FUNDAMENTOS DE COMPUTACIÓN
GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
Universidad de Cantabria

Práctica 6 (laboratorio)
29 de octubre del 2019

Secuencias de objetos en PYTHON

PYTHON tiene varias clases de objetos no escalares (compuestos):

- Cadenas/listas/tuplas
  - Una **str**(cadena) es una sucesión de caracteres encerrada entre comillas (simples o dobles).
  - Una **list**(lista) es una secuencia de objetos (números, cadenas, listas, ...). Los elementos de una lista deben estar encerrados entre corchetes y separados por comas.
  - Una **tuple**(tupla) es una secuencia de objetos (números, cadenas, listas, tuplas ...). Los elementos de una tupla deben estar encerrados entre paréntesis y separados por comas.

```
>>> a = [1, 3.1, 'VIP',  ["Pedro", 942100013], ["Juan", 942805678] ]
>>> type(a)
<class 'list'>
>>> b = ( True, [2, 3, 5, 7], 'NON SMOKING', (2**(1 /2), 2) )
>>> type(b)
<class 'tuple'>
>>> tuple(a)
(1, 3.1, 'VIP', ['Pedro', 942100013], ['Juan', 942805678])
>>> str(a)
"[1, 3.1, 'VIP', ['Pedro', 942100013], ['Juan', 942805678]]"
>>> list(b)
[True, [2, 3, 5, 7], 'NON SMOKING', (1.4142135623730951, 2)]
>>> c = "hola, mundo"
>>> type(c)
<class 'str'>
>>> list(c)
['h', 'o', 'l', 'a', ',', ' ', 'm', 'u', 'n', 'd', 'o']
>>> tuple(c)
('h', 'o', 'l', 'a', ',', ' ', 'm', 'u', 'n', 'd', 'o')
```

- La función **len()** y el operador **in**
  - La función **len()** devuelve la longitud de la cadena/lista/tupla, es decir el número de caracteres/elementos.
  - El operador de pertenencia **in** aplicado a un objeto de una cadena/lista/tupla devuelve el booleano False o True, si el objeto pertenece o no pertenece.

```
>>> a = [1, 3.1, 'VIP',  ["Pedro", 942100013], ["Juan", 942805678]]
>>> len(a)
5
>>> b = (True, [2, 3, 5, 7], 'NON SMOKING', (2**(1 /2), 2))
>>> len(b)
4
>>> len('abcdef')
6
>>> 'VIP' in a
True
>>>  False in b
False
>>> 0 in []:
False
```

- Indexar y Extraer

  Los corchetes permiten la indexación en una cadena/lista/tupla para obtener el valor en un determinado índice/posición. `L[i]`, donde i es entero positivo, aplicado a la cadena/lista/tupla `L` devuelve el carácter/elemento de la posición $i+1$ de `L`.

```
>>>  L = ['ax', 'by', 'cz']
>>> L[0]
'ax'
>>> L[1]
'by'
>>> 'L[2]'
'cz'
>>> len(L[0])
2
>>> L[0][1]
'x'
```

```
>>> L[-1]
'cz'
>>> L[-2]
'by'
>>> L[-3]
'ax'
>>> L[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

- Cortar/Trocear cadenas/listas/tuplas.

  - Podemos cortar la cadena/tupla/lista:`L`, usando `L[inicio:detener:paso]`.
  - Si se dan dos números, `L[inicio: detener]`, paso = 1, por defecto
  - También se puede omitir números y dejar solo dos puntos

```
>>> L = (2, 4, 8, 16, 32, 64, 128, 256)
>>> L[3:6]
(16, 32, 64)
>>> L[3:6:2]
(16, 64)
>>> L[:]                                  # Evalua L[0:len(L)]
(2, 4, 8, 16, 32, 64, 128, 256)
>>> L[:2]                                 # Evalua L[0:2]
(2, 4)
>>> L[2:]                                 # Evalua L[2:len(L)]
(8, 16, 32, 64, 128, 256)
>>> L[:-1]                                # Evalua L[0:-1+Len(L)]
(2, 4, 8, 16, 32, 64, 128)
```

- Operador concatenación y repetición.

  - Operador + (concatenación de cadena/lista/tupla): acepta dos cadenas/listas/tuplas como operandos y devuelve la cadena/lista/tupla que resulta de unir la segunda a la primera.
  - Operador ***** (repetición de cadena/lista/tupla): acepta una cadena/lista/tupla y un entero positivo, y devuelve la concatenación de la cadena/lista/tupla consigo misma tantas veces como indica el entero.

```
>>> "Hola," +" "+ 'Santander'
'Hola, Santander'
>>> [['a',1],['b',2]] + [['c',3]]
[['a', 1], ['b', 2], ['c', 3]]
>>> [['a',1],['b',2]] + ['c',3]
[['a', 1], ['b', 2], 'c', 3]
>>> ['a','b'] + []
['a', 'b']
```

```
>>> (0,1) + (10,11)
(0, 1, 10, 11)
>>> 'holaX' * 3
'holaXholaXholaX'
>>> ()*2
()
>>> [1,2]*3
[1, 2, 1, 2, 1, 2]
```

- Objetos mutables e inmutables

  PYTHON usa la minima memoria necesaria. Ciertos objetos son **inmutables**, es decir, no pueden modificar su valor. Los objetos númericos, `int`, `float` son inmutables. Si dos variables almacenan el mismo objeto, sus identificadores apuntan a la misma zona de la memoria.

Las cadenas y las tuplas (STR, TUPLE) también son **immutables**. Por eso, asignar a una posición indexada de la cadena o tupla resulta en un error; sin embargo, las listas (LIST) pueden ser modificadas, son objetos **mutables**:

```
>>> a = 'hola'
>>> a[0] = 'H'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> Telf = [["Pedro", 942100013], ["Juan", 942805678]]
>>> Telf[1][0] = "Alan"
>>> Telf
[['Pedro', 942100013], ['Alan', 942805678]]
>>> Telf[0][1] = 1000000
```

- Lista: objeto mutable

  Una lista almacena en memoria referencias a los objetos, y no los objetos.

```
>>> L1 = [2, 3, 5]
>>> a = 2
>>> L2 = [2, 3, 5]
>>> L3 = L2
```

  PYTHON reserva nueva memoria para la lista `L2` aunque exista otra lista de idéntico valor. Como el contenido de cada celda ha resultado ser un valor inmutable (un entero), se han compartido las referencias a los mismos.

```
>>> L1 is L2
False
>>> L1 == L2
True
>>> L1[0] is a
True
>>> L3 is L2
True
>>> L2[0] = 89;  (L1,L2,L3)
([2, 3, 5], [89, 3, 5], [89, 3, 5])
```

- **Ejercicios**

  1. ¿Qué resultados se obtendrán al evaluar las siguiente expresiones? Calcula primero a mano el valor resultante de cada expresión y comprueba, con la ayuda del ordenador, si tu resultado es correcto.
     - 'hola'[0]; "hola"[0][0]; 'hola'[-1]; 'hola'[-1][0]; 'a'*3; 'a'*3[2]; type('hola'[2]); type([]).
     - type('hola'); int(2.1); float(2) ; str(12); list('hola'); tuple(12); int('2017'); str([1,2,'hola'])
     - type(['silencio',1,[5.0,5]][0][2]); ['silencio',1,[5.0,5]][-2]; ['silencio',1,[5.0,5]][-2]; ('silencio',1,[5.0,5])[0][2].
     - '- -'*5; [1,a]+[1,a]; [1,a]*2; --5; 'a'*3+'/*'*5+2*'abc'+'+'; 2*'12'+'.'+'3'*3+'e-'+4*'76'
     - list([]); len([1,[]); len("hola")-1; len([]*10); len(2); len(["hola",[1,2]][0]); float([3]).
     - a = [2,3,5]; a += [7,11,13]; a
     - a = (2,3,5); a += (7,11,13); a
     - 'ol' in "hola, Santander"
  2. Identifica regularidades en las siguiente cadenas, y escribe expresiones que, partiendo de subcadenas más cortas y, utilizando los operadores de concatenación y repetición, produzcan las cadenas que se muestran.

```
a) '%%%%%./././<-><->'
b) '(@)(@)(@)======(@)(@)(@)======'
c) 'asdfasdfasdf=-=-=-=-=-=-=-??????asdfasdf'
d) '........******---******---........******---******---'
e) '. . . . . . . . . . . . . . . . . . . . . . . . . . .'
```

  3. Escribe un programa que pida un entero entre 1 y 12, y muestre por pantalla el mes correspondiente.

```
$ python mes.py
¿Dime un entero entre 1 y 12? 10
Es el mes de Octubre.
$ python mes.py
¿Dime un entero entre 1 y 12? 15
Entero entre  1 y 12. Usted ha escrito 15.
```

4. Escribe un programa que pregunte al usuario su nombre y escriba las iniciales (sin tener en cuenta nombres apellidos que tengan más de una palabra).

```
$ python iniciales.py
¿Cómo se llama Ud.? Ramón Menéndez Pidal
R. M. P.
```

AYUDA: En el siguiente programa se ilustra el método `split`:

frase.py

```
1 print('Escriba una frase: ')
2 frase = input()
3 palabras = frase.split()
4 print(len(palabras))
5 print(palabras)
```

```
$ python frase.py
Escriba una frase: escriba una frase
3
['escriba', 'una', 'frase']
```

¿Cuál es el tipo del resultado de evaluar la expresión `frase.split(' ')`?

El método `split` retorna una lista con todos elementos encontrados al dividir la cadena por un separador.

```
>>> "Hola, Santander, 29 de octubre, 2018".split(",")
['Hola', ' Santander', ' 29 de octubre', ' 2018']
```

5. Diseña un programa en Python que pida una fecha donde todos los datos sean numéricos y muestre por pantalla la fecha convertida. Por ejemplo:

```
>>> Introduce una fecha ( día, mes, año ):
29,10,2018
La fecha convertida es el 29 de Octubre del 2018.
```

6. Al contrario que en la mayor parte de Europa, no es costumbre en España que la mujer adopte el apellido de su marido. Tradicionalmente, al nacer un niño se le asignan el apellido de su padre y el de su madre, por este orden. De manera no oficial, la sucesión de apellidos se puede prolongar tanto como sea posible según la memoria familiar, añadiendo a los dos primeros el segundo del padre, el segundo de la madre, el tercero del padre, etc.

   ¿Cómo podemos calcular el antepasado que otorga el apellido número n de una persona? La representación binaria del número $n - 1$ es una manera de expresarlo. Nos gustaría un programa que diera el resultado mostrado a la izquierda, pero vamos, de momento, a conformarnos con uno que se comporte de manera similar al de la derecha.

```
$ python ascendiente.py
Número de apellido: 1
El padre.
$ python ascendiente.py
Número de apellido: 2
La madre.
$ python ascendiente.py
Número de apellido: 4
La madre de la madre.
$ python ascendiente.py
Número de apellido: 13
La madre de la madre del padre del padre.
```

```
$ python ascendiente.py
Número de apellido: 1
PADRE
$ python ascendiente.py
Número de apellido: 2
MADRE
$ python ascendiente.py
Número de apellido: 4
MADRE MADRE
$ python ascendiente.py
Número de apellido: 13
MADRE MADRE PADRE PADRE
```

Hagamos ahora lo contrario:

```
$ python apellido.py
Ascendiente: PPPPPM
2
$ python apellido.py
Ascendiente: MMMMMP
63
```

```
$ python apellido.py
Ascendiente: MPM
6
```

AYUDA:

- El método `replace`. La solución que propone el siguiente programa para hacer frente a los *robots rastreadores* de direcciones electrónicas es algo precaria, pero nos sirve para ilustrar el *método* `replace`, que permite hacer sustituciones a partir de una cadena de caracteres.

email.py

```
1 print('Escriba su dirección: ')
2 cadena = input()
3 nueva = cadena.replace('@',' (at) ')
4 nueva = nueva.replace('.',' (dot) ')
5 print(nueva)
```

```
$ python email.py
Escriba su dirección: mengano.zutanez@gmail.com
mengano (dot) zutanez (at) gmail (dot) com
```

De esta manera, convertimos una dirección de correo electrónico en un código (mínimamente) más seguro para colocar, por ejemplo, en una página web.

- Las funciones `bin`. y `int`.

```
>>> bin(5)
'0b101'
>>> bin(5)[2:]
'101'
```

```
>>> int('0101',2)
5
```