Clase_2_Variables

November 22, 2021

1 Curso de Python

1.1 por Carlos Feinstein

---

2 Variables más complejas

En python existen estrucuturas que engloban a las variables en estructuras muchos más complejas, llamadas contenedores.

Los principales son

##Tipos de datos:

- Listas
- Tuplas
- Diccionarios
- Sets

2.1 Listas

Es una colección de objetos, pueden ser de tipos diferentes. Tienen orden.

Las listas pueden mutar, su contenido es modificable

[1]:
```
L = ['azul','verde','rojo'] # las lista se definen con una colección de datos␣
↪entre corchetes
```

[2]:
```
type(L) # Imprime el tipo de variable que es L
```

[2]:
```
list
```

[3]:
```
L[1]
```

[3]:
```
'verde'
```

[4]:
```
L[0]
```

[4]:
```
'azul'
```

[5]:
```
L[-1] # Ultimo elemento
```

[5]: 'rojo'

[6]:
```
L[-3]
```

[6]: 'azul'

[7]:
```
L=L+['rojo','amarillo']
```

[8]:
```
print(L)
```

```
['azul', 'verde', 'rojo', 'rojo', 'amarillo']
```

[9]:
```
L[1:3] # L[Comienzo:final:paso] elementos de índice i cumplen con comienzo <= i␣
↪< final !! OJO, el final NO ESTÄ incluido !!
       # Esto es irritante no pasa nada igual en los demás lenguajes
```

[9]: ['verde', 'rojo']

[10]:
```
L[2:] # Los valores de contorno pueden obviarse
```

[10]: ['rojo', 'rojo', 'amarillo']

[11]:
```
L[-2:]
```

[11]: ['rojo', 'amarillo']

[12]:
```
L[::2] # L[start:stop:step] cada dos elementos
```

[12]: ['azul', 'rojo', 'amarillo']

[13]:
```
L[::-1]
```

[13]: ['amarillo', 'rojo', 'rojo', 'verde', 'azul']

[14]:
```
# Puedo modificar el contenido de la lista

L[2] = 'verde'
print(L)
```

```
['azul', 'verde', 'verde', 'rojo', 'amarillo']
```

[15]:
```
L.append('rosa') # agregar un valor al final
L
```

[15]: ['azul', 'verde', 'verde', 'rojo', 'amarillo', 'rosa']

[16]:
```
L.insert(2, 'blue')    #L.insert(index, object) -- insertar un elemento usando␣
↪el índice
```

[17]:
```
L.extend(['magenta', 'violeta'])
L
```

[17]:
```
['azul',
 'verde',
 'blue',
 'verde',
 'rojo',
 'amarillo',
 'rosa',
 'magenta',
 'violeta']
```

[18]:
```
L.append(2)
L
```

[18]:
```
['azul',
 'verde',
 'blue',
 'verde',
 'rojo',
 'amarillo',
 'rosa',
 'magenta',
 'violeta',
 2]
```

[19]:
```
L = L[::-1] # Orden reverso
L
```

[19]:
```
[2,
 'violeta',
 'magenta',
 'rosa',
 'amarillo',
 'rojo',
 'verde',
 'blue',
 'verde',
 'azul']
```

[20]:
```
L2 = L[:-3] # Elimina ya que corta los últimos 3 elementos
print(L)
print(L2)
```

```
[2, 'violeta', 'magenta', 'rosa', 'amarillo', 'rojo', 'verde', 'blue', 'verde',
'azul']
[2, 'violeta', 'magenta', 'rosa', 'amarillo', 'rojo', 'verde']
```

[21]:
```
#L[25] # Fuera del rango, da error
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-21-531a86702de7> in <module>
----> 1 L[25] # Fuera del rango, da error

IndexError: list index out of range
```

[22]:
```
print(L)
print(L[20:25]) # No hay error cuando rebano (slicing).
print(L[20:])
print(L[2:20])
```

```
[2, 'violeta', 'magenta', 'rosa', 'amarillo', 'rojo', 'verde', 'blue', 'verde',
'azul']
[]
[]
['magenta', 'rosa', 'amarillo', 'rojo', 'verde', 'blue', 'verde', 'azul']
```

[23]:
```
print(L.count('yellow'))
```

```
0
```

[25]:
```
print(L.count('amarillo'))
```

```
1
```

[26]:
```
L1=L[2:]
L1.sort() # ¿usar tab para ver los metodos o funciones del objeto?
print(L1)
```

```
['amarillo', 'azul', 'blue', 'magenta', 'rojo', 'rosa', 'verde', 'verde']
```

[27]:
```
a = [1,2,3]
b = [10,20,30]
```

[28]:
```
print(a+b) # NO es lo que uno espera, pero tiene cierta lógica
```

```
[1, 2, 3, 10, 20, 30]
```

[29]:
```
#print(a*b) # No se pueden multiplicar listas (Usar Numpy -> la clase que␣
↪viene)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-29-69c1ea4e436c> in <module>
----> 1 print(a*b) # No se pueden multiplicar listas (Usar Numpy -> la clase qu
 ↪viene)

TypeError: can't multiply sequence by non-int of type 'list'
```

[30]:
```
L = list(range(4)) # Crear una lista a partir de un definición numérica. El
 ↪número es la cantidad de elementos sin contar el último

L
```

[30]:
```
[0, 1, 2, 3]
```

[31]:
```
L = list(range(2, 20, 2)) # cada 2 enteros
L
```

[31]:
```
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Los elementos no tienen que ser del mismo tipo.

[32]:
```
L = [1, '1', 1.4]
L
```

[32]:
```
[1, '1', 1.4]
```

Remuevo el n+1-avo elemento

[33]:
```
L = list(range(0,20,2))
print(L)
del L[5]
print(L)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
[0, 2, 4, 6, 8, 12, 14, 16, 18]
```

Slicing: extrayendo parte de la lista

[34]:
```
a = [[1, 2, 3], [10, 20, 30], [100, 200, 300]] # No es 2D (no es una MATRIZ,
                                                # OJO es una tabla de tablas
print(a)
print(a[0])
print(a[1][1])
```

```
[[1, 2, 3], [10, 20, 30], [100, 200, 300]]
[1, 2, 3]
20
```

[35]:
```
#print(a[1,1]) # NO FUNCIONA
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-7d58e8635b16> in <module>
----> 1 print(a[1,1]) # NO FUNCIONA

TypeError: list indices must be integers or slices, not tuple
```

[36]:
```
b = a[1]
print(b)
```

```
[10, 20, 30]
```

[37]:
```
b[1] = 999
print(b)
```

```
[10, 999, 30]
```

[38]:
```
print(a) # Cambiando b cambiamos a, Cuidado !!!
```

```
[[1, 2, 3], [10, 999, 30], [100, 200, 300]]
```

[39]:
```
b[1] is a[1][1]
```

[39]:
```
True
```

[40]:
```
c = a[1][::] # copiar en vez de rebanar
print(c)
c[0] = 77777
print(c)
print(a)
```

```
[10, 999, 30]
[77777, 999, 30]
[[1, 2, 3], [10, 999, 30], [100, 200, 300]]
```

2.2 Tuplas

Son como las listas, pero sus valores son inmutables (no se pueden modificar durante la corrida del programa)

En muchas maneras las tuplas son parecidas a las listas, pero la diferencia más importante es que las tuplas pueden usarse como índices (keys) en un diccionario y como elementos de un set, mientras las listas NO PUEDEN ser usadas. Veremos que son diccionarios y set a continuación.

Usan () en vez de las [].

[41]:
```
T = (1,2,3)
T
```

[41]: (1, 2, 3)

[42]:
```
T2 = 1, 2, 3
print(T2)
type(T2)
```

```
(1, 2, 3)
```

[42]: tuple

[43]:
```
T[1]
```

[43]: 2

[ ]:
```
### Las tuplas son inmutables
```

[44]:
```
#T(1) = 3 # No funciona
```

```
  File "<ipython-input-44-582110943611>", line 1
    T(1) = 3 # No funciona
           ^
SyntaxError: can't assign to function call
```

[45]:
```
T=(3,2,1)  # la tengo que escribir toda de nuevo
T
```

[45]: (3, 2, 1)

2.3 Diccionarios

Permiten organizar los datos en campos con nombre apropiados

[47]:
```
a_dictionary = {'uno' : 1.0,
                'dos' : 2.0,
                'corazon' : '\u2665',
                'una_lista' : ['esta', 'es', 'una', 'lista']
                }
```

Para acceder a los datos:

[49]:
```
print(a_dictionary['corazon'])
print(a_dictionary['una_lista'])
print(a_dictionary['uno'])
```

```
['esta', 'es', 'una', 'lista']
1.0
```

3 Estructuras de control

Bloques son definidos por "indentation" o sea la sangría. No hay indicación de final

[50]:
```
for i in [1,2,3]:
      print(i+1)        # Prestar atención a la sangría (indentation en inglés)
```

```
2
3
4
```

[51]:
```
for cosa in [1,'ff',2]:
    print(cosa)
    print('end')
print('final end')   # El final de la sangría indica el final del bloque "for"
```

```
1
end
ff
end
2
end
final end
```

[52]:
```
# Si defino un diccionario
ATOMIC_MASS = {}  # <-- Diccionario vacio

ATOMIC_MASS['H'] = 1
ATOMIC_MASS['He'] = 4
ATOMIC_MASS['C'] = 12
ATOMIC_MASS['N'] = 14
ATOMIC_MASS['O'] = 16
ATOMIC_MASS['Ne'] = 20
ATOMIC_MASS['Ar'] = 40
ATOMIC_MASS['S'] = 32
ATOMIC_MASS['Si'] = 28
ATOMIC_MASS['Fe'] = 55.8

# Puedo imprimir a partir de la keys, todos los valores del diccionario. Ojo.␣
 ↪esto no sale ordenado
```

```
for key in ATOMIC_MASS.keys():
    print(key, ATOMIC_MASS[key])
```

```
H 1
He 4
C 12
N 14
O 16
Ne 20
Ar 40
S 32
Si 28
Fe 55.8
```

Se pueden utilizar las sentencias de Control con un estilo bastante parecido al Fortran

[53]:
```
for i in range(10):
    if i > 5:
        print(i)

# Prestar atención a la doble sangría del if
```

```
6
7
8
9
```

[54]:
```
for i in range(10):
    if i > 5:
        print(i)
    else:
        print(i ,'es menor que cinco')
print('END')
```

```
0 es menor que cinco
1 es menor que cinco
2 es menor que cinco
3 es menor que cinco
4 es menor que cinco
5 es menor que cinco
6
7
8
9
END
```

3.1 SET

Los "sets" son listas sin numerar, es decir los elementos no tienen una posicón númerica con la cual el programa podría elegirlos para una actividad. Se usan a ciegas.

Se distinguen de las listas por el uso de los {}.

No son muy usados, pero existen y esán disponibles.

Ejemplo:

[55]:
```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
apple
banana
cherry
```

3.2 Funciones

[56]:
```
def func1(x):
    print(x**3)

func1(5)
```

```
125
```

[57]:
```
def func2(x):
    """
    Return the cube of the parameter
    """
    return(x**3)

a = func2(3)

print(a)

help(func2)
```

```
27
Help on function func2 in module __main__:

func2(x)
    Return the cube of the parameter
```

[ ]:
```
# Otra manera hacer shift tab dentro del parentesis
```

```
func2(3)
```

[58]:
```
print(a)
print(func2(4))
```

```
27
64
```

[59]:
```
def func3(x, y, z, a=0, b=1):
    """
    Esta función tiene 5 argumentos, pero dos de ellos tiene valores por␣
 ↪default (entonces no son obligatorios)
    """
    return a + b * (x**2 + y**2 + z**2)**0.5
D = func3(3, 4, 5)
print(D)
```

```
7.0710678118654755
```

[60]:
```
E = func3(3, 4, 5, 10, 100)
print(E)
```

```
717.1067811865476
```

La función lambda es para crear funciones de una línea:

[61]:
```
J = lambda x, y, z: (x**2 + y**2 + z**2)**0.5
J(1,2,3)
```

[61]:
```
3.7416573867739413
```

Recursividad n!=n(n-1)!=n(n-1)(n-2)

[62]:
```
def fact(n):
    if n <= 0:
        return 1
    return n*fact(n-1)


print(fact(5))
print(fact(20))
print(fact(10))
```

```
120
2432902008176640000
3628800
```

3.3 Funciones escritas en archivos (scripts)

4 Tengo en una archivo que se llama ex1, la función suguiente

def f1(x):

```
"""
Este es un ejemplo de la función que devuelve: x**2    - parameter: x
"""
return x**2
```

[63]:
```
import ex1

# Esto importa un archivo que se llama ex1.py del directorio
# o de los directorios que indique poniendo el camino
# from one of the directories in the search path
# es lo que impulsó mucho al Python en el área científica.

print(ex1.f1(4))
```

```
16
```

[64]:
```
from ex1 import f1
print(f1(3))

#Cargo de ex1 SÓLO la función f1
```

```
9
```

[65]:
```
from ex1 import * # aquí cargo todas las funciones. Lo cual es una mala idea se
# cargan todas las funciones con su nombre y pueden sobreescribir otros nombres
# que estaba usando para las funciones

print(f1(4))
```

```
16
```

[ ]:
```
import ex1 as tt
print(tt.f1(10))

# Ahora cargo todas las funciones de ex1, pero las tengo que llamar con el␣
 ↪prefijo
# tt en el nombre, es decir la función f1() es ahora tt.f1
```

4.1 Importando Librerías

No todas las capacidades de python están en el lenguaje, sino que son externas al Python. Para usar estas capacidades hay que "importarlas" Este incluso es el caso para usar funcioanes matemáticas simples. Por ejemplo:

[66]:
```
print(sin(3.))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-66-23e9b31fd5de> in <module>
----> 1 print(sin(3.))

NameError: name 'sin' is not defined
```

[67]:
```
import math
print(math.sin(3.))
```

```
0.1411200080598672
```

[68]:
```
help(math.sin)
```

```
Help on built-in function sin in module math:

sin(x, /)
    Return the sine of x (measured in radians).
```

[69]:
```
print(math.pi)
```

```
3.141592653589793
```

5 FIN

[ ]:
```

```