



Programación  
Licenciatura en Agroinformática  
Licenciatura en Bioinformática

# Tema 05: Estructuras de Datos

# Tipos de Datos Primitivos

- En general, se distinguen 4 tipos de datos primitivos:
  - Enteros (int).
  - Reales (float).
  - Lógicos (bool).
  - Caracteres (char).
- En Python, el tipo de dato “char” no existe, sino que se considera como dato primitivo a las cadenas de caracteres (strings).

# Estructuras de Datos



- Se pueden definir estructuras de datos que están compuestas por distintos datos primitivos.
- Estas estructuras de datos poseen operaciones propias, distintas a las de los datos primitivos.
- En Python, las estructuras de datos pueden clasificarse en:
  - Estructuras inmutables.
  - Estructuras mutables.

# Tuplas

- Son secuencias de elementos **ordenados** e **inmutables**.
- Se definen usando paréntesis y valores separados por comas.

```
tupla1 = ("elem1", "elem2", "elem3")
```

- En una misma tupla se pueden combinar distintos tipos de datos primitivos.
- Para acceder a los distintos datos en una tupla se usan índices indicados entre corchetes

# Ejemplos de Tuplas

```
tupla1 = ("elem1", "elem2", "elem3")
tupla2 = (1.0, "2") #Distintos tipos de datos
tupla3 = (17,) #Tupla con solo un elemento
tupla4 = () #Tupla vacía
tupla1[1] #Devuelve "elem2"
tupla3[0] #Devuelve 17
```

# Listas

- Son secuencias de elementos **ordenados** y **mutables**.
- Se definen usando corchetes y valores separados por comas.

```
lista1 = ["elem1", "elem2", "elem3"]
```

- Al igual que con las tuplas, una lista puede almacenar distintos tipos de datos primitivos.
- Los elementos individuales también se acceden con corchetes, como en las tuplas

# Ejemplos de Listas

```
lista1 = ["elem1", "elem2", "elem3"]  
lista2 = [1.0, "2"] #Distintos tipos de datos  
lista3 = [17] #Lista con solo un elemento  
lista4 = [] #Lista vacía  
lista1[2] #Devuelve "elem3"  
lista2[0] #Devuelve 1.0
```

# Arreglos (Arrays)

- Son una estructura muy común en muchos lenguajes de programación.
- Son colecciones de datos de un mismo tipo.
- En Python no trae por defecto este tipo de dato, pero pueden usarse listas para implementarlos.



# Inmutable vs Mutable

- Los elementos en estructuras inmutables no pueden ser alterados.
- Las estructuras mutables permiten la asignación de nuevos valores a elementos individuales.
- Para modificar un elemento inmutable (como una tupla) es necesario crear una estructura nueva

# Immutable vs Mutable

```
lista = [1, 2, 3, 4, 5]
print(lista)
lista[2] = "valor nuevo"
print(lista)
tupla = (1, 2, 3, 4, 5)
print(tupla)
tupla[2] = "valor nuevo"
print(tupla)
```

```
[1, 2, 3, 4, 5]
[1, 2, 'valor nuevo', 4, 5]
(1, 2, 3, 4, 5)
Traceback (most recent call last):
  File "prueba.py", line 8, in <module>
    tupla[2] = "valor nuevo"
TypeError: 'tuple' object does not support item assignment
```

# Modificar un elemento de una tupla

- Para modificar una tupla es necesario crear una nueva.
- La tupla nueva se puede obtener como la suma de la tupla original en 2 rangos, antes y después del elemento a modificar, y una tupla nueva

```
# Modificando un elemento de una tupla
tupla = (1, 2, 3, 4, 5)
print(tupla)
tupla_nueva = tupla[0:2]+("valor nuevo",)+tupla[3:5]
print(tupla_nueva)
```

# Strings

- Un `string` se define como una cadena ordenada de caracteres.
- En lenguajes como C o Java, el tipo de datos `string` es considerado como una estructura de datos.
- En Python, si bien es un dato primitivo, los `strings` tienen comportamientos similares a una tupla.
- Los caracteres individuales pueden accederse mediante índices, pero son **inmutables**.

# Strings inmutables

```
string = "datos"
print(string)
print("Segunda letra: ", string[1])
print("Ultimas 3 letras: ", string[-3:len(string)])
# string[-3:len(string)] es lo mismo que string[-3:]
string[0] = "p"
```

```
datos
Segunda letra: a
Ultimas 3 letras: tos
Traceback (most recent call last):
  File "test.py", line 5, in <module>
    string[0] = "p"
TypeError: 'str' object does not support item assignment
```

# Recorriendo listas y tuplas

- Puede accederse a los elementos de una lista o tupla mediante índices.
- Como ocurre con los strings, pueden usarse en un bucle `for` para recorrer los elementos individuales.

```
lista = [1, 2, 3, 4, 18]
```

```
for i in lista:
```

```
    print (i)
```

```
tupla = ("a", "b", "a", "c", "a")
```

```
for j in tupla:
```

```
    print(j)
```

```
1
2
3
4
18
a
b
a
c
a
```

# Recorriendo listas y tuplas

- La función `len` devuelve el largo de una lista o tupla.
- Por lo tanto, también se podría usar para recorrer estas estructuras de datos.
- El primer índice en una lista o tupla es el 0!!

```
lista = [1, 2, 3, 4, 18]
```

```
for i in range(0, len(lista)):
```

```
    print ("El elemento", i, "de la lista es", lista[i])
```

```
El elemento 0 de la lista es 1
El elemento 1 de la lista es 2
El elemento 2 de la lista es 3
El elemento 3 de la lista es 4
El elemento 4 de la lista es 18
```

# Operaciones sobre listas



| Operación               | Resultado   |
|-------------------------|---|
| <code>L.count(x)</code> | Calcula la frecuencia de <code>x</code> en <code>L</code> .                       |
| <code>L.index(x)</code> | Devuelve el índice de la primera ocurrencia de <code>x</code> en <code>L</code> . |
| <code>if x in L</code>  | Devuelve <b>True</b> si el valor <code>x</code> se encuentra en <code>L</code> .  |

Estas operaciones no modifican  
el contenido de la lista



# Operaciones sobre listas



| Operación                    | Resultado   |
|------------------------------|---|
| <code>L.append (x)</code>    | Agrega el valor <code>x</code> al final de la lista <code>L</code> .  |
| <code>L.insert (x, i)</code> | Agrega el valor <code>x</code> en la posición <code>i</code> de la lista <code>L</code> . Todos los elementos después de <code>i</code> son desplazados a la derecha. |
| <code>L.remove (x)</code>    | Elimina la primera ocurrencia de <code>x</code> en <code>L</code> .   |
| <code>L.pop (i)</code>       | Elimina el valor que está en la posición <code>i</code> y desplaza los valores restantes a la izquierda. Si no se indica <code>i</code> , se elimina el último valor. |
| <code>L.reverse ()</code>    | Invierte los valores en <code>L</code> .  |
| <code>L.sort ()</code>       | Ordena los valores de menor a mayor (se puede incluir la opción “reverse = True” para ordenar de mayor a menor).  |
| <code>L + L2</code>          | Concatena las listas <code>L</code> y <code>L2</code> .   |
| <code>L*x</code>             | Concatena el contenido de <code>L</code> <code>x</code> veces   |

# Separando un string

- `split` es una función muy utilizada cuando se trabaja con `strings`.
- Esta función devuelve una lista de valores a partir de un `string` y un carácter separador.
- Los elementos resultantes también son del tipo `string`

```
string = "100,200,300,400,500,600"
```

```
lista = string.split(",")
```

```
print (lista) # Se obtiene ["100", "200", ...
```

# Estructuras anidadas

- Los elementos dentro de una lista o tupla pueden ser de cualquier tipo, incluidas otras listas y tuplas.
- Las listas anidadas se utilizan usualmente para representar matrices y tablas.
- Para acceder a un valor individual se deben indicar los índices entre corchetes.

# Estructuras anidadas

| Apellido       | Año  | Categoría |
|----------------|------|-----------|
| Houssay        | 1947 | Medicina  |
| Leloir         | 1970 | Química   |
| Milstein       | 1984 | Medicina  |
| Perez Esquivel | 1980 | Paz       |
| Saavedra Lamas | 1936 | Paz       |

```
tabla = [["Houssay", 1947, "Medicina"],  
         ["Leloir", 1970, "Química"],  
         ["Milstein", 1984, "Medicina"],  
         ["Perez Esquivel", 1980, "Paz"],  
         ["Saavedra Lamas", 1936, "Paz"]]
```

```
print (tabla[0][1])  
print (tabla[2][2])  
print (tabla[1][-2])  
print(len(tabla))
```

# Ejemplo 1

- Encontrar el número de diferencias entre dos palabras del mismo largo.

Entrada:

pal1, pal2

Salida:

diferencias

Pseudocódigo:

Pedir pal1 y pal2

diferencias = 0

if largo(pal1) == largo(pal2):

    for i in rango(0, largo("pal1")):

        if pal1[i] != pal2[i]:

            diferencias += 1

    Devolver diferencias

else:

    Devolver mensaje de error.

# Ejemplo 2

- Dada la siguiente tabla, encontrar quién tiene más discos solistas y quién fue el último en nacer

| Nombre | Nacimiento | Discos solistas |
|--------|------------|-----------------|
| John   | 1940       | 12              |
| Paul   | 1942       | 18              |
| George | 1943       | 12              |
| Ringo  | 1940       | 21              |

Entrada:  
tabla

Salida:  
menor nacimiento; mayor discos

# Ejemplo 2

## Pseudocódigo:

```
Cargar tabla # Sin el encabezado
nacimiento = 0
nombre_nacimiento
discos = 0
nombre_discos
# Nacimiento indice 1
for i in range (0, len(tabla)):
    if tabla[i][1] > nacimiento:
        nombre_nacimiento = tabla[i][0]
        nacimiento = tabla[i][1]
Devolver nombre_nacimiento, nacimiento
# Discos indice 2
for i in tabla:
    if tabla[i][2] > discos:
        nombre_discos = tabla[i][0]
        discos = tabla[i][2]
Devolver nombre_discos, discos
```

# Código Ejemplo 2

```
tabla = [ ["John", 1940, 12], ["Paul", 1942, 18],  
          ["George", 1943, 12], ["Ringo", 1940, 21]]  
nacimiento = 0  
nombre_nacimiento = ""  
discos = 0  
nombre_discos = ""  
# Nacimiento indice 1  
for i in range (0, len(tabla)):  
    if tabla[i][1] > nacimiento:  
        nombre_nacimiento = tabla[i][0]  
        nacimiento = tabla[i][1]  
print("El mas joven de los 4 es:", nombre_nacimiento)  
  
# Discos indice 2  
for i in range (0, len(tabla)):  
    if tabla[i][2] > discos:  
        nombre_discos = tabla[i][0]  
        discos = tabla[i][2]  
print("El que mas discos solistas tiene es:", nombre_discos)
```



# Ejemplo 3

- A continuación se listan los casos de COVID-19 por millón de habitantes en distintos países de la región. Encuentre el mes y el país que más casos hubo.

|        | Argentina | Bolivia  | Brazil   | Chile     | Paraguay | Uruguay  |
|--------|-----------|----------|----------|-----------|----------|----------|
| Mayo   | 3,497.22  | 321.87   | 2,663.24 | 7,403.10  | 179.89   | 8,145.90 |
| Junio  | 3,016.98  | 1,478.21 | 6,247.39 | 14,761.68 | 726.60   | 9,296.43 |
| Julio  | 4,265.65  | 9,171.10 | 6,886.15 | 12,868.75 | 7,693.69 | 3,553.14 |
| Agosto | 2,604.38  | 5,310.83 | 2,780.59 | 13,690.73 | 1,201.25 | 2,730.96 |

# Ejemplo 3

## Pseudocódigo:

Cargar `tabla` # Números sin países ni meses

`maximo` = 0

`pais_maximo`

`mes_maximo`

`países` = [Argentina, Bolivia, Brasil, Chile, Paraguay, Uruguay]

`meses` = [Mayo, Junio, Julio, Agosto]

for `i` in range (0, len(`tabla`)):

    for `j` in range (0, len(`tabla[i]`)):

        if `tabla[i][j]` > `maximo`:

`maximo` = `tabla[i][j]`

`mes_maximo` = `j`

`pais_maximo` = `i`

Devolver `meses[i]`, `países[j]`

# Código Ejemplo 3

```
casos = [[3497.22, 321.87, 2663.24, 7403.10, 179.89, 8145.90],
         [3016.98, 1478.21, 6247.39, 14761.68, 726.60, 9296.43],
         [4265.65, 9171.10, 6886.15, 12868.75, 7693.69, 3553.14],
         [2604.38, 5310.83, 2780.59, 13690.73, 1201.25, 2730.96]]
países = ["Argentina", "Bolivia", "Brazil", "Chile", "Paraguay",
          "Uruguay"]
meses = ["Mayo", "Junio", "Julio", "Agosto"]
maximo = 0
pais_maximo = ""
mes_maximo = ""

for i in range (0, len(casos)):
    for j in range (0, len(casos[i])):
        if casos[i][j] > maximo:
            maximo = casos[i][j]
            mes_maximo = meses[j]
            pais_maximo = países[i]
print("La mayor cantidad de casos se registraron en", pais_maximo,
      "en el mes de", mes_maximo)
```