Listas

Un lista es un tipo de datos semejante a una tupla: una agrupación de datos bajo una sola variable, donde los datos pueden ser de distintos tipos y cada uno es identificado por un índice.

In [ ]:

```
# Para crear una lista se indica con [ ]
lista1 = [1, 2, 3, 4, 5]
# Se puede convertir una tupla en una lista
lista2 = list((1, 2, 3, 4, 5))
# Se puede crear una lista de un solo valor
lista3 = [10]
# Iclusive se puede crear una lista vacia
lista4 = []

print(lista1)
print(lista2)
print(lista3)
print(lista4)

print(type(lista1))
```

Una lista, a diferencia de una tupla, es un tipo de dato "*mutable*".

In [ ]:

```
print(lista1)
lista1[0] = 100
print(lista1)
```

Al igual que en una tupla, los elementos de una tupla se especifican con un índice:

In [ ]:

```
print("Primer valor de la lista:", lista1[0])
print("Ultimo valor de la lista:", lista1[-1])
```

Y al igual que una tupla, una lista es un *iterable*

In [ ]:

```
lista1 = [10, 20, 30, 40, 50]
for i in range(len(lista1)):
    print(lista1[i])
```

In [ ]:

```
for num in lista1:
    print(num)
```

In [ ]:

```
for i, v in enumerate([10, 20, 30, 40, 50]):
    print("Indice:", i, "\tValor:", v)
```

Una lista puede contenetr también datos de diferentes tipos:

In [ ]:

```
lista = [100, 3.1415, 'Nombre']
for data in lista:
    print(data)
```

Pero el número de funciones disponibles en una lista (por el hecho de ser mutable) es más amplio:

In [ ]:

```
lista = [1, 2, 3]
dir(lista)
```

Como se observa se tiene:

```
append(data)     -> Agrega al final de una lista un dato
clear()          -> Borra los elementos de una lista
copy()           -> Copia una lista en otra
count(val)       -> Cuenta cuantos valores val hay en una lista
extend(iterable) -> Agrega un iterable a una lista (varios valores en lugar de append que agre
ga solo uno)
index(val)       -> Retorna el indice donde se encuentra un valor
insert(index,obj)-> Inserta un objeto a partir de un índice en una lista
pop(index)       -> Retorna el valor de un indice y lo elimina de una lista
remove(val)      -> Elimina un valor de una lista
reverse()        -> Invierte el orden de los elementos de una lista
sort()           -> Ordena los elementos de una lista de menor a mayor
```

In [ ]:

```
# Se crea una lista vacia
lista = []
print(lista)
# Se agrega un elemento al final de una lista
lista.append(10)
print(lista)
# Se agrega una lista al final de una lista
lista.extend([20, 30, 40])
print(lista)
# Esto tambien se puede hacer con el operador '+'
lista = lista + [12, 14, 16]
print(lista)
# Y se puede duplicar una lista con un operador '*'
lista = lista * 2
print(lista)
# Cuantos veces esta presente el valor 20?
print("Veces que hay 20?:", lista.count(20))
# En que indice se encuentra un dato
print("Donde se encuentra el valor 16:", lista.index(16))
# Se puede insertar un dato en un indice (1 en el indice 0)
lista.insert(0, 1)
print(lista)
# Se puede extraer y eliminar el ultimo valor de una lista
val = lista.pop()
print(lista)
print("Dato extraido:", val)
# Se puede extraer y eliminar un dato en cualquier indice
val = lista.pop(3)
print(lista)
print("Dato extraido en el indice 3:", val)# Se puede eliminar un dato de una lista (si hay varios dato
s iguales, elimina el primero)
lista.remove(10)
print(lista)
# Se puede invertr el orden de los elementos de una lista
lista.reverse()
print(lista)
# Se pueden ordenar los elementos en orden ascendente
lista.sort()
print(lista)
# O ordenas de forma descendente
lista.sort(reverse=True)
print(lista)
# Y se puede eliminar todos los elementos de una lista
lista.clear()
print(lista)
```

Las listas tienen una característica particular: cuando se copia una lista, lo que se genera no es una nueva lista sino que se asocian dos listas:

In [ ]:

```
# Se crea una lista
lista1 = [1, 2, 3, 4, 5]
# Se copia esta lista en otra
lista2 = lista1
# Se modifica el primer valor de la lista1
lista1[0] = 10
# Se imprime la lista1
print(lista1)
# Se imprime la lista2
print(lista2)
```

Aunque solo se cambio el primer elemento de la lista1, el primer elemento de la lista2 también ha sido modificado. Esto porque al copiar una lista es otra ambas son lo mismo. La función *id()* retorna el número de identificación de una variable en el sistema.

In [ ]:

```
print("ID de lista1 =",id(lista1))
print("ID de lista2 =",id(lista2))
```

Esta conducta se puede evitar creando una nueva lista a partir de lista1 y copiarla a lista2, utilzando la función *list*

In [ ]:

```
# Se crea una lista
lista1 = [1, 2, 3, 4, 5]
# Se copia esta lista en otra
lista2 = list(lista1)
# Se modifica el primer valor de la lista1
lista1[0] = 10
# Se imprime la lista1
print(lista1)
# Se imprime la lista2
print(lista2)

print("ID de lista1 =",id(lista1))
print("ID de lista2 =",id(lista2))
```

Lista de listas

Se puede utilizar la funcion insert() para insertar un valor en una lista en un índice determinado:

```
index = 10
val = 100
lista.insert(index, val)
```

Sin embargo, tambien se puede insertar una lista dentro de una lista:

In [ ]:

```
lista = [1, 2, 3]
lista.insert(3, [1, 2, 3])
print(lista)
```

Esto tambien se puede hacer manualmente:

In [ ]:
```
lista = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
print(lista)
```

En este caso los índices de cada dado son anidados:

In [ ]:
```
print("Lista:", lista)
print("Segundo dato en lista:", lista[1])
print("Primer dato en el segundo dato de lista:", lista[1][0])
```

Esto permite construír listas dentro de listas para formar estructuras n-dimensionales. Por ejemplo, *lista* es un arreglo matricial de 3 x 3.

In [ ]:
```
lista = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print("{} {} {}".format(lista[0][0], lista[0][1], lista[0][2]))
print("{} {} {}".format(lista[1][0], lista[1][1], lista[1][2]))
print("{} {} {}".format(lista[2][0], lista[2][1], lista[2][2]))
```

Esto se puede reducir utilizando lazos anidados:

In [ ]:
```
for m in range(3):
    for n in range(3):
        print(lista[m][n],"",end='')
    print()
```

Listas por comprensión

La función *range()* permte generar una secuencia de números que pueden ser convertidos en una lista con la función *list()*. Sin embargo, la capacidad de la función range() es bastante limitada a una secuencia simple. Por ejemplo, si se quiere crear una lista con los valores de todos los numeros de 1 a 20 elevados al cuadrado, se necesita crear la lista utilizando un lazo:

In [ ]:
```
lista = []
for i in range(1,21):
    lista.append(i**2)

print(lista)
```

Python tiene una forma más eficiente de realizar utilizando *listas por comprensión*. La definición de este tipo de listas incluye una instrucción que contiene toda la lógica de un lazo for en un orden diferente:

In [1]:
```
lista = [num**2 for num in range(1,21)]
print(lista)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400]
```

Inclusive, una lista por comprensión puede contener una contrucción más compleja, como generar una lista de los cuadrados del rango numérico 1-20 que sean pares:

In [ ]:

```
lista = []
for i in range(1,21):
    val = i**2
    if val % 2 == 0:
        lista.append(i**2)

print(lista)
```

In [2]:

```
lista = [num**2 for num in range(1,21) if num % 2 == 0]
print(lista)
```

```
[4, 16, 36, 64, 100, 144, 196, 256, 324, 400]
```

Utilizando listas por comprensión se pueden crear conversiones de listas de forma compacta e inclusive filtrar valores:

In [ ]:

```
# Listado de temperaturas en grados centigrados
temperaturas_C = [22, 26, 28, 26, 27, 29, 28, 30, 31, 29, 31, 27, 26]

# Listado de temperaturas en grados Fahrenheit
temperaturas_F = [5/9 * C + 32 for C in temperaturas_C]

# zip() extrae los elementos de n listas y las guarda en n variables
print("         Celcius   Fahrenheit")
print("         ========  ==========")
i = 1
for tempC, tempF in zip(temperaturas_C, temperaturas_F):
    print("Dia {:2} : {:.2f} °C    {:.2f} F".format(i, tempC, tempF))
    i += 1

# Se crea una lista con los indices de las temperaturas mayores y se actualiza a numeros de dias (index
=0 es igual a dia=1)
dias_temp_max = [i+1 for i, temp in enumerate(temperaturas_C) if temp >= max(temperaturas_C)]
print("\nDias mas calurosos:")
for dia in dias_temp_max:
    print("  - Dia", dia)
```

El uso de listas por compresión en un buen ejemplo de un código breve, conciso, compacto: en resumen, un código "pythonico"