Cronograma

- Solución de algunos ejercicios de la clase pasada
- Nuevo tipo de dato
  - Listas
    - Acceso
    - Matrices
    - Particionado
    - Operaciones
    - Funciones
- Ejercicios

Posibles soluciones

Ejercicio 5 (clase pasada):

```
def isosceles(ancho):
    for i in range(1, ancho):
        for j in range(i):
            print("* ", end="")
        print()
    for i in range(ancho, 0,-1):
        for j in range(i):
            print("* ", end="")
        print()

ancho = int(input("Anchura del triangulo: "))

isosceles(ancho)
```

Nuevos tipos de datos

- Vimos algunos tipos básicos
  - Como los números: enteros, decimales y complejos.
  - Las cadenas de texto
  - Los tipos booleanos
- Nos queda por ver las colecciones
  - Listas
  - Tuplas
  - Diccionarios

Listas

- Es un tipo de colección ordenada que nos permite agrupar elementos.
- En Python **NO** tenemos Arrays o Vectores
- Las listas pueden contener cualquier tipo de dato: números, cadenas, booleanos, y también listas
- Podemos crear una lista indicando sus elementos, separado por coma y encerrarlos entre [ ]
- Por ejemplo:
  - **mi_lista = [66, False,”Verdad”, [1,2,3,4]]**

Listas (2)

- Las listas tienen su propio tipo de objeto ***list***
  - type(mi_lista)
    - <class 'list'>
- Para acceder a los elementos de una lista se puede hacer a través de su índice
  - mi_lista[0] → 66
- Tener en cuenta que el primer índice de una lista es el 0 y el ultimo es: **largo de la lista - 1.**

Listas (3)

Listas anidadas

- Una lista anidada es una lista que aparece como elemento dentro de otra lista.
  - lista = ["hola", 2.0, 5, [10, 20]]
  - print lista[3] → [10, 20]
- Para extraer los elementos de la lista anidada utilizamos doble paréntesis recto.
  - print lista[3][0] → 10

Listas (4)

Índices negativos

- Al igual que los índices de las variables de tipo ***str*** podemos utilizar valores negativos.
- Si utilizamos un valor negativo quiere decir que estamos contando desde el final.
  - mi_lista = [66, True,"Verdad", [1,2,3,4]]
  - mi_lista[-1] →[1,2,3,4]
  - mi_lista[-3] →True
  - mi_lista[-len(mi_lista)] →66
  - mi_lista[len(mi_lista)] → Out of range!

Listas (5)

Particionando listas

- También podemos seleccionar una sublista de la lista inicial utilizando la notación de corte : (igual que en strings)
  - **mi_lista[inicio:fin]** obtenemos una **nueva lista** con los elementos desde la posición inicio hasta la posición fin sin incluir este último.
- También podemos utilizar este mecanismo para modificar la lista, e incluso se puede cambiar su largo

Listas (6)

Ejemplos

```
mi_lista = [66, False,"Verdad", [1,2,3,4]]
print (mi_lista)  → [66, False, 'Verdad', [1, 2, 3, 4]]

mi_lista[0:2] = [1,True]
print (mi_lista) → [1, True, 'Verdad', [1, 2, 3, 4]]

print (len(mi_lista))   → 4
mi_lista[0:2] = [2,3,4,5]

print (mi_lista)
print (len(mi_lista))   ├→ 6
```

Listas (7)

- Podemos **reemplazar** varios elementos
  - lista = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’]
  - lista[1:3] = [’x’, ’y’]
  - lista → [’a’, ’x’, ’y’, ’d’, ’e’, ’f’]

- Podemos **eliminar** elementos de una lista asignando una lista vacía.
  - lista = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’]
  - lista[1:3] = []
  - lista → [’a’, ’d’, ’e’, ’f’]

Listas (8)

- Podemos incluir elementos en el medio de la lista
  - lista = [’a’, ’d’, ’f’]
  - lista[1:1] = [’b’, ’c’]
  - lista → [’a’, 'b', 'c',’d’, ’f’]

  - lista = [’a’, ’d’, ’f’]
  - lista[1:1] = [45, False]
  - lista → ['a', 45, False, 'd', 'f']

Operaciones con listas

- Con el tipo de datos **lista** también podemos utilizar los operadores matemáticos.
  - El operador + concatena listas:
    - a = [1, 2, 3]
    - b = a + [4,5,6]
  - El operador * repite una lista un número dado de veces
  - a = [0,1,2]
  - b = a * 3
  - B → [0,1,2,0,1,2,0,1,2]

Funciones para listas

- Podemos utilizar la función len() para saber el largo de la lista.
  - mi_lista = [2, 3, 4, 5, [1, 2, 3, 4]]
  - len(mi_lista) → 5
  - len (mi_lista[4]) → 4
  - len (mi_lista[0]) → ERROR: Object of type 'int' has no len()
- Podemos agregar un elemento al final de una lista utilizando la operación append()
  - mi_lista.append(0)
  - mi_lista → [2, 3, 4, 5, [1, 2, 3, 4],0]

Más funciones del tipo *list*

- Para insertar un nuevo valor en la posición cuyo índice es **k** (y desplazar un lugar el resto de la lista) se utiliza la operación ***insert()***.
  - mi_lista → [2, 3, 4, 5, [1, 2, 3, 4],0]
  - mi_lista.insert(4,6)
  - mi_lista → [2, 3, 4, 5,6, [1, 2, 3, 4],0]

- Contar ocurrencias de un elemento: **count**
  - lista = ['a', 'd', "a", 'f']
  - print (lista.count("a")) → 2

Más funciones del tipo *list (2)*

- Para preguntar si un valor determinado es un elemento de una lista usaremos la operación **in**
  - mi_lista → [2, 3, 4, 5, [1, 2, 3, 4], 2]
  - 5 **in** mi_lista → True
  - 6 **in** mi_lista → False
  - print("esta a?: {}".format("a" in lista))
- Para averiguar la posición de un valor dentro de una lista se utiliza la operación ***index()***
  - mi_lista.index(2) → 0
  - mi_lista.index(0) →
    - ValueError: list.index(x): x not in list

Borrado en un lista

- Si queremos eliminar un elemento de cierto índice podemos utilizar **remove**
  - a = [’uno’, ’dos’, ’tres’]
  - a.remove(a[0])
  - a →['dos','tres']
- O también permite eliminar por ocurrencias, pero elimina sólo la primera
  - lista = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’, 'a']
  - lista.remove('a')
  - lista → [’b’, ’c’, ’d’, ’e’, ’f’, 'a']

Borrado en una lista (2)

```
x = [1, 2, 3, 4, 2, 2, 3]
def remove_values_from_list(the_list, val):
        while val in the_list:
            the_list.remove(val)
remove_values_from_list(x, 2)
>>> x -> [1, 3, 4, 3]

a = [-1, 1, 66.25, 333, 333, 1234.5, 333]
a = [item for item in a if item != 333]
```

Ordenar Listas

Python provee dos operaciones para obtener una lista ordenada a partir de una lista desordenada.

- Para dejar la lista original intacta pero obtener una nueva lista ordenada a partir de ella se usa la función ***sorted()***
  - mi_lista → [7, 3, 4, 5, 0]
  - lista_ordenada = sorted(mi_lista)
  - lista_ordenada → [0, 3, 4, 5, 7]
- Para modificar directamente la lista original usaremos la operación sort()
  - ds=[5,3,4,5]
  - ds.sort()
  - ds → [3,4,5,5]

Ordenar Listas (2)

```
mi_lista = [7, "a", 3, False, 4, 5, 0]
mi_lista.sort()
print (mi_lista)
```

TypeError: unorderable types: str() < int()

(In)mutabilidad

- En **Python** existen tipos de datos **inmutables** y tipos de datos **mutables**.
- Una variable de un tipo **inmutable** es un objeto cuyo estado no puede ser modificado una vez creado.
- Los números, los strings y las tuplas (aún no vimos éste tipo) son inmutables.
- Por el contrario, las listas son mutables.

(In)mutabilidad (2)

```
>>> a = 5
>>> b = 'pradera'
```

- La variable *a* contiene un número; la variable *b* contiene un string. Hemos dicho que los números y los strings son inmutables. ¿Significa eso que no podemos modificar ni *a* ni *b*?

```
>>> a = 6
>>> b = 'casa'
```

- ¡Por supuesto que no! La inmutabilidad es otra cosa entonces

(In)mutabilidad(3)

- Bien, ya hemos visto con los ciclos for, que un string puede ser recorrido como una secuencia (tal como las listas y las tuplas que veremos en próximamente).
- Dado que un string es una secuencia de caracteres, podemos acceder a sus caracteres de forma individual:

```
>>> serie = 'la casa de la pradera'
>>> serie[3]
'c'
```

- Pero no podemos modificarlos:

```
>>> serie[3] = 'k'
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
   serie[3] = 'k'
TypeError: 'str' object does not support item assignment
```

(In)mutabilidad (4)

- Veamos un par de ejemplos.
  - Para strings (inmutables):

```
a= "pepe"
b = a
print(a, b)
a = "maria"
print(a, b)
```

Salida

```
pepe pepe
maria pepe
```

  - Para listas (mutables)

```
l1= [10, 11]
l2 = l1
print(l1, l2)
l1[0] = 20
print(l1,l2)
```

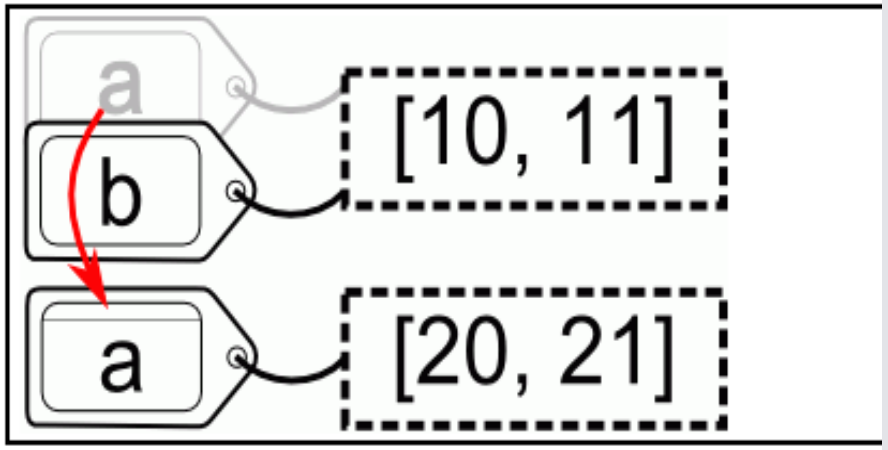
Salida

```
[10, 11] [10, 11]
[20, 11] [20, 11]
```

(In)mutabilidad (5)

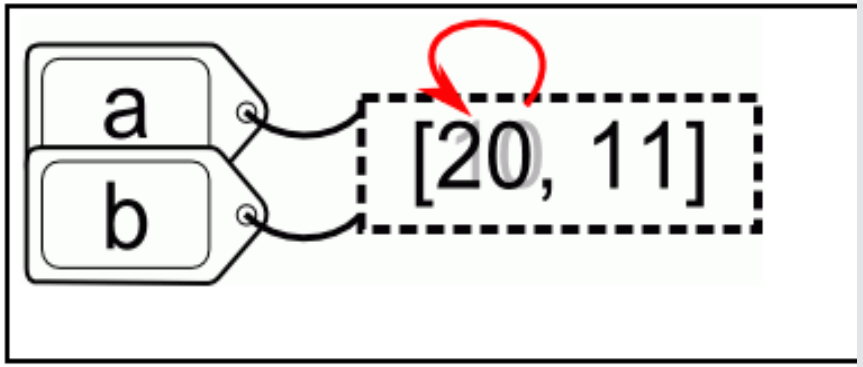
A nivel de variable de tipo lista (inmutable)

```
>>> a = [10, 11]
>>> b = a
>>> a, b
([10, 11], [10, 11])
>>> a = [20, 21]
>>> a, b
([20, 21], [10, 11])
```



A nivel de elementos, como ya vimos, es mutable.

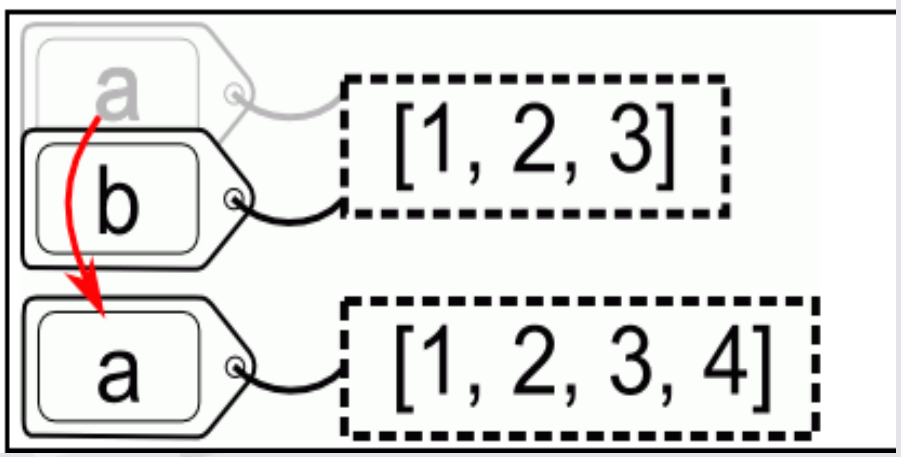
```
>>> a = [10, 11]
>>> b = a
>>> a, b
([10, 11], [10, 11])
>>> a[0] = 20
>>> a, b
([20, 11], [20, 11])
```



(In)mutabilidad (6)

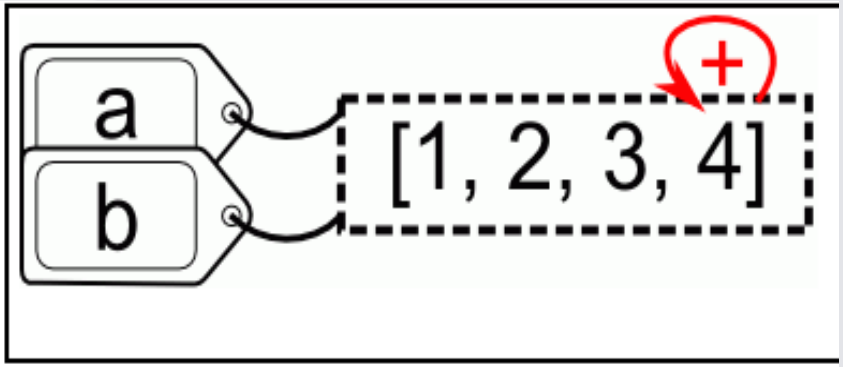
- Operador “+”: Copia (inmutable)

```
>>> a = [1, 2, 3]
>>> b = a
>>> a, b
([1, 2, 3], [1, 2, 3])
>>> a = a + [4]
>>> a, b
([1, 2, 3, 4], [1, 2, 3])
```



- Función ***append*** - Referencia (mutable)

```
>>> a = [1, 2, 3]
>>> b = a
>>> a, b
([1, 2, 3], [1, 2, 3])
>>> a.append(4)
>>> a, b
([1, 2, 3, 4], [1, 2, 3, 4])
```



Ejercicios

- **Ejercicio 1.** Dada una lista de números enteros, escribir una función que:
  - a) Devuelva la sumatoria y el promedio de los valores.
  - b) Devuelva una lista con el factorial de cada uno de esos números.
- **Ejercicio 2.** Dada una lista de números enteros y un entero k, escribir una función que:
  - a) Devuelva tres listas, una con los menores, otra con los mayores y otra con los iguales a k.
  - b) Devuelva una lista con aquellos que son múltiplos de k.

Ejercicios (2)

- **Ejercicio 3:** Escriba una función que reciba una lista de palabras y devuelva una segunda lista, igual a la primera, pero al revés (no se trata de escribir la lista al revés, sino de crear una lista distinta)

  Ejemplo:

  Dígame cuántas palabras tiene la lista: 4

  Dígame la palabra 1: Alberto

  Dígame la palabra 2: Carmen

  Dígame la palabra 3: Benito

  Dígame la palabra 4: Daniel

  La lista creada es: ['Alberto', 'Carmen', 'Benito', 'Daniel']

  La lista inversa es: ['Daniel', 'Benito', 'Carmen', 'Alberto']

Ejercicios (3)

- **Ejercicio 4:** Escriba una función que reciba dos listas de palabras y que imprima las siguientes listas:
  - Lista de palabras que aparecen en las dos listas.
  - Lista de palabras que aparecen en la primera lista, pero no en la segunda.
  - Lista de palabras que aparecen en la segunda lista, pero no en la primera.
  - Lista de palabras que aparecen en ambas listas.

Ejercicios (4)

- Ejemplo:

  La primera lista es: ['Carmen', 'Alberto', 'Benito', 'Carmen']

  La segunda lista es: ['Benito', 'Juan', 'Carmen']

  Palabras que aparecen en las dos listas: ['Carmen', 'Benito']

  Palabras que sólo aparecen en la primera lista: ['Alberto']

  Palabras que sólo aparecen en la segunda lista: ['Juan']

  Todas las palabras: ['Carmen', 'Benito', 'Alberto', 'Juan']