PARA LEER ANTES DE CLASE

CodigoGO

BACKEND-PYTHON

Armando Ruiz

Condiciones
Sentencia If (Si)

Permite dividir el flujo de un programa en diferentes caminos. El if se ejecuta siempre que la expresión que comprueba devuelva True

```
In [1]:
if True:  # equivale a if not False
    print("Se cumple la condición")
    print("También se muestre este print")
```

```
Se cumple la condición
También se muestre este print
```

Podemos encadenas diferentes If

```
In [2]:
a = 5
if a == 2:
    print("a vale 2")
if a == 5:
    print("a vale 5")
```

```
a vale 5
```

O también anidar If dentro de If

```
In [3]:
a = 5
b = 10
if a == 5:
    print("a vale",a)
    if b == 10:
        print("y b vale",b)
```

```
a vale 5
y b vale 10
```

Como condición podemos evaluar múltiples expresiones, siempre que éstas devuelvan True o False

```
In [4]:
if a==5 and b == 10:
    print("a vale 5 y b vale 10")
```

```
a vale 5 y b vale 10
```

Sentencia Else (Sino)

Se encadena a un If para comprobar el caso contrario (en el que no se cumple la condición).

```
In [5]:
n = 11
if n % 2 == 0:
    print(n,"es un número par")
else:
    print(n,"es un número impar")
```

```
11 es un número impar
```

Sentencia Elif (Sino Si)

Se encadena a un if u otro elif para comprobar múltiples condiciones, siempre que las anteriores no se ejecuten.

```
In [6]:
comando = "OTRA COSA"
if comando == "ENTRAR":
    print("Bienvenido al sistema")
elif comando == "SALUDAR":
    print("Hola, espero que te lo estés pasando bien aprendiendo Python")
elif comando == "SALIR":
    print("Saliendo del sistema…")
else:
    print("Este comando no se reconoce")
```

```
Este comando no se reconoce
```

```
In [8]:
nota = float(input("Introduce una nota: "))
if nota >= 9:
    print("Sobresaliente")
elif nota >= 7:
    print("Notable")
elif nota >= 6:
    print("Bien")
elif nota >= 5:
    print("Suficiente")
else:
```

```
    print("Insuficiente")
```

```
Introduce una nota: 10
Sobresaliente
```

Es posible simular el funcionamiento de elif con if utilizando expresiones condicionales

```
In [9]:
nota = float(input("Introduce una nota: "))
if nota >= 9:
    print("Sobresaliente")
if nota >= 7 and nota < 9:
    print("Notable")
if nota >= 6 and nota < 7:
    print("Bien")
if nota >= 5 and nota < 6:
    print("Suficiente")
if nota < 5:
    print("Insuficiente")
```

```
Introduce una nota: 8
Notable
```

Iteraciones

Iterar significa realizar una acción varias veces. Cada vez que se repite se denomina iteración.

Sentencia While (Mientras)

Se basa en repetir un bloque a partir de evaluar una condición lógica, siempre que ésta sea True.

Queda en las manos del programador decidir el momento en que la condición cambie a False para hacer que el While finalice.

In [1]:

```
c = 0
while c <= 5:
    c+=1
    print("c vale",c)
```

```
c vale 1
c vale 2
c vale 3
c vale 4
c vale 5
c vale 6
```

Sentencia Else en bucle While

Se encadena al While para ejecutar un bloque de código una vez la condición ya no devuelve True (normalmente al final).

In [2]:

```
c = 0
while c <= 5:
    c+=1
    print("c vale",c)
else:
    print("Se ha completado toda la iteración y c vale",c)
```

```
c vale 1
c vale 2
c vale 3
c vale 4
c vale 5
```

```
c vale 6
Se ha completado toda la iteración y c vale 6
```

Instrucción Break

Sirve para "romper" la ejecución del While en cualquier momento. No se ejecutará el Else, ya que éste sólo se llama al finalizar la iteración.

In [4]:

```
c = 0
while c <= 5:
    c+=1
    if (c==4):
        print("Rompemos el bucle cuando c vale",c)
        break
    print("c vale",c)
else:
    print("Se ha completado toda la iteración y c vale",c)
```

```
c vale 1
c vale 2
c vale 3
Rompemos el bucle cuando c vale 4
```

Instrucción Continue

Sirve para "saltarse" la iteración actual sin romper el bucle.

In [9]:

```
c = 0
while c <= 5:
    c+=1
    if c==3 or c==4:
        # print("Continuamos con la siguiente iteración",c)
        continue
    print("c vale",c)
else:
    print("Se ha completado toda la iteración y c vale",c)
```

```
c vale 1
c vale 2
c vale 5
c vale 6
Se ha completado toda la iteración y c vale 6
```

Creando un menú interactivo

In [10]:

```
print("Bienvenido al menú interactivo")
while(True):
    print("""¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir""")
    opcion = input()
    if opcion == '1':
        print("Hola, espero que te lo estés pasando bien")
    elif opcion == '2':
        n1 = float(input("Introduce el primer número: "))
        n2 = float(input("Introduce el segundo número: "))
        print("El resultado de la suma es: ",n1+n2)
    elif opcion =='3':
        print("¡Hasta luego! Ha sido un placer ayudarte")
        break
    else:
        print("Comando desconocido, vuelve a intentarlo")
```

```
Bienvenido al menú interactivo
¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir
1
Hola, espero que te lo estés pasando bien
¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir
2
Introduce el primer número: 10
Introduce el segundo número: 5
El resultado de la suma es:  15.0
¿Qué quieres hacer? Escribe una opción
```

```
    1) Saludar
    2) Sumar dos números
    3) Salir
kdjsk
Comando desconocido, vuelve a intentarlo
¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir
3
¡Hasta luego! Ha sido un placer ayudarte
```

Recorriendo los elementos de una lista utilizando While

```
In [1]:
numeros = [1,2,3,4,5,6,7,8,9,10]
indice = 0
while indice < len(numeros):
    print(numeros[indice])
    indice+=1
```

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

Sentencia For (Para) con listas

```
In [2]:
for numero in numeros:  # Para [variable] en [lista]
    print(numero)
```

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

Modificar ítems de la lista al vuelo

Para asignar un nuevo valor a los elementos de una lista mientras la recorremos, podríamos intentar asignar al número el nuevo valor:

```
In [3]:
for numero in numeros:
    numero *= 10
```

```
In [4]:
numeros
```

```
Out[4]:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Sin embargo, ésto no funciona. La forma correcta de hacerlo es haciendo referencia al índice de la lista en lugar de la variable:

```
In [5]:
indice = 0
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for numero in numeros:
    numeros[indice] *= 10
    indice+=1
numeros
```

```
Out[5]:
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Podemos utilizar la función enumerate() para conseguir el índice y el valor en cada iteración fácilmente:

```
In [6]:
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for indice,numero in enumerate(numeros):
    numeros[indice] *= 10
numeros
```

```
Out[6]:
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

For con cadenas

```
In [7]:
cadena = "Hola amigos"
for caracter in cadena:
    print(caracter)
```

```
H
o
l
a

a
m
i
g
```

```
o
s
```

Pero debemos recordar que las cadenas son inmutables:

```
In [9]:
for i,c in enumerate(cadena):
    cadena[i] = "*"
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-9-8ba888c46579> in <module>()
      1 for i,c in enumerate(cadena):
----> 2     cadena[i] = "*"

TypeError: 'str' object does not support item assignment
```

Sin embargo siempre podemos generar una nueva cadena:

```
In [13]:
cadena2 = ""
for caracter in cadena:
    cadena2 += caracter * 2
```

```
In [11]:
cadena
```

```
Out[11]:
'Hola amigos'
```

```
In [14]:
cadena2
```

```
Out[14]:
'HHoollaa  aammiiggooss'
```

La función range()

Sirve para generar una lista de números que podemos recorrer fácilmente, pero no ocupa memoria porque se interpreta sobre la marcha:

```
In [15]:
for i in range(10):
    print(i)
```

```
0
1
```

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

```
In [16]:
range(10)
```

```
Out[16]:
range(0, 10)
```

```
In [17]:
for i in [0,1,2,3,4,5,6,7,9]:
    print(i)
```

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

Si queremos conseguir la lista literal podemos transformar el range a una lista:

```
In [18]:
list(range(10))
```

```
Out[18]:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Las Tuplas

Son unas colecciones parecidas a las listas, con la peculiaridad de que son inmutables.

In [14]:

```
tupla = (100,"Hola",[1,2,3],-50)
```

In [2]:

```
tupla
```

Out[2]:

```
(100, 'Hola', [1, 2, 3, 4], -50)
```

Indexación y slicing

In [3]:

```
tupla[0]
```

Out[3]:

```
100
```

In [4]:

```
tupla[-1]
```

Out[4]:

```
-50
```

In [6]:

```
tupla[2:]
```

Out[6]:

```
([1, 2, 3, 4], -50)
```

In [8]:

```
tupla[2][-1]
```

Out[8]:

```
4
```

Inmutabilidad

In [9]:

```
tupla[0] = 50
```

```
---------------------------------------------------------------------------
--
```

```
TypeError                                 Traceback (most recent call las
t)
<ipython-input-9-b45433b4cee9> in <module>()
----> 1 tupla[0] = 50

TypeError: 'tuple' object does not support item assignment
```

Función len()

```
In [10]:

len(tupla)
Out[10]:

4
In [13]:

len(tupla[2])
Out[13]:

3
```

Métodos integrados

index()

Sirve para buscar un elemento y saber su posición en la tupla. Da error si no se encuentra.

```
In [15]:

tupla.index(100)
Out[15]:

0
In [16]:

tupla
Out[16]:

(100, 'Hola', [1, 2, 3], -50)
In [17]:

tupla.index('Hola')
Out[17]:

1
In [18]:

tupla.index('Otro')
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<ipython-input-18-640d616163a2> in <module>()
```

```
----> 1 tupla.index('Otro')

ValueError: tuple.index(x): x not in tuple
```

count()

Sirve para contar cuantas veces aparece un elemento en una tupla.

```
In [19]:

tupla.count(100)
Out[19]:

1
In [20]:

tupla.count('Algo')
Out[20]:

0
In [21]:

tupla = (100,100,100,50,10)
In [22]:

tupla.count(100)
Out[22]:

3
```

append() ?

Al ser inmutables, las tuplas **no disponen** de métodos para modificar su contenido.

```
In [23]:

tupla.append(10)
---------------------------------------------------------------------------
--
AttributeError                            Traceback (most recent call las
t)
<ipython-input-23-758d195ec9d7> in <module>()
----> 1 tupla.append(10)

AttributeError: 'tuple' object has no attribute 'append'
```

Los diccionarios

Son junto a las listas las colecciones más utilizadas. Se basan en una estructura mapeada donde cada elemento de la colección se encuentra identificado con una clave única. Por tanto, no puede haber dos claves iguales. En otros lenguajes se conocen como arreglos asociativos.

In [1]:

```
vacio = {}
```

In [2]:

```
vacio
```

Out[2]:

```
{}
```

Tipo de una variable

In [3]:

```
type(vacio)
```

Out[3]:

```
dict
```

Definición

Para cada elemento se define la estructura -> clave:valor

In [4]:

```
colores = {'amarillo':'yellow','azul':'blue'}
```

También se pueden añadir elementos sobre la marcha

In [ ]:

```
colores['verde'] = 'green'
```

In [5]:

```
colores
```

Out[5]:

```
{'amarillo': 'yellow', 'azul': 'blue', 'verde': 'green'}
```

In [6]:

```
colores['azul']
```

Out[6]:

```
'blue'
```

In [7]:

```
colores['amarillo']
```

```
Out[7]:

'yellow'
```

Las claves también pueden ser números, pero son un poco confusas

```
In [8]:

numeros = {10:'diez',20:'veinte'}
In [9]:

numeros[10]
Out[9]:

'diez'
```

Modificación de valor a partir de la clave

```
In [10]:

colores['amarillo'] = 'white'
In [11]:

colores
Out[11]:

{'amarillo': 'white', 'azul': 'blue', 'verde': 'green'}
```

Función del()

Sirve para borrar un elemento del diccionario.

```
In [12]:

del(colores['amarillo'])
In [13]:

colores
Out[13]:

{'azul': 'blue', 'verde': 'green'}
```

Trabajando directamente con registros

```
In [14]:

edades = {'Hector':27,'Juan':45,'Maria':34}
In [15]:

edades
Out[15]:

{'Hector': 27, 'Juan': 45, 'Maria': 34}
In [16]:

edades['Hector']+=1
In [17]:

edades
```

```
Out[17]:

{'Hector': 28, 'Juan': 45, 'Maria': 34}
In [18]:

edades['Juan'] + edades['Maria']
Out[18]:

79
```

Lectura secuencial con for .. in ..

Es posible utilizar una iteraciín for para recorrer los elementos del diccionario:

```
In [19]:

for edad in edades:
    print(edad)
Maria
Hector
Juan
```

El problema es que se devuelven las claves, no los valores

Para solucionarlo deberíamos indicar la clave del diccionario para cada elemento.

```
In [20]:

for clave in edades:
    print(edades[clave])
34
28
45
In [21]:

for clave in edades:
    print(clave,edades[clave])
Maria 34
Hector 28
Juan 45
```

El método .items()

Nos facilita la lectura en clave y valor de los elementos porque devuelve ambos valores en cada iteración automáticamente:

```
In [23]:

for c,v in edades.items():
    print(c,v)
Maria 34
Hector 28
Juan 45
```

Ejemplo utilizando diccionarios y listas a la vez

Podemos crear nuestras propias estructuras avanzadas mezclando ambas colecciones. Mientras los diccionarios se encargarían de manejar las propiedades individuales de los registros, las listas nos permitirían manejarlos todos en conjunto.

```
In [24]:

personajes = []
In [1]:

p = {'Nombre':'Gandalf','Clase':'Mago','Raza':'Humano'}
In [26]:

personajes.append(p)
In [27]:

personajes
Out[27]:


[{'Clase': 'Mago', 'Nombre': 'Gandalf', 'Raza': 'Humano'}]
In [28]:

p = {'Nombre':'Legolas','Clase':'Arquero','Raza':'Elfo'}
In [29]:

personajes.append(p)
In [30]:

p = {'Nombre':'Gimli','Clase':'Guerrero','Raza':'Enano'}
In [31]:

personajes.append(p)
In [32]:

personajes
Out[32]:

[{'Clase': 'Mago', 'Nombre': 'Gandalf', 'Raza': 'Humano'},
 {'Clase': 'Arquero', 'Nombre': 'Legolas', 'Raza': 'Elfo'},
 {'Clase': 'Guerrero', 'Nombre': 'Gimli', 'Raza': 'Enano'}]
In [33]:

for p in personajes:
    print(p['Nombre'], p['Clase'], p['Raza'])
Gandalf Mago Humano
Legolas Arquero Elfo
Gimli Guerrero Enano
```