 3,6
for a, b, c in [(1, 2, 3), (4, 5, 6)]:
    print(a, b, c)
```

```
a => 1
b => 2
c => 3
----------------------
1
2
3
----------------------
1 2 3
4 5 6
```

0.3.3 Sentencias break, continue, else

Break y Continue

- Solo tienen sentido dentro de los bucles
- Break permite terminar el bucle por completo
- Continue permite saltar a la siguiente iteración sin que se rompa el bucle
- Pueden aparecer en cualquier parte del bucle, pero normalmente aparecen dentro de las sentencias condicionales (como el if)

[64]:
```
#Ejemplo de break

calificacion_minima = 7
estudiantes_notas = [10, 4, 4, 4, 2.5, 1.9, 7 , 2.3, 7 , 10 , 2.3 , 10]

#Recorrera toda la lista
for calificaciones in estudiantes_notas:
    #Imprimira todas las calificaciones
    print(calificaciones)
    #Si en las calificaciones, se encuentre la nota mínima
    if calificaciones == calificacion_minima:
        #Imprime el mensaje solo con el primer 7 que haya
        print("Primera nota mínima de 7")
        #Lo hace gracias al break
```

```
        break
#Imprimimos fuera del bucle para ver el total
print(calificaciones)
```

```
10
4
4
4
2.5
1.9
7
Primera nota mínima de 7
7
```

[66]:
```
#Recorre todos los números del 1 al 10
for indice in range(11):
    #Si el número que está dentro del índice es par, entonces:
    if indice % 2 == 0:
        #Continuara el bucle sin tener en cuenta a ese número
        #Literalmente se salta de ese número
        continue
    #Mostramos el índice
    print(f"add number {indice}")
```

```
add number 1
add number 3
add number 5
add number 7
add number 9
```

[71]:
```
#El continue puede ayudar a reducir el número de niveles de anidamiento
#Recorremos todos los números hasta el 9
for index in range (10):
    #Si el index no es par, entonces
    if index % 2 != 0:
        #mustra mensaje y el número seguido
        print('Numero impar:', end=' ')
        #numero que está dentro del índice
        print(index)
```

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

[74]:
```
#Recorremos todos los números del 0 al 4
for indice in range(5):
    #recorremos otros números del 0 al 1
```

```
    for otro_num in range(2):
        #imprimimos la expresión del indice menos el otro número
        print(f"{indice} - {otro_num}")
        #Si el indice es 3 y el otro número llega a 0, entonces
        if indice == 3 and otro_num == 0:
            #se termina el bucle
            break
```

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

Else

- Los bucles pueden tener una secuencia else
- Esta sintaxis es única de Python
- Se ejecuta cuando el bucle termina con normalidad, es decir, cuando no termina a causa de un break

[78]:
```
lista = [60,60,30,60]
num_encontrar = 60
#Recorremos toda la lista y la almacenamos en la variable "numeros"
for numeros in lista:
    #Si la variable "numeros" es igual al numero para encontrar, entonces
    if numeros == num_encontrar:
        #Imprime que lo encontró
        print("Número encontrado")
        #termina el bucle
        break
#Si no lo encuentra
else:
    #mensaje
    print("No pude encontrar el número")
```

```
Número encontrado
```

[82]:
```
lista = [10,22,33,44,55,66,77]
encontrar_num = 44
encontrado = False #Colocamos por defecto que no se ha encontrado

#Recorremos toda la lista y almacenamos en la variable "numeros"
for numeros in lista:
```

```
    #Si en numeros está el número que queremos
    if numeros == encontrar_num:
        #el estado de encontrado cambia a True
        encontrado = True
        #Se termina el bucle
        break
#Si encontrado es Vedadero
if encontrado:
    #Mensaje
    print("Dato Encontrado")
#Si no es verdadero
else:
    #Mensaje
    print("Dato no encontrado")
```

```
Dato Encontrado
```

0.4 Comprensión de Listas

Permiten construir listas a través de la ejecución repetida (for) de una expresión para cada item de un objeto iterable

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

[84]:
```
#Ejemplo: repetir los carácteres de un string
[caracter for caracter in "Alejandro"]
```

[84]:
```
['A', 'l', 'e', 'j', 'a', 'n', 'd', 'r', 'o']
```

[89]:
```
lista = []
#Hacemos un for para que lea por índice cada letra del string
for char in "Adriel":
    #vamos a insertar cada letra dentro de la lista vacía
    lista.append(char)
#Imprimimos lista para ver el resultado
print(lista)
```

```
['A', 'd', 'r', 'i', 'e', 'l']
```

[91]:
```
#Por cada caracter que tenga el string, añadimos un dos a la lista en vez de la␣
 ↪letra
[2 for _ in 'Enrique']
```

[91]:
```
[2, 2, 2, 2, 2, 2, 2]
```

0.4.1 Versión Extendida

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

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

[97]:
```
[y for y in range(9) if y % 2 == 0]
```

[97]:
```
[0, 2, 4, 6, 8]
```

[106]:
```
#Basicamente está haciendo esto
#recorremos del 0 al 8 y lo guardamos en y
lista_vacia = []

for y in range(9):
    #si y es par, entonces
    if y % 2 == 0:
        #Añadimos a la lista vacía
        lista_vacia.append(y)
#Imprimimos la nueva lista
print(lista_vacia)
```

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

[103]:
```
#Creamos una lista vacía
pares = []
#recorremos del 0 al 8 y guardamos en "x"
for x in range(9):
    #si el número de x es par, entonces
    if x % 2 == 0:
        #lo añadimos a la lista vacía
        pares.append(x)
print(pares)
```

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

0.4.2 Versión Completa

la sentencia if de una comprensión de lista también puede contener una expresión alternativa

Sintaxis `[<expression_1> if <condition> else <expression_2>`

[114]:
```
[0 if x % 2 == 0 else x for x in range(9)]

#En cristiano sería basicamente
#Creamos una lista vacía
lista_traducida = []
#creamos la variable x para almacenar los números desde 0 a 8
for x in range(9):
    #si x es par, entonces
    if x % 2 == 0:
        #en la lista se pondrá un 0
        lista_traducida.append(0)
```

```
    #sino
    else:
        #la lista tendrá el número original
        lista_traducida.append(x)
#vemos la nueva lista
print(lista_traducida)
```

```
[0, 1, 0, 3, 0, 5, 0, 7, 0]
```

[108]:
```
[x**2 if x > 2 else x for x in range(-4,5) if x > 0]
```

[108]:
```
[1, 2, 9, 16]
```

[111]:
```
#Creamos una lista vacía
lista = []
# recorremos el rango de -4 a 4 y se almacena en "xd"
for xd in range(-4,5):
    #si xd es mayor a 0 entonces
    if xd > 0:
        #si xd es mayor a dos entonces
        if xd > 2:
            #añade el xd elevado al cuadrado
            lista.append(xd**2)
        #sino
        else:
            #añade solo el número de xd
            lista.append(xd)
#muestra la lista
print(lista)
```

```
[1, 2, 9, 16]
```

0.4.3 Anidamiento

Las listas comprendidas soportan anidamiento en sus expresiones

[120]:
```
"""
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]
#leer de derecha a izquierda para hacer el bucle tradicional xd
[(x, y) for x in lista_1 for y in lista_2]
```

[120]:
```
[(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)]
```

[121]:
```
#En cristiano
lista_1 = [1,2,3]
lista_2 = [4,5]
#Creamos una lista vacía
lista_completa = []
#recorremos lista 2 y guardamos en "y"
for y in lista_2:
    #recorremos lista 1 y guardamos en "x"
    for x in lista_1:
        #Juntamos las listas en nuestra nueva lista creada y la metemos
        #por defecto, se pondrán en tuplas según su índice 1 con 1, 2 con 2 y␣
 ↪así
        lista_completa.append((x,y))

print(lista_completa)
```

```
[(1, 4), (2, 4), (3, 4), (1, 5), (2, 5), (3, 5)]
```

0.5 Otros tipos de Compresiones en Python

0.5.1 Comprensión de Diccionarios

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

[123]:
```
#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))
```

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

[128]:
```
#En cristiano
#Creamos un diccionario vacío
d = {}
#Recorremos del 0 al 9 y guardamos en "x"
for x in range(10):
    #según el índice de el diccionario, este multiplicará su valor por el mismo
    d[x] = x * x

print(d)
```

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

[129]:
```
items = ['Banana', 'Pear', 'Olives']
price = [1.1, 1.4, 2.4]

shopping = {k:v for k,v in zip(items,price)}
```

```
print(shopping)
```

```
{'Banana': 1.1, 'Pear': 1.4, 'Olives': 2.4}
```

[130]:
```
#En cristiano
items = ['Banana', 'Pear', 'Olives']
price = [1.1, 1.4, 2.4]

#Creamos el diccionario vacío
shopping = {}

#Haremos dos cosas
#1) Creamos dos variables para almacenar dos listas
#2) Juntamos las listas por clave y valor (items, price)
for k , v in zip(items, price):
    #hacemos el clave y valor
    shopping[k] = v
#imprimimos
print(shopping)
```

```
{'Banana': 1.1, 'Pear': 1.4, 'Olives': 2.4}
```

[131]:
```
items = ['Banana', 'Pear', 'Olives']
price = [1.1, 1.4, 2.4]

#1) Creamos dos variables para almacenar dos listas
#2) Juntamos las listas por clave y valor (items, price)
for k in zip(items, price):
    #Imprimimos el tipo de valor que tiene k
    print(type(k))
    #imprimimos el resultado de k
    print(k)
#Se hace tupla porque no específicamos un nuevo diccionario y el valor
#queda en el oxígeno, cuando pasa eso, por default se hacen tuplas
```

```
<class 'tuple'>
('Banana', 1.1)
<class 'tuple'>
('Pear', 1.4)
<class 'tuple'>
('Olives', 2.4)
```

0.5.2 Comprensión de Set

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))

[138]:
```
#En cristiano
c = set({})
#recorremos del 0 al 9 y guardamos en "x"
for x in range(10):
    #aquí usaremos la función add() para añadir en los sets
    c.add(x)
#valor
print(c)
#tipo de dato
print(type(c))
```

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

0.5.3 Comprensión Tuplas

[139]:
```
tupla = tuple(x for x in range(4))
print(tupla)
```

```
(0, 1, 2, 3)
```

[142]:
```
#En cristiano

tupla = []

for x in range(4):
    tupla.append(x)


tupla = tuple(tupla)
print(tupla)
print(type(tupla))
```

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

0.6 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.

0.6.1 Try / except

Permite capturar excepciones y actuar en consecuencia

[145]:
```
#Ejemplo: error de acceso fuera de rango.
lista = [6, 1, 0, 5]
lista[4]
print('Código tras el error') #saldrá error de que está fuera de rango
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[145], line 3
      1 #Ejemplo: error de acceso fuera de rango.
      2 lista = [6, 1, 0, 5]
----> 3 lista[4]
      4 print('Código tras el error')

IndexError: list index out of range
```

[149]:
```
lista = [6, 1, 0, 5]
i = 300
#Creamos un try (es como un if else)
try:
    #a  = numero que está en el indice
    a = lista[i]
#En caso que no exista, se crea una excepción llamada "IndexError"
#Es porque el error está en el index
except IndexError:
    print('He capturado la excepción de tipo IndexError')
   #Devolvemos un valor para que no caiga
    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 más específicos posible, pero también se pueden usar en una sentencia **try-except** que capture cualquier error.

[147]:
```
#Se puede utilizar en una sola sentencia
try:
    #4 para 0 no debería dar nada
    4/0
#Como saldrá un error
except:
    #Imprimimos un mensaje con el error
    print('He capturado la división por cero')
```

```
He capturado la división por cero
```

- Recoger la excepción e imprimir el mensaje de error

[150]:
```
#Aquí es algo nuevo porque podemos traer el error directamente

lista = [6, 1, 0, 5]
#Creamos un try
try:
    #Imprimimos el indice 4 de la lista
    print(lista[4])
#Como no existe ese indice, podemos guardar el error en la variable e
except Exception as e:
    #Imprimimos el error transformandolo a string
    print("Error = " + str(e))

print('Reacheable Code')
```

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

0.6.2 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.

[156]:
```
#Básicamente el finally se ejecuta exista o no un error
lista = [6, 1, 44, 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
44
```

0.6.3 Raise

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

[162]:
```
#Básicamente el raise me permite crear mis propios errores
lista = [6, 1, 0, 5]
indice = 12
```

```
try:
    if indice >= len(lista):
        raise IndexError('Soy un error creado, el índice esta fuera de rango')
except IndexError as err:
    print('Excepción de tipo IndexError capturada', err)
```

```
Excepción de tipo IndexError capturada Soy un error creado, el índice esta fuera
de rango
```

1 Ejercicios de Programación en Python

1.1 1. Suma de elementos de una lista

Escribe un programa que calcule la suma de todos los elementos de una lista dada.
La lista sólo puede contener elementos numéricos.

1.2 2. Eliminar elementos duplicados

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`.

1.3 3. Diccionario (x, x²)

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}`.

1.4 4. Buscar palabra que empiece por "a"

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.
Debe mostrar un mensaje que indique el éxito o el fracaso de la búsqueda y, en caso de éxito, mostrar la palabra encontrada.

1.5 5. Índices de múltiplos de 3

Dada una lista `L` de números positivos, escribe un programa que muestre otra lista ordenada que contenga todo índice `i` tal que `L[i]` es múltiplo de 3.
Ejemplo: para `L = [3, 5, 13, 12, 1, 9]` el programa mostrará `[0, 3, 5]`.

1.6 6. Clave con el valor más alto

Dado un diccionario con claves `str` y valores `int` o `float`, escribe un programa que muestre la clave cuyo valor sea el más alto.
Ejemplo: `{'a': 4.3, 'b': 1, 'c': 7.8, 'd': -5}` → respuesta: `'c'`.

1.7 7. Producto condicionado entre dos listas

Dadas las listas `a = [2, 4, 6, 8]` y `b = [7, 11, 15, 22]`, escribe un programa que multiplique cada elemento de `a` mayor que 5 por cada elemento de `b` menor que 14 y muestre los resultados.

1.8 8. División con manejo de excepciones

Escribe un programa que pida un valor numérico `X` al usuario usando `input` y muestre `10 / X`. El programa debe manejar correctamente las excepciones mostrando mensajes informativos.

1.9 9. Acceso a diccionario con manejo de errores

Crea un diccionario cualquiera y pide al usuario una clave.
Si existe, muestra su valor. Si no existe, muestra un mensaje de error adecuado.

1.10 10. List comprehension de enteros positivos

Escribe una list comprehension que construya una lista con los números enteros positivos de una lista dada que puede contener números `float`.

1.11 11. Set comprehension de vocales

Escribe una set comprehension que, dada una palabra, construya un conjunto con sus vocales.

1.12 12. Números del 0 al 50 que contengan el dígito 3

Escribe una list comprehension que construya:
`[3, 13, 23, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 43]`.

1.13 13. Tamaño de palabras en una frase

Escribe una dictionary comprehension que, dada una frase, construya un diccionario con el tamaño de cada palabra.
Ejemplo: `"Soy un ser humano"` → `{'Soy': 3, 'un': 2, 'ser': 3, 'humano': 6}`.

1.14 14. Números del 1 al 10 mitad numérico mitad texto

Escribe una list comprehension que genere:
`[1, 2, 3, 4, 5, 'seis', 'siete', 'ocho', 'nueve', 'diez']`.

[164]:
```
"""
1) Escribe un programa que calcule la suma de todos los elementos de una lista␣
 ↪dada. La lista
sólo puede contener elementos numéricos

"""
#LÓGICA
#1) Crear una lista de números (no tan grande para comprobar xd)
lista = [1,4,5]
#2) Crear un acumulador
```

```
acumulador = 0

#3) Crear un bucle for para recorrer la lista y guardarlo en una variable
for numeros in lista:
    #Sumar el acumulador más los números para que a medida que vaya subiendo el␣
 ↪índice, se vayan sumando
    acumulador+= numeros
    #print(acumulador)
#4) fuera del for, imprimir para ver el resultado final, en caso de querer ver␣
 ↪uno por uno, se hace el print dentro del for
print(acumulador)
```

```
10
```

[169]:
```
"""
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'
"""

#LÓGICA

#1) Crear una lista con elementos duplicados
lista_duplicada = [1,1,1,1,2,2,2,2,3,3,3,4,2,1,2,3,4,5,6]
#2) Creamos una nueva lista vacía
lista_nueva = []

#3) Creamos un bucle for que recorrera toda la lista duplicada y se enviará a␣
 ↪una variable
for num_repetidos in lista_duplicada:
    #4) Añadiremos un if en el cual pondremos la condición "si no está en␣
 ↪lista_nueva" entonces:1
    if num_repetidos not in lista_nueva:
        #Añadimos el número a la lista vacía
        lista_nueva.append(num_repetidos)
#Imprimimos
print(lista_nueva)
```

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

[173]:
```
"""
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}

"""


#LÓGICA: básicamente tenemos que hacer que los valores sean el cuadrado de las␣
 ↪claves

#1) Crear la sentencia en una sola linea
#diccionario = {d:x*x for x in range(12)}
diccionario = {x: x*x for x in range(12)}
print(diccionario)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11:
121}
```

[172]:
```
#En cristiano sería
diccionario = {}

for x in range(12):
    diccionario[x] = x*x

print(diccionario)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11:
121}
```

[193]:
```
"""
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.
"""
#LÓGICA

#1) Crear una lista de palabras
lista_palabras = ["Goku", "Vegeta", "Jonnath", "me gusta", "la carne", "la␣
 ↪leche", "y el pan"]

#2) Palabra por buscar
palabras_buscar = "y el pan"
```

```
#3) Creamos un estado que por defecto estará en falso
encontrada = False


#4) Creamos un for para recorrer las palabras que hay en la lista y almacenar␣
 ↪en "palabras"
for palabras in lista_palabras:
    #Si la palabra para buscar está en "palabras" entonces
    if palabras_buscar in palabras:
        #Cambio el estado a verdadero
        encontrada = True
        #Imprimimos el texto acierto
        print("Encontramos la palabra? ", encontrada)
        print("palabra: ", palabras_buscar)
#En caso que no se encuentre la palabra, sale el texto negativo
if palabras_buscar not in palabras:
    print("Encontramos la palabra? ", encontrada)
```

```
Encontramos la palabra?  True
palabra:  y el pan
```

[205]:
```
"""
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.

"""

#LÓGICA
#1) Crear una lista llamada "L"
L = [5,4,3,1,2,6,8,7,10,9]
#2) Creamos una lista vacía
L_ordenada = []
#3) Creamos el índice
indice = 0

#4) Crear un bucle que permita recorrer la lista y guardarlo en una variable
for nueva in L:
    #5) Crear un if donde diga que todos los números multiplos de 3 en el␣
 ↪índice se muestren
    if indice % 3 == 0:
```

```
        #6) Agregamos una variable que contenga al número que está en el indice
        nueva_var= L[indice]
        #7) añadimos a la nueva lista (por el momento estará desordenada)
        L_ordenada.append(nueva_var)
    #8) hacemos que el indice vaya aumentando y recorriendo todo
    indice += 1
#9) ordenamos la lista
L_ordenada.sort()
#10) Print
print(L_ordenada)

#en caso que solo sea el indice es así:

"""
if indice % 3 == 0:
    L_ordenada.append(indice)
"""
print()
```

```
[1, 5, 8, 9]
```

[206]:
```
"""
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.
"""

#LÓGICA

#1) Crear un diccionario con claves string y valores numéricos
diccionario = {'a': 4.3, 'b': 1, 'c': 7.8, 'd': -5}

#2) Crear una variable para guardar el valor más alto (empieza muy bajo)
mayor_valor = float('-inf')

#3) Crear una variable para guardar la clave asociada al mayor valor
mayor_clave = ""

#4) Crear un bucle for para recorrer el diccionario usando items()
for clave, valor in diccionario.items():
```

```
    #5) Comparar si el valor actual es mayor al valor más alto guardado
    if valor > mayor_valor:

        #6) Si es mayor, actualizar el valor más alto
        mayor_valor = valor

        #7) Guardar la clave correspondiente a ese nuevo valor más alto
        mayor_clave = clave

#8) Mostrar por pantalla la clave cuyo valor es el mayor del diccionario
print("La clave con el valor más alto es:", mayor_clave)
```

```
La clave con el valor más alto es: c
```

[207]:
```
"""

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.

"""

#LÓGICA

#1) Crear la lista a
a = [2, 4, 6, 8]

#2) Crear la lista b
b = [7, 11, 15, 22]

#3) Crear dos bucles for para recorrer ambas listas
for x in a:
    #4) Verificar si el valor de a es mayor que 5
    if x > 5:
        for y in b:
            #5) Verificar si el valor de b es menor que 14
            if y < 14:
                #6) Mostrar el resultado de la multiplicación
                print(x * y)
```

```
42
66
56
88
```

[208]:
```
"""
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
"""

#LÓGICA

#1) Pedir un valor al usuario
try:
    x = float(input("Ingrese un número: "))

    #2) Realizar la división
    resultado = 10 / x

    #3) Mostrar el resultado
    print("Resultado:", resultado)

#4) Capturar error si no se ingresa un número
except ValueError:
    print("Debe ingresar un valor numérico válido")

#5) Capturar error si se intenta dividir por cero
except ZeroDivisionError:
    print("No se puede dividir para cero")
```

```
Ingrese un número:  12
Resultado: 0.8333333333333334
```

[210]:
```
"""
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.

"""

#LÓGICA

#1) Crear un diccionario cualquiera
diccionario = {"nombre": "Juan", "edad": 25, "ciudad": "Quito"}

#2) Pedir al usuario una clave
clave = input("Ingrese una clave: ")

#3) Usar try / except para buscar la clave
try:
    #4) Mostrar el valor asociado
    print("Valor:", diccionario[clave])
except KeyError:
    #5) Mostrar mensaje si la clave no existe
    print("La clave ingresada no existe en el diccionario")
```

```
Ingrese una clave:  nombre
Valor: Juan
```

[ ]:
```
"""
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.
"""
#LÓGICA

#1) Crear una lista con enteros y floats
lista = [1, 2.5, 3, -1, 4, 6.7, 5]

#2) Construir una nueva lista solo con enteros positivos
nueva_lista = [x for x in lista if isinstance(x, int) and x > 0]

#3) Mostrar la lista resultante
print(nueva_lista)
```

[ ]:
```

```

[213]:
```
"""
Escribe una set comprehension que, dada una palabra, construya un conjunto que␣
 ↪contenga
las vocales de dicha palabra.
```

```
"""
#LÓGICA

#1) Pedir una palabra al usuario
palabra = input("Ingrese una palabra: ")

#2) Crear un conjunto con las vocales
vocales = {letra for letra in palabra.lower() if letra in "aeiou"}

#3) Mostrar el conjunto
print(vocales)
```

```
Ingrese una palabra:  a
```

```
{'a'}
```

[215]:
```
"""
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].
"""

#LÓGICA

#1) Crear la list comprehension
lista = [x for x in range(51) if '3' in str(x)]

#2) Mostrar la lista
print(lista)
```

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

[216]:
```
"""
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}
"""

#LÓGICA

#1) Definir la frase
frase = "Soy un ser humano"

#2) Crear el diccionario con dictionary comprehension
diccionario = {palabra: len(palabra) for palabra in frase.split()}
```

```
#3) Mostrar el diccionario
print(diccionario)
```

```
{'Soy': 3, 'un': 2, 'ser': 3, 'humano': 6}
```

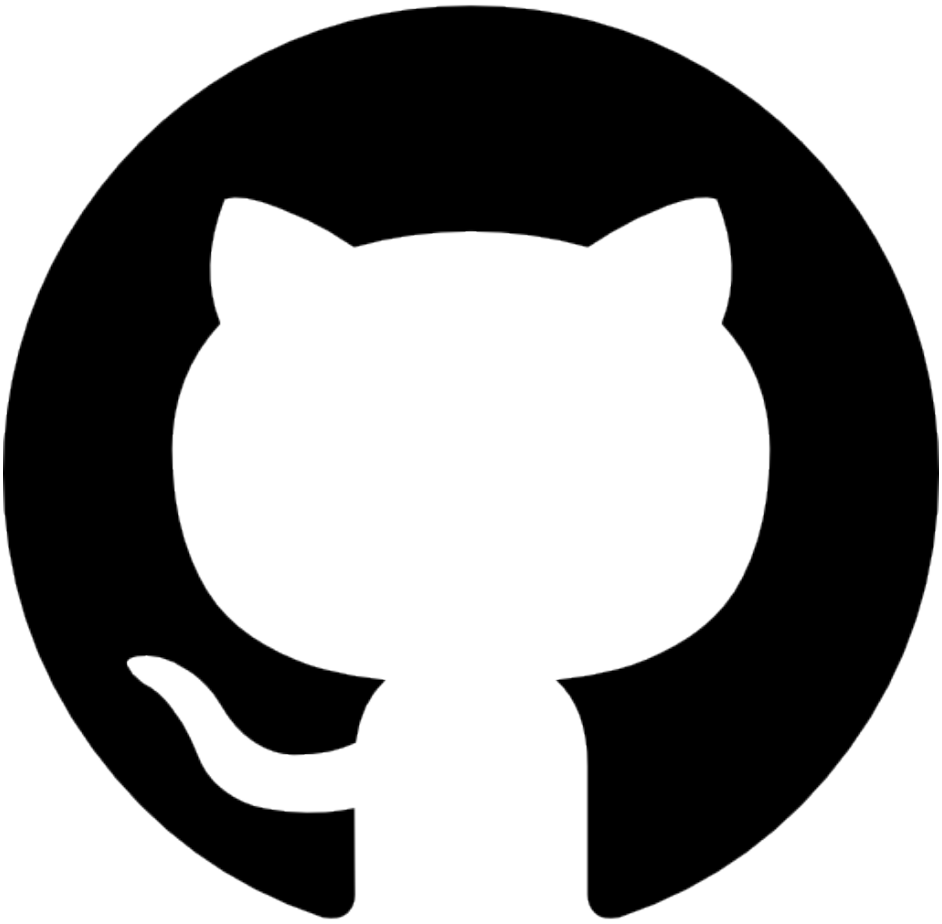
[217]:
```
"""
Escribe una list comprehension que construya una lista que incluya todos los␣
 ↪números del 1
al 10 en orden. La primera mitad se mostrarán en formato numérico; la segunda␣
 ↪mitad en
texto. Es decir, el resultado será: [1, 2, 3, 4, 5, 'seis', 'siete', 'ocho',␣
 ↪'nueve', 'diez'].
"""

#LÓGICA

#1) Crear lista con nombres en texto
texto = ["seis", "siete", "ocho", "nueve", "diez"]

#2) Construir la lista combinada
lista = [i for i in range(1, 6)] + texto

#3) Mostrar la lista
print(lista)
```

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

1.15 Bibliografía

1. Christian. (2021, junio 4). Doble loop en una list comprehension? (lista de diccionarios). Stack Overflow en español. Recuperado de https://es.stackoverflow.com/questions/459575/doble-loop-en-una-list-comprehension-lista-de-diccionarios
2. Lleims. (2018, octubre 7). list comprehensions. Recuperado de https://es.stackoverflow.com/questions/202612/list-comprehensions
3. Dictionary comprehension (consulta de publicaciones etiquetadas). (2025). Stack Overflow. Recuperado de https://stackoverflow.com/questions/tagged/dictionary-comprehension
4. Fellman, N. (s. f.). How to handle exceptions in a list comprehensions. Recuperado de https://qastack.mx/programming/1528237/how-to-handle-exceptions-in-a-list-comprehensions

2 Github



2.0.1 Click aquí para ver el repositorio

[ ]: