UTN AVELLANEDA
Tecnicatura Universitaria en Programación

# Listas avanzado

## Programación y Laboratorio I

Versión ‘22

UTN AVELLANEDA
Tecnicatura Universitaria en Programación

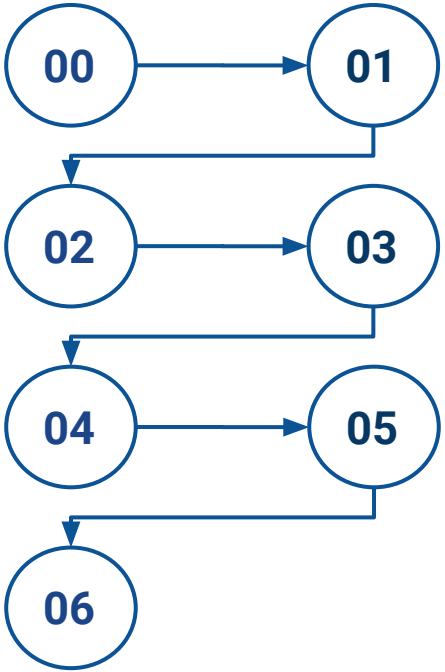
# Listas Avanzado

Funciones lambda

Recorrer listas

Metodos de list

Ordenamiento




¿Qué es una lista?

Copiar listas

Map, filter, reduce, ...

# Funciones lambda

Las funciones lambda o anónimas son una manera abreviada de escribir una función simple. Según la documentación de python:

"***...son solo una notación abreviada si eres demasiado perezoso para definir una función***"

# Funciones lambda

Una función encargada de recibir dos argumentos que retorna la suma de los mismos.

```
def sumar(a, b):
    return a+b
```

Se podría expresar en forma de una función lambda de la siguiente manera.

```
lambda a, b : a + b
```

# Funciones lambda

La función lambda no tiene un nombre, y por lo tanto para ser llamada se deberá:

```
print(lambda a, b : a + b(4,5)) # 9
```

Otra alternativa es asignarla a una variable:

```
sumar = lambda a, b : a + b
print(sumar(4,5)) # 9
```

# Operadores ternarios

Los operadores ternarios son más conocidos en Python como expresiones condicionales. Estos operadores evalúan si una expresión es verdadera o no.

```
resultado_if_true if a < b else resultado_if_false
```

```
mayor = lambda a, b : a if a > b else b
print(mayor(4,5)) # 5
```

# ¿Qué es una lista?

Son uno de los tipos más versátiles del lenguaje, ya que permiten almacenar un conjunto arbitrario de datos.

```
lista = ["Marty", "Emmett", "Biff"]
print(type(lista))    # <class 'list'>
print(lista[1])       # Emmett
lista[1] = "Jennifer"
print(lista[1])       # Jennifer
```

# Listas anidadas

La definición de lista en python habilita la posibilidad de poder tener como elemento de una lista cualquier objeto, inclusive otra lista.

```
lista_1 = [["Marty","McFly"], ["Emmett", "Brown"]]
```

UTN AVELLANEDA
Tecnicatura Universitaria en Programación

# Recorrer listas

Por ser la lista de python un objeto iterable es posible recorrerla utilizando un **for**

```
lista = ["Marty", "Emmett", "Biff"]
for elemento in lista:
    print(elemento)

# Marty
# Emmett
# Biff
```

# Copiar listas

No es posible copiar una lista simplemente escribiendo lista_2 = lista_1, ya que en ese caso lista_2 solo será una referencia a lista_1.

**Existen dos tipos de copias posibles:**

- Superficial (shallow copy)
- Profunda (deep copy)

# Copiar listas

Superficial (shallow copy), solamente se copian las referencias a los elementos contenidos en el objeto.

```
lista_1 = [["Marty","McFly"], "Emmett", "Biff"]
lista_2 = lista_1.copy()# lista[:] es equivalente
lista_1[0][0] = "MARTY"
print(lista_2)
# [['MARTY', 'McFly'], 'Emmett', 'Biff']
```

# Copiar listas

Deep copy, si el objeto contiene subobjetos estos se copian recursivamente.

```
from copy import deepcopy
lista_1 = [["Marty","McFly"], "Emmett", "Biff"]
lista_2 = deepcopy(lista_1)# Opción 1
lista_1[0][0] = "MARTY"
print(lista_2)
#[['Marty', 'McFly'], 'Emmett', 'Biff']
```

# append()

El método append() añade un elemento al final de la lista

```
lista = ["Marty", "Emmett", "Biff"]
lista.append("Jennifer")
print(lista)
#['Marty', 'Emmett', 'Biff', 'Jennifer']
```

# insert()

El método insert() añade un elemento en una posición o índice determinado

```
lista = ["Marty", "Emmett", "Biff"]
lista.insert(1,"Jennifer")
print(lista)
#['Marty', 'Jennifer', 'Emmett', 'Biff']
```

# extend()

El método extend() permite añadir una lista a la lista inicial.

```
lista = ["Marty", "Emmett", "Biff"]
lista.extend(["Jennifer","George"])
print(lista)
#['Marty', 'Emmett', 'Biff', 'Jennifer', 'George']
```

# pop()

El método pop() elimina y retorna el elemento ubicado en el índice pasado por parámetro, por defecto elimina el último elemento.

```
lista = ["Marty", "Emmett", "Biff"]
elemento_eliminado = lista.pop(1)
print(lista)
print(elemento_eliminado)
# ['Marty', 'Biff']
# Emmett
```

# remove()

El método remove() recibe como argumento un objeto y lo borra de la lista.

```
lista = ["Marty", "Emmett", "Biff"]
lista.remove("Marty")
print(lista)
#['Emmett', 'Biff']
```

# index()

El método index() recibe como parámetro un objeto y devuelve el índice de su primera aparición.

```
lista = ["Marty", "Emmett", "Biff"]
print(lista.index("Emmett"))
# 1
```

# enumerate()

Si necesitamos un índice acompañado con la lista, que tome valores desde **0** hasta **n-1**:

```
lista1 = ["Marty", "Emmett", "Biff"]

for indice,elemento in enumerate(lista1):
    print(indice,elemento)
```

# zip()

## Permite iterar múltiples listas a la vez.

```
lista1 = ["Marty", "Emmett", "Biff"]
lista2 = ["McFly", "Brown", "Tannen"]
lista3 = ["17", "71", "18"]
for elem_l1, elem_l2, elem_l3 in zip(lista1, lista2,lista3):
   print(elem_l1, elem_l2, elem_l3)
```

# map()

La función map() pasa como parámetros a una función a cada uno de los elementos de una lista, dando como resultado una nueva lista formada por los elementos que dicha función retorna.

```
lista = ["Marty", "Emmett", "Biff"]
lista_resultado = list(map(str.upper,lista))
print(lista)
#['Marty', 'Emmett', 'Biff']
print(lista_resultado)
#['MARTY', 'EMMETT', 'BIFF']
```

# filter()

La función filter() filtra una lista de elementos para los que una función devuelve True.

```
lista = [17, 71, 18]
lista_resultado = list(filter(lambda elem : elem >= 18, lista))
print(lista_resultado)
#[71, 18]
```

# reduce()

La función reduce() se utiliza principalmente para llevar a cabo un cálculo acumulativo sobre una lista de valores y devolver el resultado, está incluida en el módulo functools.

```
from functools import reduce
lista = [17, 71, 18]
suma = reduce(lambda x, y: x + y, lista)
print(suma)
# 106
```

# shuffle(lista)

Es un método del módulo **random**, se utiliza para mezclar una lista.

```
from random import shuffle
lista = ["Marty", "Emmett", "Biff"]
shuffle(lista)
print(lista) #['Emmett', 'Marty', 'Biff']
```

# sort()

El método sort() ordena los elementos de menor a mayor por defecto.

```
lista = ["Marty", "Emmett", "Biff"]
lista.sort()
print(lista)
#['Biff', 'Emmett', 'Marty']
```

# sort()

También permite ordenar de mayor a menor si se pasa como parámetro reverse=True.

```
lista = ["Marty", "Emmett", "Biff"]
lista.sort(reverse=True)
print(lista)
#['Marty', 'Emmett', 'Biff']
```

# sort(key=...)

Tiene un parámetro **key** para especificar una función que se llamará por cada elemento de la lista y su retorno se utilizará para hacer las comparaciones.

```
lista = ["Marty", "Emmett", "Biff"]
lista.sort(key=len)
print(lista)
#['Biff', 'Marty', 'Emmett']
```