clase2-estructuras-basicas

April 27, 2024

 # Clase 2 - 27/04/2024: Estructuras básicas

1 Parte 1: Más sobre Condicionales

Condicionales:

- 1.1) Condicionales más utilizados: `if/else/elif`
- 1.2) Otros condicionales: `match/case`

1.1) Condicionales más utilizados: `if/else/elif`

1.1.1) Condicional Simple

[61]:
```
num_1 = 1
num_2  = 20
if num_1 < num_2:
  print(f'El numero {num_1} es menor al numero {num_2}')
else:
  print(f'El numero {num_2} es mayor o igual a mi numero {num_1}')
```

```
El numero 1 es menor al numero 20
```

1.1.2) Condiciones anidadas

Con Mismo tipo de dato

[62]:
```
num_1 = 1   # Ejecutar con num_1 = 20 y ver cual es el resultado
num_2  = 20
if num_1 < num_2 or num_1 == num_2 :
  print(f'El numero {num_1} es menor o igual al numero {num_2}')
else:
  print(f'El numero {num_2} es mayor a mi numero {num_1}')
```

```
El numero 1 es menor o igual al numero 20
```

Mezclando tipos de datos.

[63]:
```
# Datos del estudiante
edad = 17
tiene_membresia_anterior = False
```

```python
# Verificar si puede ingresar al club
if edad >= 18 or tiene_membresia_anterior:
    print("¡Bienvenido al club!")
else:
    print("Lo siento, no cumples con los requisitos para ingresar al club.")
```

```
Lo siento, no cumples con los requisitos para ingresar al club.
```

1.1.3) Vamos complejizando: Sumamos `elif`.

Esta sintaxis tiene la particularidad de que ante una serie de condiciones, al encontrar la primer ocurrencia **verdadera** no sigue evaluando las demás restantes.

[64]:
```python
# Puntuación del estudiante
puntuacion = 85

# Determinar la calificación del estudiante
if puntuacion >= 90:
    calificacion = "A"
elif puntuacion >= 80:
    calificacion = "B"
elif puntuacion >= 70:
    calificacion = "C"
elif puntuacion >= 60:
    calificacion = "D"
else:
    calificacion = "F"
# Imprimir la calificación del estudiante
print(f"La calificación del estudiante es: {calificacion}")
```

```
La calificación del estudiante es: B
```

Utilizamos la función `input()` para recibir un dato del usuario que vimos en la clase anterior. Recordar que dicho valor ingresado siempre viene como `string` y debe ser casteado (convertido) a `int()`

[65]:
```python
# Datos de equipos
edad = int(input("Ingresa tu edad: "))

# En base a la edad ingresada por el usuario, determinamos en qué categoría se␣
 ↪encuentra
if edad < 8:
  print('Categoría correspondiente: Infantiles')
elif edad >= 8 and edad < 12:
  print('Categoría correspondiente: Juveniles I')
elif  edad >= 12 and edad < 14:
  print('Categoría correspondiente: Juveniles II')
elif edad >= 14 and edad < 20:
  print('Categoría correspondiente: Mayores')
```

```
else:
  print('Categoría correspondiente: Adultos')
```

```
Ingresa tu edad: 32
Categoría correspondiente: Adultos
```

Vamos complejizando: Agregamos condicionales en base a distintos tipos de datos

[66]:
```
# Datos de equipos
club = input("Ingresa tu Club: ")
edad = int(input("Ingresa tu edad: "))

# En base a la edad ingresada por el usuario, determinamos en qué categoría se␣
 ↪encuentra

# OJO!: Según si es boca o River, nombrarán distinto a la categoría
if edad < 8 and club == 'Boca Juniors':
  print('Categoría correspondiente: Menores')
elif edad < 8 and club == 'River Plate':
  print('Categoría correspondiente: Infantiles')
elif edad > 8:
  print('Categorías avanzadas')
else:
  print('El club ingresado no está registrado en nuestra base')
```

```
Ingresa tu Club: Boca Juniors
Ingresa tu edad: 32
Categorías avanzadas
```

1.2) Otros condicionales: `match/case`

Este algoritmo nos sirve cuando los condicionales a evaluar no son demasiado complejos. Otorga una sintaxis más legible y sencilla.

[67]:
```
x = 10
match x:
    case 1:
        print('x es 1')
    case 5:
        print('x es 5')
    case _:
        print('x no esta dentro de mis valores esperados <unknown>')
```

```
x no esta dentro de mis valores esperados <unknown>
```

Vamos con un ejemplo más complejo.

Nota 1: El caracter `|` es la representación del condicional `or`

Nota 2: El caracter `_` es una convención en python para representar "cualquier otro caso".

[68]:
```python
edad = int(input('Ingrese su edad: '))
match edad:
        case 0 | 1 | 2:
            print("Bebé")
        case 3 | 4 | 5:
            print("Niño pequeño")
        case 6 | 7 | 8 | 9 | 10:
            print("Niño")
        case 11 | 12 | 13 | 14 | 15:
            print("Adolescente")
        case 16 | 17 | 18 | 19 | 20:
            print("Joven adulto")
        case _:
            print("Adulto")
```

```
Ingrese su edad: 32
Adulto
```

Ejercicios Parte 1

Parte 1 - Ejercicio de clase

Escribe un programa en Python que solicite al usuario que ingrese un número. - Determinar si el número ingresado es positivo, negativo o cero, e imprimir el resultado correspondiente en pantalla. - Si es positivo multuplicar el número por 2 - Si es negativo multuplicar el número por 3 - Imprimir los resultados en pantalla

[69]:
```python
# Pedir al usuario que ingrese un número
numero = float(input("Por favor, ingresa un número: "))

# Verificar si el número es positivo, negativo o cero
if numero > 0:
    resultado_final = numero*2
    print(f"El número final es {resultado_final}.")
elif numero < 0:
    resultado_final = numero*3
    print(f"El número final es {resultado_final}.")
else:
    print("El número ingresado es cero.")
```

```
Por favor, ingresa un número: 2
El número final es 4.0.
```

Parte 1 - Ejercicio de tarea 1

Escribe un programa en Python que solicite al usuario que ingrese un número. - Luego, el programa debe determinar si el número ingresado es par o impar, y realizar una transformación sobre el número según el caso. - Si el número es par, dividirlo por 2. - Si el número es impar, elevarlo al cuadrado. Finalmente, el programa debe imprimir el resultado transformado en pantalla.

[70]:
```
# Pedir al usuario que ingrese un número
numero = int(input("Por favor, ingresa un número: "))

# Verificar si el número ingresado es par o impar. Recordemos que % se usa para␣
 ↪obtener el 'resto' de una división
if numero % 2 == 0:
    # Si el número es par, dividirlo por 2 utiliando '//' para división entera
    resultado_transformado = numero // 2
else:
    # Si el número es impar, elevarlo al cuadrado
    resultado_transformado = numero ** 2

print(f"El número transformado es {resultado_transformado}.")
```

```
Por favor, ingresa un número: 2
El número transformado es 1.
```

Parte 1 - Ejercicio de tarea 2

Supongamos que tienes cuatro productos en una tienda en línea: zapatillas, antiparras, mancuernas y tobilleras. Cada producto tiene un estado activo que indica si está disponible para la venta o no, y un precio asociado. Se debe definir una variable por cada producto (4), sus estados (4 variables más) y su precio (4 variables más).

Escribir un programa en Python que determine el primer producto en oferta que encuentre. Un producto se considera en oferta si está disponible para la venta y su precio es menor o igual a $20.000. El programa debe imprimir en pantalla el nombre de los productos en oferta.

Código inicial

[71]:
```
# Datos de los productos
producto_1 = 'Zapatillas'
producto_2 = 'Antiparras'
producto_3 = 'Mancuerna'
producto_4 = 'Tobilleras'
producto_1_active = True
producto_2_active = True
producto_3_active = True
producto_4_active = False
producto_1_precio = 200000
producto_2_precio = 20500
producto_3_precio = 12100
producto_4_precio = 5000

# Determinar productos en oferta, siempre y cuand esté activo y su precio sea␣
 ↪<= a 200000
# Completar el programa...
```

Código resolución

[72]:
```python
# Datos de los productos
producto_1 = 'Zapatillas'
producto_2 = 'Antiparras'
producto_3 = 'Mancuerna'
producto_4 = 'Tobilleras'
producto_1_active = True
producto_2_active = True
producto_3_active = True
producto_4_active = False
producto_1_precio = 200000
producto_2_precio = 20500
producto_3_precio = 12100
producto_4_precio = 5000

# Encontrar el primer producto en oferta e informar.
print("Productos en oferta:")
if producto_1_active and producto_1_precio <= 20000:
    print(producto_1)
elif producto_2_active and producto_2_precio <= 20000:
    print(producto_2)
elif producto_3_active and producto_3_precio <= 20000:
    print(producto_3)
elif producto_4_active and producto_4_precio <= 20000:
    print(producto_4)
```

```
Productos en oferta:
Mancuerna
```

2 Parte 2: Estructuras I

- 2.1) Listas
- 2.2) Set (Conjuntos)
- 2.3) Matrices

2.1) Listas

- Permiten almacenar objetos mediante un **orden definido**, es decir son una colección ordenada.
- Permiten duplicados
- Versátiles: permiten distintos tipos de datos
- Mutables
- Uso: Cuando se requiere orden e indexación

2.1.1) Lista vacía, homogenea y hetergenea

[73]:
```python
# Vacia
lista_vacia =[]
# Homogenea: Mismo tipo de dats
```

```
lista1 = [12, 34, 55, 10, 23]
#Hetegorenea: Mezclado tipos de datos
lista2 = [12, 3.14, 'Juan', True, 22]
```

2.1.2) Opraciones con Listas

2.1.2.1) Obtener un elemento: `index()`

[74]:
```
compras = ['Frutas', 'Carne', 'Limpieza']
print('El 1er elemento de mi lista es',compras[0])
print('El 2do elemento de mi lista es',compras[1])
print('El 3er elemento de mi lista es',compras[2])

# Adultosprint('El 4er elemento de mi lista es',compras[3])  #! Nota: Error "list index out of range" (Fuera de rango)
```

```
El 1er elemento de mi lista es Frutas
El 2do elemento de mi lista es Carne
El 3er elemento de mi lista es Limpieza
```

2.1.2.2) Trocear una lista

[75]:
```
trozo1 = compras[1:]
print(trozo1)
```

```
['Carne', 'Limpieza']
```

2.1.2.3) Obtener rango de elementos indicando los saltos `step`

[76]:
```
# start-stop-step
trozo2 = compras[1::2]
print(trozo2)
```

```
['Carne']
```

[77]:
```
trozo3 = compras[4::-1]
print(trozo3)
```

```
['Limpieza', 'Carne', 'Frutas']
```

2.1.2.4) Invertir una lista: `reversed()`

[78]:
```
lista = ['Boca', 'River', 'Racing', 'Independiente', 'San Lorenzo']
lista_nueva_1 = list(reversed(lista))
print(lista_nueva_1)
```

```
['San Lorenzo', 'Independiente', 'Racing', 'River', 'Boca']
```

2.1.2.5) Añadir elementos a una lista: `append()`

[79]:
```
lista = ['Juan', 12, 'Martin', 15, 'Pablo']
lista.append(20)
print(lista)
```

```
['Juan', 12, 'Martin', 15, 'Pablo', 20]
```

2.1.2.6) Combinar listas

[80]:
```
lista1 = ['Huevos', 'Calabaza', 'Verdura']
lista2 = ['Carne', 'Pollo', 'Legumbres']

# Conservando lista original: "+" o "+="
compras = lista1 + lista2
print(compras)
```

```
['Huevos', 'Calabaza', 'Verdura', 'Carne', 'Pollo', 'Legumbres']
```

2.1.2.7) Borrar elementos: `remove()`

[81]:
```
# Por el elemento: remove()
lista = ['Huevos', 'Calabaza', 'Verdura']
lista.remove('Verdura')     # Si hay Duplicados: Solo borra la primer␣
 ↪existencia.
print(lista)
```

```
['Huevos', 'Calabaza']
```

2.1.2.8) Pertenencia de elementos: `is in`.

[82]:
```
lista = ['Huevos', 'Calabaza', 'Verdura', 'Jamon', 'Arroz']
pertenencia1 = 'Arroz' in lista
pertenencia2 = 'Sal' in lista
print(pertenencia1, pertenencia2)
```

```
True False
```

2.1.2.9) Ordenar una lista: `sort()`

[83]:
```
lista = ['Huevos', 'Calabaza', 'Verdura', 'Jamon', 'Arroz', 'Verdura']
ordenada = sorted(lista)
print(ordenada)
# ordenada = sorted(lista, revserse = True)  En sentido inverso
```

```
['Arroz', 'Calabaza', 'Huevos', 'Jamon', 'Verdura', 'Verdura']
```

2.1.2.10) Longitud de una lista: `len()`

[84]:
```
lista = ['Huevos', 'Calabaza', 'Verdura', 'Jamon', 'Arroz', 'Verdura']
long = len(lista)
print(long)
```

```
6
```

2.1.2.11) Iterar `indice` y `elemento`: `enumerate()`

[85]:
```
lista = ['Huevos', 'Calabaza', 'Verdura', 'Jamon', 'Arroz', 'Verdura']
for indice, elemento in enumerate(lista):
    print(indice, elemento)
```

```
0 Huevos
1 Calabaza
2 Verdura
3 Jamon
4 Arroz
5 Verdura
```

Nota: Profundizaremos en bucles al final de esta clase.

2.1.2.12) Copiar una lista: `copy()`

[86]:
```
listaA = ['Real Madrid', 'Barcelona','Atletico de Madrid']
listaB = listaA.copy()   # Se crea una nueva lista
print(listaB)
```

```
['Real Madrid', 'Barcelona', 'Atletico de Madrid']
```

Más sobre `listas`: https://docs.python.org/es/3/tutorial/introduction.html#lists

2.2) Set (Conjuntos)

Serie de valores únicos **sin orden establecido**.

- Soportan diferentes tipos de datos
- Soportan operaciones de Union, Intersección y diferencia.
- Se eliminan automáticamente los duplicados
- Uso: Cuando se necesita asegurar valores únicos y realizar operaciones de conjuntos.
- Mutables

2.2.1) Definir conjunto. Conjunto vacío

[87]:
```
set_vacio = {}
set1 = {2,55,12,22,7}
print(set_vacio)
print(set1)
```

```
{}
{2, 55, 22, 7, 12}
```

2.2.2) Eliminación de duplicados

[88]:
```
set1 = {1,1,2}
print(set1)
```

```
{1, 2}
```

2.2.3) Obtener elemento

No se puede obtener un elemento así como hacemos en las `listas` ya que son una colección desordenada de elementos únicos

2.2.4) Añadir un elemento: `add()`

[89]:
```
# Notar que se añaden de forma ordenada
set = {1, 3, 4, 10}
set.add(5)
set.add(1.2)
set.add(0.5)
print(set)
```

```
{0.5, 1, 1.2, 3, 4, 5, 10}
```

[90]:
```
# Notar que se añaden de forma ordenada
set = {'Juan', 'Pedro', 'Manuel'}
set.add(5)
set.add(True)
print(set)
```

```
{'Juan', True, 5, 'Manuel', 'Pedro'}
```

2.2.5) Remover un elemento: `remove()`

[91]:
```
set = {1, 3, 4, 10}
set.remove(10)
print(set)
```

```
{1, 3, 4}
```

2.2.6) Longitud de un conjunto: `len()`

[92]:
```
set1 = {1, 3, 4, 10}
set2= {True, 'Juan', 3.14, 'Maria'}
print(len(set1))
print(len(set2))
```

```
4
4
```

2.2.7) Pertenencia de un elemento: `in` o `not in`

[93]:
```
set = {1, 3, 4, 10}
print(1 in set)
print(1 not in set)
```

```
True
False
```

2.2.8) Intersección de conjuntos: `&` Y `intersection()`

[94]:
```
set1 = {1, 2}
set2 = {2, 3}
# Metodo 1
intersection1 = set1 & set2
# Metodo 1
intersection2 = set1.intersection(set2)
print(intersection1)
print(intersection2)
```

```
{2}
{2}
```

2.2.9) Unión de conjuntos: `|` Y `union()`

[95]:
```
set1 = {1, 2}
set2 = {2, 3}
# Metodo 1
union1 = set1|set2
# Metodo 2
union2 = set1.union(set2)
print(union1)
print(union2)
```

```
{1, 2, 3}
{1, 2, 3}
```

2.2.9) Diferencia simétrica

[96]:
```
set1 = {1, 2}
set2 = {2, 3}
dif1 = set1 ^ set2
dif2 = set1.symmetric_difference(set2)
print(dif1)
print(dif2)
```

```
{1, 3}
{1, 3}
```

2.2.10) Diferencia no simétrica

[97]:
```
set1 = {1, 2}
set2 = {2, 3}
dif1 = set1 - set2
dif2 = set1.difference(set2)
print(dif1)
print(dif2)
```

```
{1}
{1}
```

Más sobre `conjuntos`: https://docs.python.org/es/3/tutorial/datastructures.html#sets

2.3) Matrices

Una matriz es una estructura de **datos bidimensional** que contiene elementos dispuestos en filas y columnas. Las matrices generalmente se representan utilizando **listas anidadas**. Una matriz es simplemente una colección bidimensional de elementos, donde cada elemento está organizado en filas y columnas.

2.3.1) Definir una matriz

[98]:
```
# Ejemplo matriz 2 filas x 3 columnas
matriz = [
    [1, 2, 3],
    [4, 5, 6]
]
print(matriz)
```

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

2.3.2) Obtener un elemento de la matriz

[99]:
```
matriz = [
    [1, 2, 3],
    [4, 5, 6]
]
print("La 1er fila de mi matriz es ",matriz[0])
print("El 3er elemento de la 1er fila es ",matriz[0][2])
```

```
La 1er fila de mi matriz es  [1, 2, 3]
El 3er elemento de la 1er fila es  3
```

[100]:
```
# Uniendo sus elementos mediante algún 'separador'
lista = ['Huevos', 'Calabaza', 'Verdura', 'Jamon', 'Arroz', 'Verdura']
string1 = ','.join(lista)
string2 = '|'.join(lista)
print(string1)
print(string2)
# Esta función es la opuesta a 'split()' en strings
```

```
Huevos,Calabaza,Verdura,Jamon,Arroz,Verdura
Huevos|Calabaza|Verdura|Jamon|Arroz|Verdura
```

Ejercicios Parte 2

Parte 2 - Ejercicio de clase

Definir una lista de productos de un local de comida para obtener el maestro de produtos - Hamburguesa Moon - Mega panchos - Nuggets extra queso - Empanadas

Se contrata un nuevo proveedor el cual adiciona 2 nuevos productos a nuestro maestro que antes no estaban: - Super pancho doble - Pizza individual

De la misma forma, el proveedor de el artículo `empanadas` deja de producir en el pais.

Definir una lista que contenga el maestro de productos inicial y realizarle las actualizaciones correspondientes imprimiendo el maestro inicial y el final.

[101]:
```
lista = ['Hamburguesa Moon', 'Mega panchos', ' Nuggest extra queso',␣
 ↪'Empanadas']
lista.append('Super pancho doble')
lista.append('Pizza individual')
lista.remove('Empanadas')
print(lista)
```

```
['Hamburguesa Moon', 'Mega panchos', ' Nuggest extra queso', 'Super pancho
doble', 'Pizza individual']
```

Parte 2 - Ejercicio de tarea 1

Dada la siguientes `lista` en Python correspondiente a la nómina inicial de la selección juvenil de futbol mixto del Club MateAmargo: `['Juan', 'Micaela', 'Emilia', 'Cecilia', 'Juan Manuel', 'Valentina', 'Luciano']` - Recientemente se sumaron al equipo: "Augusto" y "David". Agregar dichos elementos a la lista inicial. - Además Juan tuvo que salir de la lista por estar fuera por lesión durante al menos 6 meses. - Sumar a la lista inicial concatenando la lista de suplentes: `['Julieta', 'Jose', 'Franco']` - Realizar una copia de 'back up' de la nomina y guardarlo en una nueva variable `lista_backup`

[102]:
```
lista = ['Juan', 'Micaela', 'Emilia', 'Cecilia', 'Juan Manuel', 'Valentina',␣
 ↪'Luciano']

# Agrego a los que se sumaron
lista.append('Augusto')
lista.append('David')

# Quitamos al lesionado
lista.remove("Juan")

# Back up de la nómina
lista_backup = lista.copy()
print(f'Mi nómina es: {lista}')
print(f'Mi back up es: {lista_backup}')
```

```
Mi nómina es: ['Micaela', 'Emilia', 'Cecilia', 'Juan Manuel', 'Valentina',
'Luciano', 'Augusto', 'David']
Mi back up es: ['Micaela', 'Emilia', 'Cecilia', 'Juan Manuel', 'Valentina',
'Luciano', 'Augusto', 'David']
```

Parte 2 - Ejercicio de tarea 2

Dada los siguientes conjuntos: - `c1 = {1, 2, 'Juan', 'Maria', True, 5.55, 24.3}` - `c2 = {1, 10, 'Diego', 'Karina', False, True, 33.3}`

Obtener la `interseccion` y la `union` de los conjuntos e imprimir en pantalla el resultado.

[105]:
```
# Defin las listas
c1 = {1, 2, 'Juan', 'Maria', True, 5.55, 1.5}
c2 = {1, 10, 'Diego', 'Karina', False, True, 33.3}

# Interseccion
interseccion = c1.intersection(c2)
# Union
union = c1.union(c2)

# Imprimo en pantalla
print(f'Mi lista1 pasada a conjunto es: {c1}')
print(f'Mi lista2 pasada a conjunto es: {c2}')
print(f'La interseccion de mis conjuntos es: {interseccion}')
print(f'La union de mis conjuntos es: {union}')
```

```
Mi lista1 pasada a conjunto es: {'Juan', 1, 2, 1.5, 5.55, 'Maria'}
Mi lista2 pasada a conjunto es: {False, 1, 33.3, 'Karina', 10, 'Diego'}
La interseccion de mis conjuntos es: {1}
La union de mis conjuntos es: {'Juan', 1, 2, 1.5, False, 5.55, 33.3, 'Karina',
10, 'Maria', 'Diego'}
```

3 Parte 3: Bucles

- 3.1) Bucles `for`
- 3.2) Bucles `while`

3.1) Bucles `for`

Recordatorio: Sólo podemos recorrer objetos `iterables`: - Strings - Listas - Tuplas - Set (Conjuntos) - Diccionarios

Nota: Listas, Tuplas, Set y Diccionarios serán profundizados más adelante.

3.1.1) Bucle `for` simple

Nos ayudamos con la función `range()` que devuelve un rango numérico iterable.

[106]:
```
rango_numerico = range(0,10)
for numero in rango_numerico:
  print(numero)
```

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

```
8
9
```

3.1.2) Uso del `break`

[107]:
```python
rango_numerico = range(0,50)
for numero in rango_numerico:
  if numero <= 10:
    print(numero)
  else:
    # Si es mayor, rompo bucle
    break
```

```
0
1
2
3
4
5
6
7
8
9
10
```

3.1.3) Uso del `continue` para condiciones anidadas

Sin `continue`

[108]:
```python
for i in range(5):
    if i % 2 == 0:
        print(f"{i} es par")
    print("Esta línea se ejecuta siempre")  # Sin el continue: La ejecución␣
 ↪llega hasta esta línea.
```

```
0 es par
Esta línea se ejecuta siempre
Esta línea se ejecuta siempre
2 es par
Esta línea se ejecuta siempre
Esta línea se ejecuta siempre
4 es par
Esta línea se ejecuta siempre
```

Con `continue`

[109]:
```python
for i in range(5):
    if i % 2 == 0:
        print(f"{i} es par")
        continue
```

```
    print("Esta línea se ejecuta solo si i no es par") # Con el continue: La␣
↪ejecución NO llega hasta esta línea (continúa la siguiente vuelta).
```

```
0 es par
Esta línea se ejecuta solo si i no es par
2 es par
Esta línea se ejecuta solo si i no es par
4 es par
```

3.1.4) Bucle `for` en una estructura: Iterando una Lista

[110]:
```
# Lista de productos del supermercado
productos = ["Leche", "Huevos", "Pan", "Manzanas", "Arroz", "Pasta"]

# Iterar sobre la lista de productos e imprimir cada uno en pantalla
print("Lista de compras:")
for producto in productos:
    print(producto)
```

```
Lista de compras:
Leche
Huevos
Pan
Manzanas
Arroz
Pasta
```

3.1.3) Bucle `for` con `condicionales`. Anexo: Diferencia `if` vselif

[111]:
```
lista_nombres = ["Pedro", "Juan", "Diego", "Luis", "Miguel", "Carlos", "Matias"]

for nombre in lista_nombres:
  if nombre == "Pedro" or nombre == "Juan":
    print(f"Evaluacion 1 - {nombre} se encuentra en el Equipo A")
    continue
  elif nombre == "Diego" or nombre == "Luis":
    print(f"Evaluacion 2 - {nombre} se encuentra en el Equipo B")
    continue
  elif nombre == "Miguel" or nombre == "Carlos":
    print(f"Evaluacion 3 - {nombre} se encuentra en el Equipo C")
    continue
  # print('No encontrado')   # Comentar el `continue` de cada condición y␣
↪descomentar acá para analizar el comportamiento.
```

```
Evaluacion 1 - Pedro se encuentra en el Equipo A
Evaluacion 1 - Juan se encuentra en el Equipo A
Evaluacion 2 - Diego se encuentra en el Equipo B
Evaluacion 2 - Luis se encuentra en el Equipo B
Evaluacion 3 - Miguel se encuentra en el Equipo C
```

`Evaluacion 3 - Carlos se encuentra en el Equipo C`

Diferencia `if` vs `elif`

Ejemplo con `if`

[112]:
```
for i in range(5):
    if i == 0:
        print(f'{i} es igual a 0')
    elif i == 0 and i<1:
        print(f'{i} es igual a 0 y menor a 1')
```

```
0 es igual a 0
```

Ejemplo con `elif`

[113]:
```
for i in range(5):
    if i == 0:
        print(f'{i} es igual a 0')
    if i == 0 and i<1:
        print(f'{i} es igual a 0 y menor a 1')
```

```
0 es igual a 0
0 es igual a 0 y menor a 1
```

3.1.4) Vamos con algo más avanzado: Incluímos la función `index()`

[114]:
```
lista_nombres = ["Pedro", "Juan", "Diego", "Luis", "Miguel", "Carlos"]
for nombre in lista_nombres:
  ubicacion_del_elemento  = lista_nombres.index(nombre)
  print(f'{nombre} se encuentra en la ubicación {ubicacion_del_elemento} de la␣
 ↪lista')
```

```
Pedro se encuentra en la ubicación 0 de la lista
Juan se encuentra en la ubicación 1 de la lista
Diego se encuentra en la ubicación 2 de la lista
Luis se encuentra en la ubicación 3 de la lista
Miguel se encuentra en la ubicación 4 de la lista
Carlos se encuentra en la ubicación 5 de la lista
```

3.2) Bucles `while`

Recordatorio: Sólo podemos recorrer objetos iterables. `while` se utiliza cuando no conozco el rango exacto a recorrer.

- Suelen ir acompañados de un `contador` y un `flag`

3.2.1) Bucle `while` simple

[115]:
```
# Inicializar el contador
numero = 1
# Imprimir los números del 1 al 5
```

```
while numero <= 5:
    print(numero)
    numero += 1
```

```
1
2
3
4
5
```

3.2.2) Bucle `while`: sumamos la función `input()`

[122]:
```
# Solicitar al usuario ingresar un número entero positivo
numero_ingresado = int(input("Por favor, ingresa un número entero positivo␣
 ↪menor que 10: "))

# Inicializar el contador y el número inicial
contador = 0
numero = 10
flag = 10

# Imprimir los números pares hasta el número ingresado por el usuario
print(f"Números positivos posteriores hasta {numero_ingresado} :")
while numero_ingresado <= numero:
  if flag <= 10:
    print(numero_ingresado)
    numero_ingresado += 1
  else:
    break
```

```
Por favor, ingresa un número entero positivo menor que 10: 5
Números positivos posteriores hasta 5 :
5
6
7
8
9
10
```

Ejercicios Parte 3

Parte 3 - Ejercicio de clase

Escribir un programa que declare una variable tipo lista que contenta elementos numéricos.

Recorrer la lista definida, evaluando si el elemento iterado es “mayor o igual a cero” o si es “menor a cero” e imprimirlo en pantalla.

[123]:
```
lista = [10, 20, 3.5, -2, 45, -120]
```

```
for n in lista:
  if n >= 0:
    print(f'El numero {n} es positivo o cero')
  else:
    print(f'El numero {n} es negativo')
```

```
El numero 10 es positivo o cero
El numero 20 es positivo o cero
El numero 3.5 es positivo o cero
El numero -2 es negativo
El numero 45 es positivo o cero
El numero -120 es negativo
```

Parte 3 - Ejercicio de tarea 1

Definir una lista con al menos 5 nombres entre los cuales esté "Pedro" y luego iterar la lista hasta encontrar a Pedro para luego imprimir en qué posición de la lista se encuentra.

Tipo: Utilizar la función `index` para obtener la posicion del elemento en la lista.

[124]:
```
lista = ['Juan','Martin','Pedro','Agustina','Ana',]

for elemento in lista:
  if elemento == 'Pedro':
    posicion = lista.index('Pedro')
    print(f'Pedr fue encntrado en la posición {posicion} de la lista')
```

```
Pedr fue encntrado en la posición 2 de la lista
```

Parte 3 - Ejercicio de tarea 2

Escribe un programa que solicite al usuario ingresar un número entero positivo y luego imprima todos los números restantes hasta el 10, desestimando en caso de que el valor sea 5. Notificar en caso de que un valor superior a 10 sea ingresado.

[125]:
```
# Solicitar al usuario un número entero positivo
numero = int(input("Ingresa un número entero positivo: "))

# Validar que el número ingresado sea positivo
while numero != 5 and numero <= 10:
  print(numero)
  numero +=1
print('Su valor supera el valor 10')
```

```
Ingresa un número entero positivo: 20
Su valor supera el valor 10
```