Estructuras de Datos

...

Programación
2024

Tipos de Datos Primitivos

- En general, se distinguen 4 tipos de datos primitivos:
  - **int** (enteros).
  - **float** (reales).
  - **bool** (lógicos).
  - **char** (caracteres).
- En Python, el tipo de dato “char” no existe, sino que se considera como dato primitivo a los ***strings*** (cadenas de caracteres).

Tipos de Datos Abstractos

- Se pueden definir tipos de datos que estén compuestos por conjuntos de (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** (no modificables).
    - Estructuras **mutables** (modificables).

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 = [] # Tupla vacía

lista1[1] # Devuelve “elem2”
lista3[0] # Devuelve 17
```

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

Inmutable 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
```

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 no pueden ser modificados.

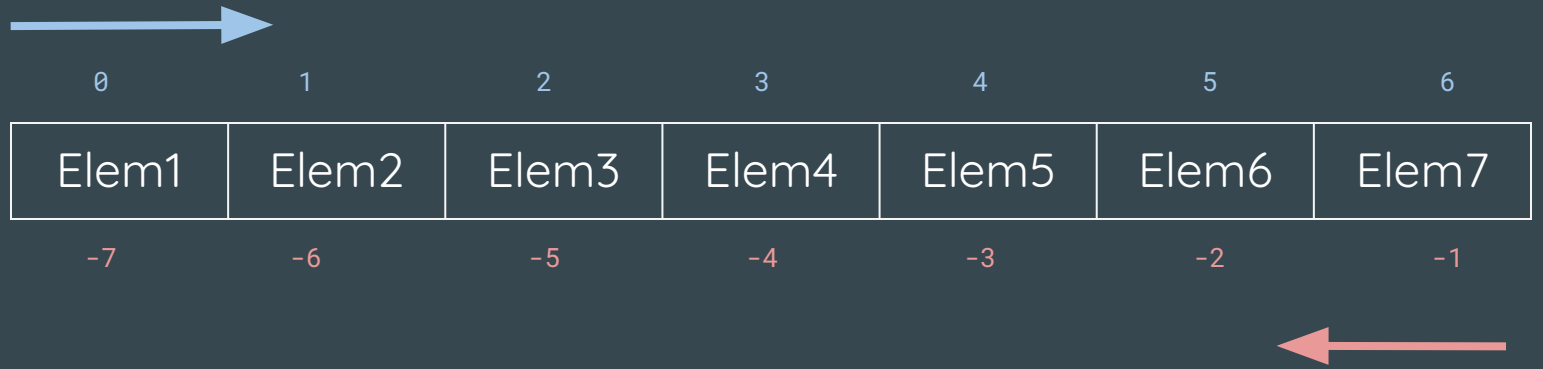
Strings: estructuras inmutables

```
string = “datos”
print(string)
print(“Segunda letra: “, string[1])
print(“Ultimas 3 letras: “, string[2:5])
string[0] = “p”
```

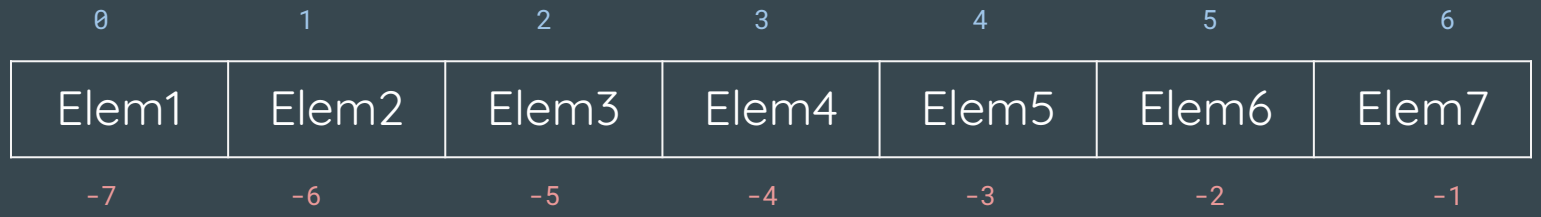
```
datos
Segunda letra:  a
Ultimas 3 letras:  tos
Traceback (most recent call last):
  File "test.py", line 5, in <module>
    string[0] = "p"
TypeError: 'str' object does not support item assignment
```

Recorriendo listas, tuplas y *strings*

- Puede accederse a los elementos de una lista o tupla mediante índices.
- Los índices siempre arrancan del número 0!!
- También es posible contar desde atrás hacia adelante, usando indices negativos



Recorriendo listas, tuplas y *strings*



- Es posible obtener un elemento de la lista o tupla usando su índice y corchetes

```
L[3] # devuelve “Elem4”
```

```
L[-3] # devuelve “Elem5”
```

- También pueden indicarse rangos usando los dos puntos (“:”). El límite superior indicado **NO** se incluye.

```
L[2:4] # devuelve “Elem3”, “Elem4”
```

```
L[:3] # devuelve “Elem1”, “Elem2”, “Elem3”
```

Recorriendo listas, tuplas y *strings*

- 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)
```

```
1
2
3
4
18
```

```
tupla = ("a", "b", "a", "c", "a")
for j in tupla:
    print(j)
```

```
a
b
a
c
a
```

Recorriendo listas, tuplas y *strings*

- 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 tuplas y listas

Operación	Resultado
if **x** in **L**	Devuelve **True** si el valor **x** se encuentra en **L**.
if **x** not in **L**	Devuelve **True** si el valor **x** NO se encuentra en **L**.
L1 + **L2**	Concatena los contenidos de **L1** y **L2**.

Estas operaciones no modifican el contenido de la lista o tupla

Operaciones sobre listas

Operación	Resultado
L.append(x)	Agrega x al final de L.
L.pop(i)	Elimina el elemento de la posición i en L. Si i no tiene valor, se elimina el último elemento.
L.reverse()	Se invierte el contenido de L.
L.sort()	Ordena los valores de menor a mayor (la opción “reverse = True” permite ordenar de mayor a menor).

Modificar un elemento de una tupla

- Como no se puede modificar una tupla, para realizar un cambio es necesario crear **una tupla 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)
```

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 dentro de la lista también son del tipo *string*

```
string = "100,200,300"
lista = string.split(",")
print (lista) # Se obtiene [“100”, “200”, “300”]
```

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: ejemplo

Investigador	Año	Categoria
Saavedra Lamas	1936	Paz
Houssay	1947	Medicina
Leloir	1970	Quimica
Perez Esquivel	1980	Paz
Milstein	1984	Medicina

Estructuras anidadas: ejemplo

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

print (tabla[0][1])  # Devuelve 1936
print (tabla[2][2])  # Devuelve “Quimica”
print (tabla[1][-3]) # Devuelve “Houssay”
print(len(tabla))    # Devuelve 5
```

Ejemplo 1

- Encuentre que clubes compartieron Maradona y Messi:
  - Maradona: Argentinos Jr, Boca, Barcelona, Napoli, Sevilla, Newell’s
  - Messi: Barcelona, PSG, Inter Miami

Ejemplo 1: pseudocódigo

1. Cargar lista de clubes maradona, messi.
2. clubes_comun = [ ]
3. Por cada club en maradona,
    a. Por cada club en messi
        i. si club_maradona == club_messi
            1. Agregar a clubes_comun
4. Devolver clubes_comun

Ejemplo 1: código

```
maradona = ["Argentinos Jr", "Boca", "Barcelona", "Napoli", "Sevilla",
"Newell's"]
messi = ["Barcelona", "PSG", "Inter Miami"]
clubes_comun = []
for i in maradona:
  for j in messi:
    if i == j:
      clubes_comun.append(i)
print(clubes_comun)

# Mejor solución
for i in maradona:
  for j in messi:
    if i == j and i not in clubes_comin:
      clubes_comun.append(i)
```

Ejemplo 2

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

Nombre	Año nacimiento	Discos solistas
John	1940	7
Paul	1942	18
George	1943	12
Ringo	1940	21

Entrada: tabla; Salida: mayor nacimiento; menor discos

Ejemplo 2: pseudocódigo

```
Cargar tabla, sin encabezados
nacimiento_mayor = -∞; nombre_nacimiento = “”;
para cada nacimiento (indice 1):
    si nacimiento > nacimiento_mayor:
        nacimiento_mayor = nacimiento
        nombre_nacimiento = nombre
discos_menor = +∞; nombre_discos = “”
para cada disco (indice 2):
    si discos < discos_menor:
        discos_menor = discos
        nombre_discos = nombre
Devolver nacimiento_mayor, nombre_nacimiento, discos_menor y nombre_discos
```

Ejemplo 2: código

```
tabla = [["John", 1940, 12], ["Paul", 1942, 18],
         [“George", 1943, 12], ["Ringo", 1940, 21]]

mayor_nacimiento = -float("inf")
nombre_nacimiento = ""
menor_discos = float("inf")
nombre_discos = ""

# Nacimiento indice 1
for i in range (0, len(tabla)):
    if tabla[i][1] > mayor_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] < menor_discos:
      nombre_discos = tabla[i][0]
      discos = tabla[i][2]
print("El que menos 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 en 2022. Encuentre el mes y el país que más casos hubo.

	Argentina	Bolivia	Brasil	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, sin encabezados
Cargar lista meses, lista paises
maximo = 0, pais = “”, mes = “”
Recorrer tabla
por cada columna en tabla: # o “por cada pais”
    por cada fila en tabla: # o “por cada mes”
        si tabla[fila][columna] > maximo:
            maximo = tabla[fila][columna]
            pais = paises[columna]
            mes = meses[fila]
Devolver maximo, pais, mes
```

Ejemplo 3: código

```
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]]
paises =  ["Argentina", "Bolivia", "Brazil", "Chile", "Paraguay", "Uruguay"]
meses = ["Mayo", "Junio", "Julio", "Agosto"]
maximo = 0
pais_maximo = ""
mes_maximo = ""

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