Fundamentos de Algoritmos

Profs. Agustín Gravano y Hernán Czemerinski

Primer Semestre de 2020

Clase 3: Procesamiento de listas y archivos

Tipo de datos **Lista**

Hasta ahora vimos tipos básicos: `bool`, `int`, `float`, `string`.

Las listas son secuencias de valores. Son un tipo compuesto.

```
[False, False, True, False]
[45, 657, 11, 0, 45, -303, 0, -11, 45]
[3.14159, 0.0]
["Marte", "Venus", "Júpiter"]
```

Listas: Operaciones básicas

```
# Creamos una lista con varios elementos.
mundiales = ["Corea-Japón", "Alemania",
             "Sudáfrica", "Brsail"]

# Leemos una posición de la lista.
print(mundiales[3])

# Sobreescribimos una posición.
mundiales[3] = "Brasil"

# Agregamos un elemento al final.
mundiales.append("Rusia")

# Imprimimos toda la lista.
print(mundiales)
```

Listas: Operaciones básicas

Operaciones de listas:

```
a = []          # Crea una lista vacía.
a = [7, 1, 2]   # Crea una lista no vacía.
len(a)          # Devuelve la longitud.
a.append(10)    # Agrega un valor al final.
print(a[i])     # Lee el i-ésimo valor.
a[i] = 8        # Escribe en i-ésima posición.
x in a          # Consulta pertenencia.
```

Listas: Operaciones básicas. Ejercicios

1. Completar el código, para construir una lista con los cuadrados de los primeros 10 enteros positivos: 1, 4, 9, 16, ...

```
cuadrados = ______________
x = 1
while x <= 10:
    cuadrados.______________
    x = x + 1
```

2. Completar el código, para imprimir los elementos de la lista *cuadrados* en una sola línea, separados con ' '.

```
i = 0
while i < ______________:
    print(______________, end=' ')
    i = i + 1
```

Listas: Iteradores

La instrucción for nos permite iterar los elementos de una lista:

```
1 for m in mundiales:
2     print(m)
```

```
1 for x in cuadrados:
2     print(x, end=' ')
```

También se puede iterar sobre **strings**:

```
1 for c in "Algoritmos":
2     print("*** "+ c +" ***")
```

Se puede iterar sobre varios tipos de datos: conjuntos, archivos, diccionarios, etc. Más adelante volveremos a estas cosas.

Listas: Indices y Sublistas *(slices)*

```
mundiales = ["Corea-Japón", "Alemania",
             "Sudáfrica", "Brasil", "Rusia"]
len(mundiales) # devuelve 5
mundiales[4]   # último elemento
mundiales[5]   # error: índice fuera de rango
mundiales[-1]  # último elemento!
```

Python permite obtener fácilmente sublistas (o *slices*):

```
mundiales[i:j] # sublista desde i hasta j-1
mundiales[1:4] # del segundo al anteúltimo
```

El operador `[i:j]` es muy versátil. `i` y `j` pueden ser negativos, e incluso omitirse:

```
mundiales[-2:] # últimos dos elementos
```

Operaciones de listas y strings

Si tenemos una lista de strings y los queremos unir en un solo string, separados por coma (p.ej.), usamos `join`.

```
colores = ["verde", "azul", "rojo"]
", ".join(colores)
# devuelve 'verde, azul, rojo'
```

La operación inversa es `split`.

```
colores = "verde, azul, rojo"
colores.split(", ")
# devuelve ['verde', 'azul', 'rojo']
```

Otras operaciones útiles de `strings`:

```
x = "programación"
"rama" in x     # True: "rama" es substring
"wxyz" in x     # False
x[4:8]          # 'rama'
```

Listas y strings. Ejercicio

Evaluar las siguientes expresiones, primero a mano, después revisar en la consola iPython.

```
1  x = "dificilisimo"
2  x.split('i')
3  x.split('i')[2:]
4  'i'.join(x.split('i')[2:])
5  'fa' + 'i'.join(x.split('i')[2:])
```

¿De qué tipo es cada expresión? ¿Qué valores tienen?

Lectura de archivos de texto

Con `open(filename)` abrimos un archivo en modo lectura.

```
f = open("provincias.csv")
```

Nota: En Windows, puede ser necesario un argumento adicional para reconocer la codificación del texto: `f = open("provincias.csv", encoding="utf-8")`

Podemos leer todo el contenido del archivo en una variable:

```
texto = f.read()
print(texto, end="")
```

También podemos iterar el archivo, leyendo línea a línea:

```
for linea in f:
  print(linea, end="")
```

Escritura de archivos de texto

Con `open(filename, "w")` abrimos un archivo en modo escritura (`w` es por *write*).

```
f = open("mundiales.txt", "w")
```

Escribimos con la instrucción `f.write("...")`.

```
for m in mundiales:
  f.write(m + "\n")    # \n indica fin de línea
```

Al terminar, es buena costumbre cerrar el archivo:

```
f.close()
```

Listas y archivos. Ejercicio

Usando los datos contenidos en el archivo `provincias.csv`,[1] completar el programa `provincias.py` para que:

1. Calcule la población total de Argentina.
2. Liste las provincias que tienen al menos dos millones de habitantes, de esta manera: "Buenos Aires, CABA, ...".

[1] Datos tomados de Wikipedia: `https://tinyurl.com/provarg`.

Listas: Rangos de enteros

La instrucción `range(i, j)` construye el rango de enteros `[i, i+1, ..., j-1]`.

Ejemplo:

```
for x in range(1, 4):
  print(x)                    # imprime 1, 2 y 3
```

Detalle técnico: `range(i, j)` es un valor iterable.
Si queremos generar *explícitamente* la lista de enteros, tenemos que convertir `range(i, j)` a una lista:

```
range(1, 4)           # devuelve range(1, 4)  :\
list(range(1, 4))     # devuelve [1, 2, 3]
```

Listas por comprensión

Empecemos con la lista de enteros del 1 al 10:

```
ls = list(range(1, 11))
ls  # devuelve [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Otra forma (rebuscada) de decir lo mismo que `ls`:

```
[x for x in ls]
```

Elevamos al cuadrado cada elemento:

```
[x*x for x in ls]
# [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

También podemos filtrar los elementos que nos interesen:

```
[x*x for x in ls if x%2==0]
# [4, 16, 36, 64, 100]
```

Listas por comprensión

En general:

```
[ EXPRESIÓN for x in LISTA if CONDICIÓN ]
```

Para cada elemento `x` de la `LISTA` que cumpla con la `CONDICIÓN` (expresión booleana), se evalúa la `EXPRESIÓN` (de cualquier tipo), construyendo así una nueva lista.

```
r = range(0,5)                          # r: [0,1,2,3,4]

[1 for x in r]                            # [1,1,1,1,1]

[x for x in r if x%2!=0]                        # [1,3]

[x%2 for x in r]                          # [0,1,0,1,0]

[str(x) for x in r]    # ['0','1','2','3','4']
```

Listas por comprensión. Ejercicio

Tenemos una función para calcular la raíz cuadrada entera de un número (ya definida en la guía de ejercicios):

```
1  def raiz_cuadrada(n):
2    r = 0
3    while r*r <= n:
4      r = r + 1
5    return r-1
```

Se pide armar una expresión con listas por comprensión, que dada una lista de números `a`, devuelva la lista de raíces cuadradas enteras de los números no negativos de `a`.

Ejemplo: Para `a = [4, -1, 0, 100, 5, -5, -1, 10]`, debe devolver `[2, 0, 10, 2, 3]`.

Asignación y pasaje de parámetros por copia vs. por referencia

(Para leer después.)

Asignación de tipos básicos por copia

Los tipos básicos (`bool`, `int`, `float`, `string`) se asignan por copia.

```
1  a = 99
2  b = a
3  b = b + 1
4  print(a, b)          # Imprime "99 100"
```

En la línea 2 se asigna a la variable `b` una copia del valor de `a`.

Entonces, cualquier modificación a `b` no podrá afectar a `a`, porque refieren a **distintos espacios en memoria**.

Asignación de listas por referencia

Copiar una lista entera podría ser caro: consumir mucho tiempo y espacio. Por eso, las listas se asignan por referencia.

```
1  a = [8, 9]
2  b = a
3  b.append(10)
4  print(a, b)        # Imprime [8,9,10] [8,9,10]
```

Ahora, se asigna a la variable b una referencia del valor de a.

Entonces, cualquier modificación a b también afectará a a, porque ambas refieren a un **mismo espacio en memoria**.

Para asignar una copia, debemos hacerlo explícitamente:

```
1  a = [8, 9]
2  b = list(a)        # list(a) es una copia de a
3  b.append(10)
4  print(a, b)        # Imprime [8,9] [8,9,10]
```

Pasaje de parámetros por copia

Análogamente, si una función tiene parámetros de tipos básicos (`bool`, `int`, `float`, `string`), los valores se pasan por copia.

```
1  def f(x):
2      print(x)     # imprime 123
3      x = 999
4      print(x)     # imprime 999
5
6  q = 123
7  f(q)
8  print(q)         # imprime 123
```

En la línea 7 a la función `f` se le pasa una copia del valor de `q`.

Entonces, cualquier modificación a `x` no podrá afectar a `q`, porque refieren a **distintos espacios en memoria**.

Pasaje de parámetros por referencia

Si una función tiene una lista como parámetro, copiarla entera podría ser caro: consumir mucho tiempo y espacio.

Por eso, las listas se pasan por referencia.

```
1  def f(x):
2      print(x)     # imprime [1,2,3]
3      x[1] = 999
4      print(x)     # imprime [1,999,3]
5
6  q = [1, 2, 3]
7  f(q)
8  print(q)         # imprime [1,999,3]
```

Ahora, a la función `f` se le pasa una referencia a la lista `q`.

Entonces, cualquier modificación a `x` también afectará a `q`, porque ambas refieren a un **mismo espacio en memoria**.

Pasaje de parámetros. Ejercicio

Determinar qué imprime por pantalla este código (primero pensarlo sin la ayuda de la computadora):

```
1  def despedir(x):
2      x.append('chau')
3      print(len(x))
4
5  a = ['hola', 'mundo']
6  despedir(a)
7  print(a)
8  despedir(list(a))
9  print(a)
```

Repaso de la clase de hoy

- Tipos de datos Lista. Operaciones, iteradores, sublistas.
- Lectura y escritura de archivos de texto.
- Listas por comprensión.
- Asignación y pasaje de parámetros por copia vs. por referencia.

Con lo visto, ya pueden resolver la sección 4 de la guía de ejercicios.