Python

SECUENCIAS

ESTRUCTURAS DE DATOS

- Concepto
  - En ciencias de la computación las estructuras de datos son formas de organizar un conjunto de datos con el objetivo de facilitar su manipulación
- Importancia
  - Usadas frecuentemente en algoritmos

Ejemplos:

- **Lista de estudiantes de la universidad**

Carnet	Nombre	Carrera
801	Abarca A. Alf	IMT
215	Badilla B. Bin	IE
509	Campos C. Car	AMB
555	Dodero D. Dod	PI
378	Esquivel E. Era	MI
209	Gonzáles G. Gon	DI

- **Lista de trabajos a imprimir**

Impresora-10

#Trabajo	Nombre	Páginas	Estado
254512	Investigación #1	25	Imprimiendo
254810	Tarea de física	8	En cola
255974	Examen final	2	En cola

SECUENCIAS EN PYTHON

- **Secuencias en Python son estructuras de datos**
- **Tipos de secuencias**
  - **Listas**
  - **Tuplas**
  - **Strings**

- Listas

  [10, 30, 25, -10]

  ["ITCR", "UCR", "UNA"]

  [-80, "abc", True, 95.25, 100]

- Tuplas

  (2016, 2017, 2016) ("Alberto", 2010100)

- Strings

  "Hola jóvenes" "Gracias"

OPERACIONES COMUNES DE LAS SECUENCIAS

- Indexación
- Membresía
- Trozo (slice)
- Mínimo, máximo, largo

Python

Tipo de secuencia:

LISTAS

LISTAS

- Estructura de datos que contiene un grupo de elementos que pueden ser de tipos de datos diferentes

- Representación de listas: los elementos de una lista se encierran entre paréntesis cuadrados (corchetes) separados por comas

[10, 20, 5, -50]

["Python", "programación"]

["hola", 10.75, 100]

[] ← lista vacía

- Crear listas en estatutos de asignación

```
lista1 = [10, 20, 5, -50]

lista2 = []
```

INDEXACIÓN

- Cada elemento de una secuencia tiene un índice, es decir, un número entero que indica la posición del elemento dentro de la secuencia
- El índice lo usamos para trabajar con un elemento específico de la secuencia

0 1 2 ← índices positivos

[10, 15, 20] ← lista

-3 -2 -1 ← índices negativos

- Índice positivo: cuando se acceden los elementos de izquierda a derecha. El primer elemento tiene el índice 0, el segundo elemento tiene el índice 1, etc.
  - Es la forma mas común trabajar con las secuencias
- Índice negativo: cuando se acceden los elementos de derecha a izquierda. El primer elemento tiene el índice –1, el siguiente –2, etc.

- Sintaxis para obtener un elemento de una lista usando los índices

lista [índice]

- Índice: valor entero
- Índice: debe existir en la secuencia

```
>>> numeros = [10, 15, 20]
>>> print(numeros[0])
10
>>> print(numeros[3-2])
15
>>> print(numeros[1.0])
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    print(numeros[1.0])
TypeError: list indices must be integers, not float
>>> numeros[3]=10
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    numeros[4]=10
IndexError: list assignment index out of range
```

MEMBRESÍA

- Operador “in”
- Se usa para determinar si un elemento pertenece a una secuencia
  - Si el elemento pertenece a la secuencia el resultado es True
  - Si el elemento no pertenece a la secuencia, el resultado es False
- El operador “in” también puede usarse junto con el operador “not”: “not in”

```
lista = [30, 35, 4.4, 20, 1]
if elemento in lista:
    print(elemento,"está en la lista")
else:
    print(elemento,"no está en la lista")
```

TROZOS (SLICES)

- Un trozo (slice) permite manejar grupos de elementos de una secuencia

- Para obtener un trozo se usan dos índices separados por ":"

  secuencia[i:j]

  - Obtiene un trozo (grupo) de elementos de izquierda a derecha:
    - ✓ Desde el elemento con el índice i
    - ✓ Hasta el elemento con el índice j - 1

- Si el primer índice no se especifica se toma desde el primer elemento
- Si el segundo índice no se especifica se toma hasta el último elemento
- Si especificamos solo los ":" estamos trabajando con toda la secuencia

- Ejemplos:

```
>>> lista = ["a","b","c","d","e","f"]
>>> lista[1:3]
['b', 'c']
>>> lista[:4]
['a', 'b', 'c', 'd']
>>> lista[3:]
['d', 'e', 'f']
>>> lista[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

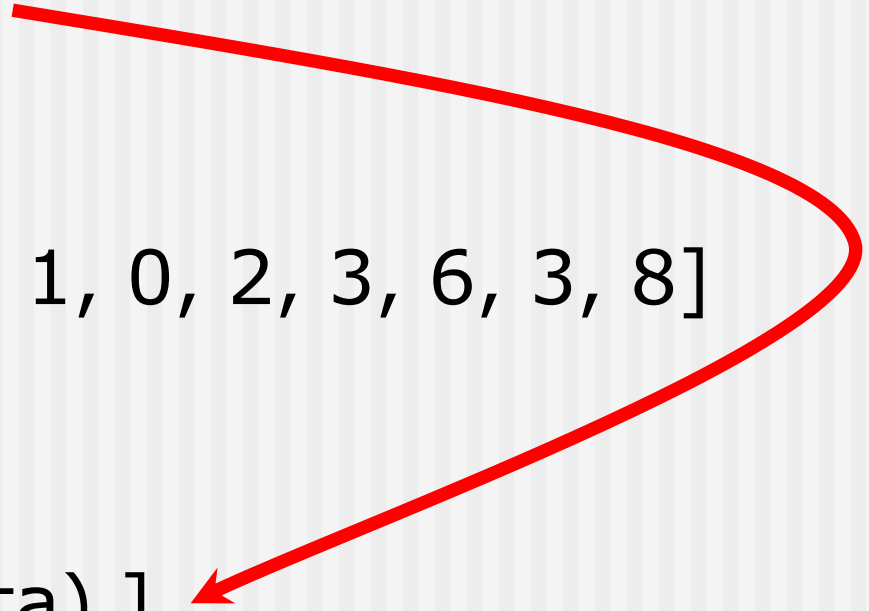
MINIMO, MAXIMO, LARGO

- Elemento de menor valor en una secuencia: función **min**
- Elemento de mayor valor en una secuencia: función **max**
- Para aplicar **min** y **max** los elementos deben ser homogéneos: mismo tipo de datos. Además las secuencias deben tener al menos un elemento
- Largo de una secuencia : función **len**

MÍNIMO, MÁXIMO Y LARGO DE UNA LISTA

```
>>> numeros=[100,20,50,75]
>>> min(numeros)
20
>>> max(numeros)
100
>>> len(numeros)
4
>>>
```

- ¿ qué sucede en la última instrucción ?



```
>>> lista = [5, 7, 8, 9, 1, 0, 2, 3, 6, 3, 8]
>>> len (lista)
11
>>> ele = lista[ len(lista) ]
```

LISTAS SON MUTABLES

- Los elementos dentro de una lista pueden ser modificados

- Usamos el operador [] en el lado izquierdo de una asignación

```
>>> fruta=["papaya", "manzana", "naranja"]
>>> fruta[1]="limón"
>>> print (fruta)
['papaya', 'limón', 'naranja']
```

AGREGAR ELEMENTOS: método append

- Método: es un tipo de función que realiza ejecuta operaciones sobre un objeto, por ejemplo sobre una lista.
- Sintaxis para usar métodos
  - Los métodos utilizan la notación de punto para hacer referencia al objeto:

Nombre del objeto . Nombre del método (argumentos)

- Método para agregar elementos al final de la lista
  - lista.append(elemento)

```
>>> a=[20, 30, 35]
>>> a.append(90)
>>> a
[20, 30, 35, 90]

>>> x=10
>>> a.append(x)
>>> a
[20, 30, 35, 90, 10]
```

BORRAR ELEMENTOS: del

```
>>> lista=["a","b","c"]
>>> del lista[1]
>>> print(lista)
['a', 'c']
```

OPERACIONES CON LISTAS

- Concatenar listas (unir): +

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> c=a+b
>>> print(c)
[1, 2, 3, 4, 5, 6]
```

- Repetir listas: *

```
>>> a=[1,2,3]
>>> print(a*3)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> l=[0]*4
>>> print(l)
[0, 0, 0, 0]
>>> lista = [2, 3, 8] * 0 # multiplicar por 0 o negativo da una lista vacía
>>> lista
[]
```

EJEMPLO USANDO LISTAS

- Contar el número de veces que un valor está en una lista

```
def cuenta(lista,valor):
    contador = 0  # para almacenar el resultado
    indice = 0  # para acceder cada elemento de la lista
    largo_lista = len(lista)
    while indice < largo_lista:
        if valor == lista[indice]:
            contador=contador+1
        indice = indice + 1
    return contador
print(cuenta([2, 74, 3, 74], 74))
2
```

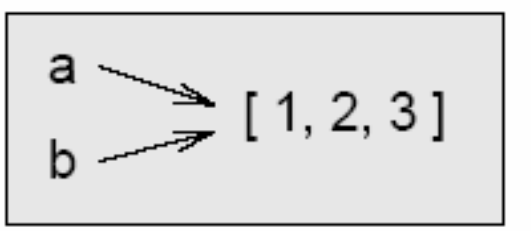
ALIAS Y COPIAS DE LISTAS

- Cuando se crean listas en una asignación, Python da a cada lista direcciones diferentes de memoria
  >>> a = [1, 2, 3]
  >>> b = [1, 2, 3]
- Pero cuando una variable tipo lista se asigna a otra variable, Python les asigna la misma dirección de memoria, a ello se le conoce como “alias”

```
>>> a = [1,2,3]

>>> b = a   # a y b son alias

>>> print (b, a)

[1, 2, 3] [1, 2, 3]
```

- Cuando tenemos “alias” los cambios hechos en una lista afectan implícitamente a la otra ya que están compartiendo la misma posición de memora

```
>>> a = [1,2,3]
>>> b = a
>>> b[0] = b[0] + 5
>>> print (b, a)
[6, 2, 3] [6, 2, 3]
```



CLONACIÓN DE LISTAS

- Es el proceso de asignar una lista a otra variable, de tal forma que las dos listas sean independientes:
  - Las listas tendrán direcciones de memoria diferentes, así los cambios en una no afectan a la otra

- Forma de clonar una lista

  Con operador de trozo (slice): “:”

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> print(b)
[1, 2, 3]
>>> b[0] = b[0] + 5
>>> print(a)
[1, 2, 3]
>>> print(b)
[6, 2, 3]
```

OTROS MÉTODOS SOBRE LISTAS

- lista.extend(L)
  - Extiende la lista concatenándole al final los elementos de la otra lista L

```
>>> a = [20, 30, 35]
>>> a.extend([1, 2])
>>> a
[20, 30, 35, 1, 2]
```

- lista.insert(i, x)
  - Inserta en la lista el elemento x en el índice i desplazando los demás elementos a la derecha. Índice puede ser negativo.

```
>>> a = [20, 30, 35]
>>> a.insert(0, 10)
>>> a
[10, 20, 30, 35]
>>> a.insert(3, 25)
>>> a
[10, 20, 30, 25, 35]
>>> a.insert(-1, 5)
>>> a
[10, 20, 30, 25, 5, 35]
```

En este método si el índice es positivo y no existe en la lista, se inserta al final

```
>>> a = [10, 20, 30, 25, 35]
>>> a.insert(10, 3)
>>> a
[10, 20, 30, 25, 35, 3]
```

En este método si el índice es negativo y no existe en la lista, se inserta al inicio

```
>>> a.insert(-10, 6)
>>> a
[6, 10, 20, 30, 25, 35, 3]
```

- lista.remove(x)
  - Borra de la lista el primer elemento que sea igual a x.

```
>>> a = [20, 30, 35, 30]
>>> a.remove(30)
>>> a
[20, 35, 30]
```

  - Si el elemento no existe da un error.

```
>>> a.remove(2)
Traceback (most recent call last):
  File "<pyshell#137>", line 1, in <module>
    a.remove(2)
ValueError: list.remove(x): x not in list
```

- lista.pop([i])
  - Borra de la lista el elemento en el índice i y lo retorna. Si no se especifica el índice, borra y retorna el último índice. Índice puede ser negativo

```
>>> a = [20, 30, 35]
>>> b = a.pop(1)
>>> b
30
>>> a
[20, 35]
>>> b = a.pop()
>>> b
35
>>> a
[20]
```

- Si el índice no existe da un error.

```
>>> b = a.pop(10)
Traceback (most recent call last):
  File "<pyshell#227>", line 1, in <module>
    b = a.pop(10)
IndexError: pop index out of range
```

- lista.index(x[, i[, j]])
  - Devuelve el primer índice donde se encuentre el elemento x en la lista. Si el elemento no existe da un error.
  - Opcionalmente podemos indicar que la búsqueda empieze en el índice i y termine en el índice j-1.

```
>>> a = [5, 20, 30, 35, 20, 25]
>>> a.index(20)
1
>>> a.index(20, 3, 6)
4
>>> a.index(10)
Traceback (most recent call last):
  File "<pyshell#254>", line 1, in <module>
    a.index(10)
ValueError: list.index(x): x not in list
```

- lista.count(x)
  - Cantidad de veces que el elemento x se encuentra en la lista

```
>>> a = [5, 20, 30, 35, 20, 25]
>>> a.count(20)
2
>>> a.count(50)
0
```

- lista.sort([reverse = True])
  - Ordena los elementos de la secuencia de menor a mayor (asc.). Si usamos la opción reverse los ordena de mayor a menor (desc.)

```
>>> lista = [3, 4, 1, -6, 10, 0]
>>> lista.sort()
>>> lista
[-6, 0, 1, 3, 4, 10]
>>> lista.sort(reverse = True)
>>> lista
[10, 4, 3, 1, 0, -6]
>>> lista.sort(reverse = True)
>>> lista
[10, 4, 3, 1, 0, -6]
```

- lista.reverse()
  - Invierte los elementos de la lista

```
>>> lista = [3, 4, 1, -6, 10, 0]
>>> lista.reverse()
>>> lista
[0, 10, -6, 1, 4, 3]
>>> lista.reverse()
>>> lista
[3, 4, 1, -6, 10, 0]
```

PRÁCTICA

- Haga la función lista_divisores que reciba un número entero >=1 y retorne una lista con sus divisores.

  Cuando el algoritmo debe crear una lista es común iniciar con una lista vacía y luego agregarle elementos.

Ejemplo del funcionamiento:

```
>>> lista_divisores(15)
[1, 3, 5, 15]
```

- Haga la función lista_random para crear una lista de n números aleatorios entre 0.0 y 1.0 (use el módulo random que tiene a su vez una función random que retorna valores aleatorios)

Ejemplo del funcionamiento:

```
>>> lista_random(5)
[0.75, 1.0, 0.56782, 0.0, 0.85]
```

"Las familias con su educación y ejemplo,

deben dar a las generaciones jóvenes un fuerte apoyo

para su humanidad y su crecimiento en la verdad, el amor y la honestidad."