Clase 11 – Listas II

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

08-10-2018

Resumen de la clase

- Las listas son variables en Python que guardan una secuencia de valores. Estos valores se denominan *elementos*.
- Por ejemplo:

```
lista1 = [1,2,3]
```

- Podemos crear listas vacías también, de la siguiente forma:

```
lista2 = []
```

- Para poder obtener algún elemento de una posición determinada un una lista, se hace de la siguiente forma:

```
elemento_en_posición_x = variable_de_tipo_lista[x]
```

Donde elemento_en_posición_x es la variable que almacenará el elemento que está en la posición x de la lista que está almacenada en la variable_de_tipo_lista.

Resumen de la clase

- **Slice:** Es la acción de obtener una nueva lista desde otra lista. Esta nueva lista tendrá ciertos elementos que dependerá de cómo hicimos el *slice.* Si `l` es una lista, se hace de la siguiente manera:

```
l[i:j]
```

Donde obtenemos los elementos desde la posición `i` hasta `j-1`.

- **Edición de elementos:** Podemos editar elementos de una lista. Si `l` es una lista y k es una posición de la lista:

```
l[k]=nuevo elemento
```

Donde la información que estaba en la posición k se sobreescribió con el nuevo elemento. Recordemos que este nuevo elemento puede ser de cualquier tipo.

Resumen de la clase

- **Agregar elementos:** Se pueden agregar nuevos elementos a una lista. Si `l` es una lista, se hace de la siguiente manera:

```
l.append(x)
```

Donde se agrega el elemento x a la lista.

- **Cantidad de elementos de una lista:** Se puede obtener la cantidad de elementos de una lista. Si `l` es una lista, se hace de la siguiente manera:

```
len(l)
```

- **`for` sobre una lista:** Podemos "recorrer" todos los elementos de una lista mediante un `for`. Si `l` es una lista, se hace de la siguiente manera:

```
for i in s1:
    i #i representa a cada uno
      #de los elementos de l.
```

Ejercicio en Clase

- Se tiene una lista en Python de nombre “nombres”, **cuyo largo no sabemos**, que tiene nombres y apellidos. Específicamente, los nombres dentro de la lista se ordenan de la siguiente manera:

```
nombres = [ … , nombre1_persona_i, nombre2_persona_i, apellido_pat_persona_i, apellido_mat_persona_i,
nombre1_persona_(i+1), nombre2_persona_(i+1), apellido_pat_persona_(i+1), apellido_mat_persona_(i+1), …]
```

- Imprima en consola el nombre completo de una persona *i*, en el siguiente formato:

```
nombre1_persona_i nombre2_persona_i apellido_pat_persona_i apellido_mat_persona_i
```

- Cree una nueva lista de nombre “nombres_completos”, que tenga los nombres completos de las personas (en el formato anterior), y en el mismo orden.

Contenidos

1. `split()`
2. `join()`
3. `insert()`
4. `pop()`
5. `in`
6. `+`
7. Ejercicios

Ejercicio 2

Para poder mejorar el código morse, usted debe implementar 3 funciones en Python:

1. Una función que pueda transformar un número binario a un decimal.
2. Una función que pueda transformar un número binario a un texto. Para esto, puede ocupar la tabla ASCII (que puede ver acá). El programa asume que cada letra en binario de la palabra que desea decodificar tiene un largo de 5 dígitos.
3. Una función que transforme un código morse a palabras. El código morse puede incorporar ahora "," para separar palabras.

65	41	101	&#65;	A
66	42	102	&#66;	B
67	43	103	&#67;	C
68	44	104	&#68;	D
69	45	105	&#69;	E
70	46	106	&#70;	F
71	47	107	&#71;	G
72	48	110	&#72;	H
73	49	111	&#73;	I
74	4A	112	&#74;	J
75	4B	113	&#75;	K
76	4C	114	&#76;	L
77	4D	115	&#77;	M
78	4E	116	&#78;	N
79	4F	117	&#79;	O
80	50	120	&#80;	P
81	51	121	&#81;	Q
82	52	122	&#82;	R
83	53	123	&#83;	S
84	54	124	&#84;	T
85	55	125	&#85;	U
86	56	126	&#86;	V
87	57	127	&#87;	W
88	58	130	&#88;	X
89	59	131	&#89;	Y
90	5A	132	&#90;	Z

- En el último ejercicio que vimos, teníamos el siguiente *string:*

"--..---.-.-..---.--..-----.-.,-..---.....-----.-...-.-,.--.--....-.-.----..--..,"

- Parte de la solución comprendía buscar las “,” e ir separando el string ¿Habrá alguna forma de hacer esto de una forma más eficiente?

split()

- Si tenemos un *string* `s` y un *string* `x`:

s.split(x)

Esta operación separa al *string* s según el *string* x. Es decir, busca la o las ocurrencia(s) de `x` en `s` y cada vez que encuentra una, "corta" a `s`. Estos elementos se almacenan en una lista que es el valor retornado por esta función. Veamos un ejemplo:

```
string_ejemplo = "felipe,lopez,rojas"
res = string_ejemplo.split(",")
print(res)
```

```
['felipe', 'lopez', 'rojas']
```

split()

- ¿Qué pasa si queremos explícitamente transformar un *string* a una lista? Si tenemos un *string* s

```
list(s)
```

El *string* se transforma en una lista cuyos elementos son los caracteres del *string* original. Veamos un ejemplo:

```
string_ejemplo = "felipe,lopez,rojas"
res = list(string_ejemplo)
print(res)
```

```
['f', 'e', 'l', 'i', 'p', 'e', ',', 'l', 'o', 'p',
'e', 'z', ',', 'r', 'o', 'j', 'a', 's']
```

Ejercicio 2

Para poder mejorar el código morse, usted debe implementar 3 funciones en Python:

1. Una función que pueda transformar un número binario a un decimal.
2. Una función que pueda transformar un número binario a un texto. Para esto, puede ocupar la tabla ASCII (que puede ver acá). El programa asume que cada letra en binario de la palabra que desea decodificar tiene un largo de 5 dígitos.
3. Una función que transforme un código morse a palabras. El código morse puede incorporar ahora "," para separar palabras.

65	41	101	A	A
66	42	102	B	B
67	43	103	C	C
68	44	104	D	D
69	45	105	E	E
70	46	106	F	F
71	47	107	G	G
72	48	110	H	H
73	49	111	I	I
74	4A	112	J	J
75	4B	113	K	K
76	4C	114	L	L
77	4D	115	M	M
78	4E	116	N	N
79	4F	117	O	O
80	50	120	P	P
81	51	121	Q	Q
82	52	122	R	R
83	53	123	S	S
84	54	124	T	T
85	55	125	U	U
86	56	126	V	V
87	57	127	W	W
88	58	130	X	X
89	59	131	Y	Y
90	5A	132	Z	Z

split()

¿Cómo podríamos aplicar split() en el ejercicio anterior?

```
def morse_modificado(s):
  sAux = s.replace(".","1").replace("-","0")
  comma_index = sAux.find(",")

  while comma_index != -1:
    bin_word = sAux[0:comma_index]
    print(decod_bin(bin_word))

    sAux = sAux[comma_index+1:len(sAux)]
    comma_index = sAux.find(",")
```

split()

```
def morse_modificado2(s):
  print(s)
  sAux = s.replace(".","1").replace("-","0")
  print(sAux)
  lista_sep_comas = sAux.split(",")
  print(lista_sep_comas)

  for bin_word in lista_sep_comas:
    print(decod_bin(bin_word))
```

join()

- Así como existe la función `split()` para poder transformar un *string* a una lista, se puede hacer la operación inversa.
- La función `join()` permite juntar los elementos de una lista en un solo *string*.
- También se puede especificar un *string* que separe los elementos de esta lista en el *string* resultante.

IMPORTANTE: La función `join()` solo se puede aplicar sobre una lista **de *strings.*** Si la lista contiene elementos que no sean *strings*, entonces Python arrojará error.

join()

- Si tenemos una lista de *strings* llamada `l`:

```
s = ''.join(l)
```

Donde s es una variable que almacenará el *string* resultante al juntar los elementos de la lista `l`. Veamos un ejemplo:

```
l = ["felipe","lópez","rojas"]
s = ''.join(l)
print(s)
```

```
felipelópezrojas
```

join()

- En el ejemplo anterior podemos ver que en el *string* resultante se juntaron todos los elementos de la lista. No obstante, no tenían separador alguno ¿Cómo podríamos unir los elementos de una lista con un *string* específico de por medio?

Este string vacío que se incluye puede ser cualquier *string*, y será el *string* que separe los elementos en el *string* resultante posterior al join().

```
s = ''.join(l)
```

join()

- Si tenemos una lista de *strings* llamada `l` y un *string* `x`:

```
s = x.join(l)
```

Donde s es una variable que almacenará el *string* resultante al juntar los elementos de la lista `l` por medio del string `x`. Veamos un ejemplo:

```
l = ["felipe","lópez","rojas"]
s = ';'.join(l)
print(s)
```

```
felipe;lópez;rojas
```

break

5 minutos

insert()

- Si `l` es una lista, y queremos agregar un elemento `x` a esta lista, podemos ejecutar:

```
l.append(x)
```

  Que agrega `x` al final de la lista

- ¿Y si queremos agregar `x` en una posición específica? Podemos ocupar la función `insert()`.

insert()

- Si `l` es una lista, `i` es un índice entre `0` y `len(l)` y `x` es un elemento que queremos agregar a esta lista, podemos ejecutar:

`l.insert(i,x)`

De esta forma, se agrega `x` en la posición `i`

- ¿Y si hay elementos después de la posición `i`? Estos se "desplazan" para dejar espacio para `x`.
- Veamos un ejemplo:

```
l = ["felipe","lópez","rojas"]
l.insert(1,"ignacio")
print(l)
```

```
['felipe', 'ignacio', 'lópez', 'rojas']
```

pop()

- Para poder sacar un elemento de una lista, podemos ocupar `pop()`.
- Por defecto, `pop()` remueve el último elemento de una lista y lo devuelve para poder ser usado. Si `l` es una lista:

```
ultimo_elemento = l.pop()
```

Donde `ultimo_elemento` es una variable que tiene al último elemento que tenía la lista `l` antes de usar `pop()`. Después de usar esta función, este último elemento es removido. Veamos un ejemplo:

```
l = ["felipe","lópez","rojas"]
ult_elem = l.pop()
print(ult_elem)
print(l)
```

```
rojas
['felipe', 'lópez']
```

pop()

- No solo podemos extraer y obtener el último elemento de una lista con la función `pop()`, sino que también puede ser un elemento en cualquier posición. Si `l` es una lista, e `i` es un índice entre `0` y `len(l)` :

```
elem_i = l.pop(i)
```

`elem_i` es una variable que contiene al elemento de la lista `l` en la posición `i` antes de hacer el `pop()`. Después de este la lista ya no contiene este elemento.

- Por ejemplo:

```
l = ["felipe","lópez","rojas"]
elem_1 = l.pop(1)
print(elem_1)
print(l)
```

```
['felipe', 'rojas']
```

in

- Para saber si un elemento está dentro de una lista, podemos ocupar el comando in. Si `l` es una lista, y `x` es un elemento que podría estar en `l`

```
x in l
```

Devolverá `True` si `x` efectivamente está dentro de la lista `l`, y `False` en caso contrario.

- Veamos un ejemplo:

```
l = ["pelota",1,True]
print(False in l)
print("pelota" in l)
```

```
False
True
```

+

- Para unir dos listas distintas, podemos ocupar el operador “+”.
- Si `l1` y `l2` son listas:

```
nueva_lista = l1 + l2
```

`nueva_lista` es una variable que contiene una lista cuyos elementos son los de `l1` y `l2`.

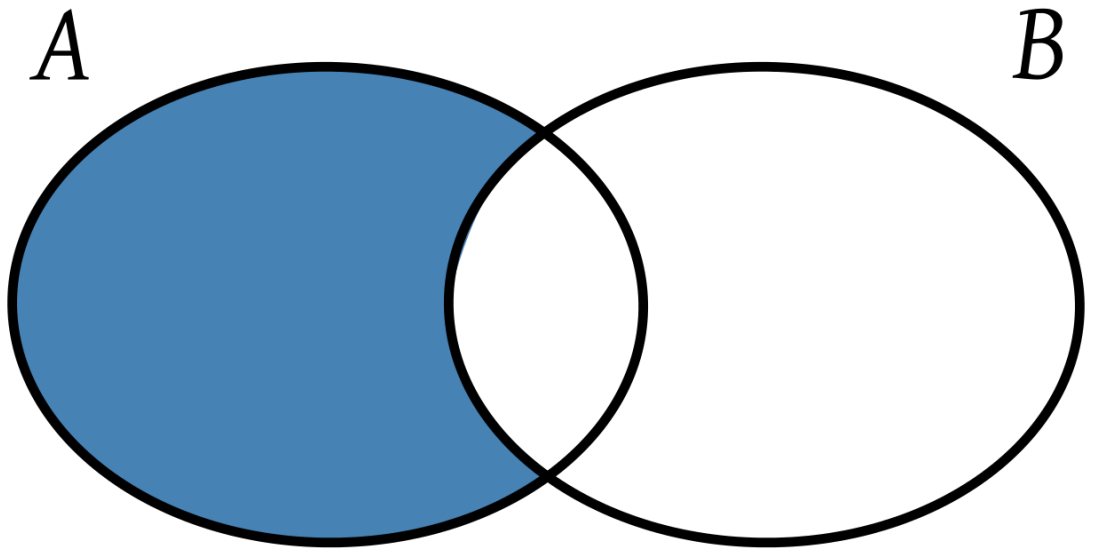
- Por ejemplo:

```
l1 = ["pelota",1,True]
l2 = ["felipe","lópez","rojas"]
nueva_lista = l1 + l2
print(nueva_lista)
print(l1 + l2)
```

```
['pelota', 1, True, 'felipe', 'lópez', 'rojas']
['pelota', 1, True, 'felipe', 'lópez', 'rojas']
```

Ejercicio 1

- Cree un programa que simule la operación A \ B entre dos conjuntos A y B cualquiera.
- Recuerde que esto equivale a:



Ejercicio 2

- Recibiste un *string* codificado que debes transformar en una palabra. La clave de esta codificación es que cada letra tiene su posición, y tu código debe ser capaz de armar la palabra ubicando cada letra en su posición correcta.
- En el string recibido uno recibe tuplas (posición,letra). De esta forma, uno puede armar la palabra final.
- No obstante, no se sabe a priori cómo distinguir las tuplas en el *string* codificado. Para poder saberlo:
  - El primer carácter del *string* codificado indica cuál carácter separa a las tuplas
  - El último carácter del *string* codificado indica cuál carácter separa a la posición y a la letra dentro de una tupla.
- Después de encontrar las posiciones y letras, debo armar mi *string* decodificado. Para esto, debo ubicar cada letra en su posición correspondiente
- Un ejemplo de esta codificación sería la siguiente:

STRING CODIFICADO: _2;L_5;A_4;T_3;O_1;E_0;P;

STRING DECODIFICADO: PELOTA

Solución Ejercicio 1

```
s = "_2;L_5;A_4;T_3;O_1;E_0;P;"
lista_s = list(s)
sep_tuplas = lista_s.pop(0)
sep_intra_tupla = lista_s.pop()
s = ''.join(lista_s)

s1 = s.split(sep_tuplas)
largo_palabra = len(s1)
print(s1)

lista_ordenada = []
for i in s1:
  lista_aux = i.split(";")
  lista_ordenada.append(int(lista_aux[0]))
  lista_ordenada.append(lista_aux[1])
print(lista_ordenada)

lista_final = ['']*largo_palabra
for i in range(0,len(lista_ordenada),2):
  lista_final.insert(lista_ordenada[i],lista_ordenada[i+1])

print(''.join(lista_final))
```

Resumen de la clase

- **`split()`:** Si tenemos un *string* `s` y un *string* `x`:

```
s.split(x)
```

Esta operación separa al *string* `s` según el *string* `x`. Es decir, busca la o las ocurrencia(s) de `x` en `s` y cada vez que encuentra una, "corta" a `s`.

- **`join()`:** Si tenemos una lista de *strings* llamada `l` y un *string* `x`:

```
s = x.join(l)
```

Donde `s` es una variable que almacenará el *string* resultante al juntar los elementos de la lista `l` por medio del string `x`.

Resumen de la clase

- **`insert()`:** Si `l` es una lista, `i` es un índice entre `0` y `len(l)` y `x` es un elemento que queremos agregar a esta lista, podemos ejecutar:

```
l.insert(i,x)
```

De esta forma, se agrega `x` en la posición `i`

¿Y si hay elementos después de la posición `i`? Estos se "desplazan" para dejar espacio para `x`.

- **`pop()`:** Si `l` es una lista, e `i` es un índice entre `0` y `len(l)`:

```
elem_i = l.pop(i)
```

`elem_i` es una variable que contiene al elemento de la lista `l` en la posición `i` antes de hacer el `pop()`. Después de este, la lista ya no contiene este elemento.

Resumen de la clase

- **`in:`** Si `l` es una lista, y `x` es un elemento que podría estar en `l`

```
x in l
```

Devolverá `True` si `x` efectivamente está dentro de la lista `l`, y `False` en caso contrario.

- **`+:`** Si `l1` y `l2` son listas:

```
nueva_lista = l1 + l2
```

`nueva_lista` es una lista cuyos elementos son los de `l1` y `l2`.

Bibliografía

- http://runest.ing.puc.cl/list.html
- A. B. Downey. Think Python: How to think like a computer scientist. Green Tea Press, 2013 -> Capítulo 10

Links

- https://repl.it/@FelipeLopez/IIC1103ListasII que contiene todos los ejemplos de la clase.

Clase 11 – Listas II

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

08-10-2018