DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

LISTAS

10145 - FUNDAMENTOS DE PROGRAMACIÓN PARA INGENIERÍA

RESUMEN DE CONTENIDOS

LISTAS

- Las listas (`list`) son una clase de objetos en Python que se utiliza para almacenar varios valores simultáneamente
- Se representan por corchetes (`[ ]`) y cada uno de sus elementos se separan por una coma (`,`)

```
valores = [1,2,3,4]
```

LISTAS

- Las listas (`list`) son una clase de objetos en Python que se utiliza para almacenar varios valores simultáneamente
- Se representan por corchetes (`[ ]`) y cada uno de sus elementos se separan por una coma (`,`)

```
valores = [1,2,3,4]
```

Nombre de variable de la lista

LISTAS

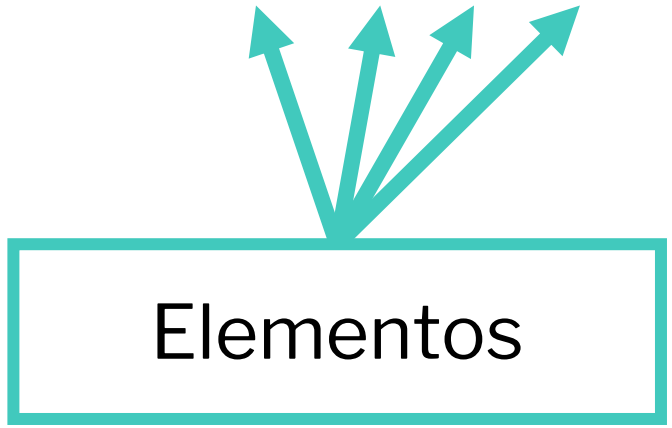
- Las listas (`list`) son una clase de objetos en Python que se utiliza para almacenar varios valores simultáneamente
- Se representan por corchetes (`[ ]`) y cada uno de sus elementos se separan por una coma (`,`)

```
valores = [1,2,3,4]
```

LISTAS

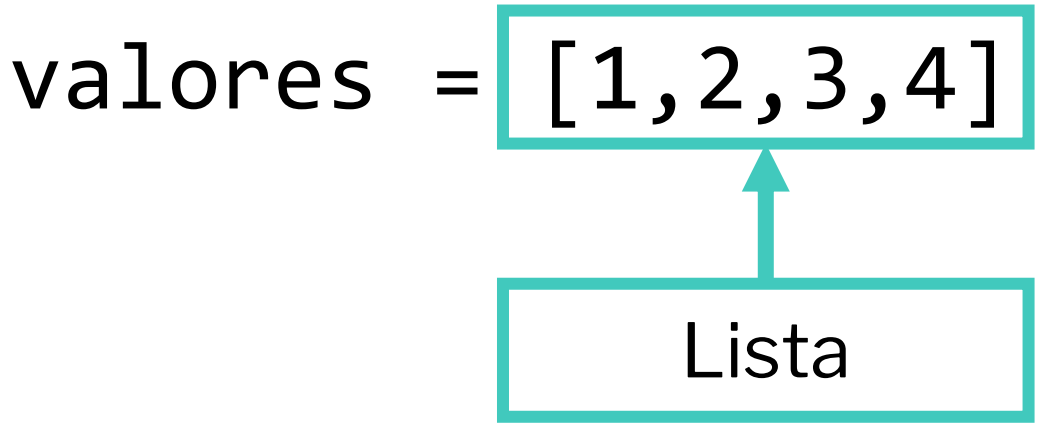
- Las listas (`list`) son una clase de objetos en Python que se utiliza para almacenar varios valores simultáneamente
- Se representan por corchetes (`[ ]`) y cada uno de sus elementos se separan por una coma (`,`)

```
valores = [1,2,3,4]
```



LISTAS

- Las listas (`list`) son una clase de objetos en Python que se utiliza para almacenar varios valores simultáneamente
- Se representan por corchetes (`[ ]`) y cada uno de sus elementos se separan por una coma (`,`)



LISTAS

- Pueden almacenar números, texto o combinaciones de tipos de datos en ellas

```
lista_ejemplo = ['Hola', True, 2.5, (2+3j)]
```

- Si se almacena una expresión esta se evaluará y luego se guardará

```
lista_operaciones = [2*4, 3+34//3, int(3.4)]
```

LISTAS

- Se puede declarar una lista vacía

```
otra_lista_de_ejemplo = []
```

- Una lista puede representar datos del mundo real

```
persona_1 = ['12345678-9', 'Ruperto',
    'Marchant Pereira', 700000]
```

- Una lista puede tener dentro de si otra lista

```
lista_indefinida = [1, 1.2, ['hola',
    'adios']]
```

INDEXACIÓN

Indice	0	1	2	3	4	5	6	7	8
Elemento	100	101	102	103	104	105	106	107	108

```
lista = [100,101,102,103,104,105,106,107,108]
```

- Para consultar u obtener elementos de una lista, es posible utilizar la **posición** del elemento entre **paréntesis cuadrados**
- Por ejemplo, para la lista anterior:

```
>>> print(lista[2])
102
```

INDEXACIÓN

- La indexación parte parte desde una posición 0 hasta el largo de la lista – 1

- Tomemos la siguiente lista de 9 elementos como ejemplo:

```
lista      =
    [100,101,102,103,104,105,106,107,108]
```

- En este caso, su primer valor (100) estará en la posición 0 y el último carácter en la posición 8 de la lista

OPERACIONES SOBRE LISTAS

- Cuando trabajamos con listas existen existen cuatro operaciones básicas que debemos conocer:
  - Consultar: Obtener el valor de un elemento de la lista
  - Agregar: Insertar un nuevo valor a la lista
  - Actualizar: Cambiar el valor en una posición de la lista
  - Borrar: Eliminar un valor existente en la lista
- Ya conocemos cómo consultar, por lo que vamos a ver el resto de las operaciones

AGREGAR

- Si deseamos añadir un elemento a una lista, podemos hacerlo utilizando el **método `.append()`**

```
>>> lista = [True, False, True]
>>> lista.append(True)
>>> print(lista)
[True, False, True, True]
```

- Para invocar el método, es necesario entregar como **parámetro** el **elemento a agregar**
- A través de este método es posible agregar elementos **al final de una lista**

ACTUALIZAR

- Es posible modificar un valor, a partir de su posición, simplemente usando el **operador asignación**

```
>>> lista = [True, False, True]
>>> lista[2] = False
>>> print(lista)
[True, False, False]
```

- De esta forma modificamos el valor del elemento en una posición dada, al **sobrescribir su valor**

BORRAR

- También es posible eliminar valores de la lista, en base a su posición utilizando el método `.pop()`

```
>>> lista = [True, False, True]
>>> lista.pop(1)
False
>>> print(lista)
[True, True]
```

- De esta forma eliminamos un elemento de la lista, en base a la **posición del elemento**

OTROS MÉTODOS INTERESANTES

- Existen otros métodos útiles de las listas que vale la pena mencionar:
  - `<list>.count()` cuenta las apariciones de un valor en la lista
  - `<list>.index()` entrega el índice de la primera aparición de un elemento
  - `<list>.remove()` elimina la primera aparición de un valor en la lista
  - `<list>.insert()` inserta un elemento en una posición determinada
  - `<list>.sort()` ordena una lista según los valores de sus elementos

RECORRIENDO LISTAS

- Cuando tenemos que realizar cambios sobre cada elemento de la lista tendremos que iterar, a fin de revisar cada uno de los elementos de ella
- Por lo que el while que vimos la clase pasada nos será de utilidad nuevamente
- Sin embargo, ¿Cuál es la condición de parada para recorrer una lista?
- ¿Me falta algún dato para poder realizar la iteración?

RECORRIENDO LISTAS

- Una forma común de iterar sobre listas es ir recorriendo los índices desde 0 hasta el último índice, y para ello, necesitamos saber la cantidad de elementos que la lista contiene

- Para ello existen en Python la función `len()` que recibe como entrada una lista (u otro tipo de dato compuesto) y entrega el número de elementos contenidos en ella

RECORRIENDO LISTAS

- Ejemplos de `len()`:

```
>>> lista=[100,101,102,103,104,105,106,107,108]
>>> len(lista)
9
>>> lista_2      =[]
>>> len(lista_2)
0
```

- Vale la pena notar que len() devuelve el valor del último índice + 1

RECORRIENDO LISTAS

- ¿Qué hace el siguiente programa?

```
# ENTRADA
lista = [100,101,102,103,104,105,106,107,108]
# PROCESAMIENTO
largo = len(lista)
i = 0
while i < largo:
    lista[i] = lista[i]     ** 2
    i = i + 1
# SALIDA
print(lista)
```

RECORRIENDO LISTAS

- ¿Qué diferencia tiene la salida de este programa con el anterior?

```
# ENTRADA
lista = [100,101,102,103,104,105,106,107,108]
# PROCESAMIENTO
largo = len(lista)
i = 0
while i < largo:
    lista[i] = lista[i]     ** 2
    i = i + 1
    # SALIDA
    print(lista[i])
```

CONSIDERACIONES FINALES

- Las listas son tipos de datos que permiten representar y almacenar un sinnúmero de abstracciones

- Además de los métodos vistos hoy, existen métodos y funciones para ordenar, separar y realizar otras transformaciones útiles sobre listas

EJERCICIOS

EJERCICIO 1

- Escriba un programa que permita crear una lista de palabras. Primero, el programa debe solicitar el número de palabras que tendrá la lista y luego solicitar el mismo número de palabras para crear la lista. Por último, el programa debe mostrar por consola cada palabra de la lista.

EJERCICIO 1

- DATOS ENTRADA:
  - Número de palabras, palabras.
- DATOS SALIDA:
  - Las palabras de la lista.
- ALGORITMO:
  1. Solicitar el número de palabras.
  2. Inicializar iterador (=0) y una lista vacía (=[]).
  3. Mientras el iterador sea menor al número de palabras, realizar 4. En caso contrario ir a 5.
  4. Solicitar una palabra, agregar la palabra a la lista e incrementar el valor del iterador.
  5. Inicializar iterador (=0)
  6. Mientras el iterador sea menor al número de palabras, realizar 7. En caso contrario ir a 8.
  7. Mostrar por consola la palabra de la lista ubicada en la posición del valor de la variable iteradora.
  8. Fin.

EJERCICIO 1

```
# DATOS DE ENTRADA
numero_palabras = int(input("Ingrese el número de palabras de la lista:
"))

# PROCESAMIENTO
lista_palabras = []
i = 0
while i < numero_palabras:
    palabra_ingresada = input("Ingrese una palabra: ")
    lista_palabras.append(palabra_ingresada)
    i = i + 1

# SALIDA
i = 0
while i < numero_palabras:
    print(lista_palabras[i])
    i = i + 1
```

EJERCICIO 2

- Implemente un programa que reciba un lista números enteros y que luego solicite un número entero e indique si el número ingresado se encuentra o no en la lista.

EJERCICIO 2

- DATOS ENTRADA:
  - Lista de números, número buscado.
- DATOS SALIDA:
  - Mensaje indicando si el número se encuentra o no en la lista.
- ALGORITMO:
  1. Solicitar una lista de números enteros.
  2. Solicitar un número a buscar.
  3. Inicializar una bandera (=False) e iterador (=0).
  4. Mientras el iterador sea menor que el largo de la lista, realizar 5 y 6. En caso contrario ir a 7.
  5. Si el número en la posición del valor del iterador es igual al número buscado, entonces asignarle el valor True a la bandera.
  6. Incrementar el valor del iterador.
  7. Si el valor de la bandera es True, informar por consola que el número se encuentra en la lista. En caso contrario, informar que el número no se encuentra en la lista.

EJERCICIO 2

```
# DATOS DE ENTRADA
lista_numeros = eval(input("Ingrese una lista de números enteros: "))
numero_buscado = int(input("Ingrese el número a buscar en la lista: "))
# PROCESAMIENTO
numero_encontrado = False
 i = 0
while i < len(lista_numeros ):
   if numero_buscado == lista_numeros[i]:
      numero_encontrado = True
   i = i + 1  # SALIDA
if numero_encontrado:
   print("El número", numero_buscado, "pertenece a la lista")
else:
   print("El número", numero_buscado, "no pertenece a la lista")
```

EJERCICIO 3

- Escriba un programa que pida al usuario que introduzca una lista (no vacía) de números no necesariamente ordenada y que busque el mayor y el menor número que se ingresó. Además, debe mostrar el promedio de los elementos de la lista.

- Ejemplo:

  lista_numeros = [12,4,34,1,2,52,1,54,98,5,235,76,57,87,123,32,32]

  menor = 1

  mayor = 235

  promedio = 53.2352941176

EJERCICIO 3

- DATOS ENTRADA:
  - Lista de números.
- DATOS SALIDA:
  - Número mayor, número menor y promedio de los números de la lista.
- ALGORITMO:
  1. Solicitar una lista de números.
  2. Inicializar las variables mayor, menor y suma con el primer elemento de la lista. Inicializar un iterador (=0).
  3. Mientras el iterador sea menor que el largo de la lista, realizar 4, 5, 6 y 7. En caso contrario ir a 8.
  4. Si el número en la posición del valor del iterador es menor al menor número encontrado hasta la iteración actual, entonces asignar el valor a menor.
  5. Si el número en la posición del valor del iterador es mayor al mayor número encontrado hasta la iteración actual, entonces asignar el valor a mayor.
  6. Sumar a la variable acumuladora el número en la posición del valor del iterador .
  7. Incrementar el iterador.
  8. Calcular promedio.
  9. Mostrar por consola que el número mayor, el número menor y el promedio.

EJERCICIO 3


```
# DATOS DE ENTRADA
lista_numeros = eval(input("Ingrese una lista de números: "))
# PROCESAMIENTO
mayor = lista_numeros[0]
menor = lista_numeros[0]
suma = lista_numeros[0]
largo = len(lista_numeros)
i = 1
while i < largo:
  if lista_numeros[i] > mayor:
    mayor = lista_numeros[i]
  if lista_numeros[i] < menor:
    menor = lista_numeros[i]
  suma = suma + lista_numeros[i]
  i = i + 1
promedio = suma / largo
# SALIDA
print("El mayor número de la lista es: ", mayor)
print("El menor número de la lista es: ", menor)
print("El promedio de la lista de números es: ", promedio)
```

EJERCICIO 4

- Construya un programa que tenga como entrada una lista de números e informe si todos los elementos de la lista son distintos o no.

EJERCICIO 4

- DATOS ENTRADA:
  - Lista de números.
- DATOS SALIDA:
  - Número mayor, número menor y promedio de los números de la lista.
- ALGORITMO:

1. Solicitar una lista de números.
2. Inicializar una bandera (=True) y un i (=0).
3. Mientras el iterador i sea menor que (largo de la lista – 1), realizar 4, 5 y 8. En caso contrario ir a 9.
4. Inicializar un segundo iterador j (=i + 1).
5. Mientras el iterador j sea menor que el largo de la lista, realizar 6 y 7. En caso contrario ir a 8.
6. Si el número en la posición i de la lista es igual a número en la posición j de la lista, entonces asignar False a la bandera.
7. Incrementar el iterador j.
8. Incrementar el iterador i.
9. Si el valor de la bandera es True, informar por consola que los números son distintos. En caso contrario, informar que se repite algún número.

EJERCICIO 4


```
# DATOS DE ENTRADA
lista_numeros = eval(input("Ingrese una lista de números: "))
# PROCESAMIENTO
todos_diferentes = True
largo_lista = len(lista_numeros)
i = 0
while i < (largo_lista – 1):
    j = i + 1
    while j < largo_lista:
        if lista_numeros[i] == lista_numeros[j]:
            todos_diferentes = False
        j = j + 1
    i = i + 1
# SALIDA
if todos_diferentes:
    print("Todos los números de la lista son diferentes")
else:
    print("Se repite un número en la lista")
```

EJERCICIOS PROPUESTOS

- Construya un programa en Python que solicite una cantidad de números enteros (n) y luego pida al usuario cada número y los agregue a una lista, para luego imprimir la lista

- Construya un programa en Python que encuentre el valor más pequeño en una lista de números

- Construya un programa en Python que encuentre el mayor valor en una lista de números

EJERCICIOS PROPUESTOS

- Construya un programa en Python que entregue una lista sin sus elementos repetidos
  - Pista: A veces es más fácil hacer este tipo de operaciones usando una lista auxiliar

- Construya un programa en Python que ordene los elementos de ella de menor a mayor

¿CONSULTAS?

TAREAS PARA TRABAJO AUTÓNOMO

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

1. Construya un programa que ordene una lista, sin usar métodos, ni funciones para ordenar, ni obtener mínimos o máximos

1. Construya un programa que encuentre el elemento que más se repite en una lista:
   - Construya el programa sin usar los métodos .count(), .sort() y .index()
   - Construya el programa sin restricciones de métodos

2. Revise las guías 1, 2, 3, 4 y 5 junto con las lecturas de coordinación (disponibles en Github y drive).