Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Clase 12: Tuplas

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

28 de Abril, 2015

Clases pasadas (string)

Definición: Un String es una cadena *inmutable* e *indexable* de caracteres.

Sintaxis

```
s = 'mi string'
s = "mi string"
s = """mi string"""
```

```
1 s1 = ""                    # String vacío
2 s2 = "yo soy tu padre"     # String Star Wars
```

Clases pasadas (string)

Indexable: Sus caracteres se obtienen indicando su índice.

```
1 s = "yo soy tu padre"
2 print(s[3])   # >>> 's'
3 print(s[-2])  # >>> 'r'
```

Inmutable: Sus caracteres no pueden ser modificados.

```
1 s = "yo soy tu padre"
2 s[0] = "Y"
3 # >>> TypeError: 'str' object does not support item
      assignment
```

Clases pasadas (string)

For: Permite recorrer un string y ejecutar código para cada uno de sus caracteres.

Sintaxis

for variable **in** string**:**
    bloque_de_código_for

```
1 s = "yo soy tu padre"
2 s2 = ""
3 for c in s:
4   s2 += chr(ord(c)+1)
5 print(s2) # >>> 'zp!tpz!uv!qbesf'
```

Clases pasadas (string)

Funciones sobre strings:

- `s[i:j]`
- `len(s)`
- `s1+s2`
- `s.find(c)`
- `s.lower()`
- `s.strip()`
- `s.replace(c1,c2)`
- `s.split(c)`
- `c.join(l)`

Clases pasadas (string)

Veamos de qué están hechos...

```
1  def misterio1(s,k): # s -> string, k -> int
2    i = 0
3    for c in s:
4      i += 1
5      if(c == " "):
6        if(i > k):
7          return True
8        i = 0
9    return False
```

```
11 def misterio2(s):    #s -> string
12   for k in range(len(s)):
13     if(not misterio1(s,k)):
14       return k-1
15   return 0
```

Clases pasadas (listas)

Definición: Una lista es una serie *mutable* e *indexable* de elementos.

Sintaxis

lista = [elemento_1, elemento_2, elemento_3, ...]

```
l1 = []                                  # lista vacía
l2 = [3,4,2,4,9,6]                       # lista de números
l3 = ["a","b","c","d"]                   # lista de strings
l4 = ["a",3.4,True]                      # lista mixta
l5 = [[1,2,3],[4,5,6],[7,8,9]]           # lista de listas
```

Clases pasadas (listas)

Indexable: Sus elementos se obtienen indicando su índice.

```
1 l = [3,4,2,4,9,6]   # lista de números
2 print(l[3])         # >>> 4
3 print(l[-2])        # >>> 9
```

Mutable: Sus elementos pueden ser modificados.

```
1 l = [3,4,2,4,9,6]   # lista de números
2 l[0] = 6
3 print(l)            # >>> [6,4,2,4,9,6]
```

Clases pasadas (listas)

For: Permite recorrer una lista y ejecutar código para cada elemento en ella.

Sintaxis

for variable **in** lista (o string)**:**
    bloque_de_código_for

```
1 l = ['fruta', 'carne', 'arroz', 'café']
2
3 # Recorre cada elemento de "l"
4 for e in l:
5   # Operamos con el "e" actual
6   print("No olvides comprar",e)
```

Clases pasadas (listas)

Funciones sobre listas:

- `l[i:j]`
- `len(l)`
- `l1+l2`
- `l.append(x)`
- `l.index(x)`
- `l.insert(i,x)`
- `l.remove(x)`
- `l.sort()`

Clases pasadas (string)

Veamos de qué están hechos (parte 2)...

```
1  def misterio3(p,q):
2    a = []
3    for i in range(p):
4      b = []
5      for j in range(q):
6        b.append(0)
7      a.append(b)
8    return a
9
10 a = misterio3(5,6)
11 a[3][2] = 1
12 a[5][2] = 2
13 a["A"][2] = 3
```

Clases pasadas (archivos)

notas.txt

```
6.7
4.4
6.7
2.9
4.9
2.1
2.8
5.6
4.6
2.4
2.4
3.8
4.2
```

Clases pasadas (archivos)

Leer un archivo:

```
1 f = open("./notas.txt")
2 for l in f:
3   print(l.rstrip())
4 f.close()
```

Escribir un archivo:

```
1 from random import random
2 f = open("notas.txt","w")
3 for i in range(79):
4   f.write(str(random()*6+1) + "\n")
5 f.close()
```

Clases pasadas (string)

Veamos de qué están hechos (parte 3)...

```
1  f = open("./notas.txt")
2  l = []
3  for s in f:
4    l.append(float(s.rstrip()) + 1)
5  f.close()
6
7  f = open("notas.txt","w")
8  for n in l:
9    f.write(str(n) + "\n")
10 f.close()
```

Tuplas

Definición: Una tupla es una secuencia ***inmutable*** e *indexable* de elementos.

Tuplas

Definición: Una tupla es una secuencia ***inmutable*** e *indexable* de elementos.

Sintaxis

lista = (elemento_1, elemento_2, elemento_3, ...)

Tuplas

Definición: Una tupla es una secuencia ***inmutable*** e *indexable* de elementos.

Sintaxis

lista = (elemento_1, elemento_2, elemento_3, ...)

```
1 t0 = ()                          # Tupla vacía
2 t1 = (3,)                        # Tupla con un elemento
3 t2 = (5,3,9)                     # Tupla de números
4 t3 = ('Juan','Pérez')            # Tupla de string
5 t4 = ('Juan','Pérez', 18)        # Tupla mixta
6 t5 = (t1,t2,t3)                  # Tupla de tuplas
```

Tuplas

Indexable: Sus elementos se obtienen indicando su índice.

```
1 t1 = (5,3,9)                  # Tupla de números
2 t2 = ('Juan','Pérez')         # Tupla de string
3 t3 = ('Juan','Pérez', 18)     # Tupla mixta
4 t4 = (t1,t2,t3)               # Tupla de tuplas
5
6 print(t4[0][2])               # >>> 9
7 print(t4[1][-1])              # >>> 'Perez'
8 print(t4[2][2])               # >>> 18
```

Tuplas

Indexable: Sus elementos se obtienen indicando su índice.

```
t1 = (5,3,9)                    # Tupla de números
t2 = ('Juan','Pérez')           # Tupla de string
t3 = ('Juan','Pérez', 18)       # Tupla mixta
t4 = (t1,t2,t3)                 # Tupla de tuplas

print(t4[0][2])                 # >>> 9
print(t4[1][-1])                # >>> 'Perez'
print(t4[2][2])                 # >>> 18
```

Inmutable: Sus elementos NO pueden ser modificados.

```
t1 = (5,3,9)        # Tupla de números
t1[0] = 2           # Error!
t1[1] = t1[1]+1     # Error!
```

Tuplas

Podemos recorrerlas mediante **for**.

```
1 t = (5,3,9,8,6,5,34,2)
2 for i in t:
3   print(i)
```

Tuplas

Podemos recorrerlas mediante **for**.

```
1 t = (5,3,9,8,6,5,34,2)
2 for i in t:
3   print(i)
```

... y usar los operadores +, *, **in**, **not in**.

```
1 t1 = (1,2,3)
2 t2 = (4,5,6)
3 print(t1+t2)          # >>> (1,2,3,4,5,6)
4 print(2*t2)           # >>> (4,5,6,4,5,6)
5 print(2 in t1)        # >>> True
6 print(2 not in t2)    # >>> True
```

Tuplas: Funciones

Podemos obtener una sub-tupla mediante **t[i:j:k]**.

```
1 t = (5,3,9,8,6,5,34,2)
2 print(t[:4])                  # >>> (5,3,9,8)
3 print(t[6:])                  # >>> (34,2)
4 print(t[::-1])                # >>> (2,34,5,6,8,9,3,5)
```

Tuplas: Funciones

Podemos obtener una sub-tupla mediante **t[i:j:k]**.

```
1 t = (5,3,9,8,6,5,34,2)
2 print(t[:4])                # >>> (5,3,9,8)
3 print(t[6:])                # >>> (34,2)
4 print(t[::-1])              # >>> (2,34,5,6,8,9,3,5)
```

... y usar las funciones **len()**, **min()** y **max()**.

```
1 t = (5,3,9,8,6,5,34,2)
2 print(len(t))               # >>> 8
3 print(max(t))               # >>> 34
4 print(min(t))               # >>> 2
```

Listas vs Tuplas

¿Es decir las tuplas son listas inmutables?

Listas vs Tuplas

¿Es decir las tuplas son listas inmutables?

¿O sea, son listas limitadas?

Listas vs Tuplas

¿Es decir las tuplas son listas inmutables?

¿O sea, son listas limitadas?

... entonces cuál es su gracia?

Listas vs Tuplas

Importante: Las tuplas se usan para agrupar datos que *naturalmente* deberían ir juntos.

Listas vs Tuplas

Importante: Las tuplas se usan para agrupar datos que *naturalmente* deberían ir juntos.

```
# coordenadas paralelepípedo (x,y,z)
p1 = (0,0,0); p2 = (0,0,1); p3 = (0,1,0)
p4 = (0,1,1); p5 = (1,0,0); p6 = (1,0,1)
p7 = (1,1,0); p8 = (1,1,1)

# usuarios (nombre, apellido, correo, edad)
u1 = ('juan','gonzalez','jgonz@uc.cl',19)
u2 = ('maría','mardoqueo','mjmard@uc.cl',18)

# Próximos feriados (día, mes, año)
f1 = ( 1,5,2015); f2 = (21,5,2015)
f3 = (29,6,2015); f4 = (16,7,2015)
f5 = (15,8,2015); f6 = (18,9,2015)
```

Listas vs Tuplas

Tupla: Secuencia heterogénea de elementos (**t[i]** tiene semántica).

Listas vs Tuplas

Tupla: Secuencia heterogénea de elementos (**t[i]** tiene semántica).

```
# coordenadas paralelepípedo (x,y,z)
p1 = (0,0,0); p2 = (0,0,1); p3 = (0,1,0)
p4 = (0,1,1); p5 = (1,0,0); p6 = (1,0,1)
p7 = (1,1,0); p8 = (1,1,1)

# usuarios (nombre, apellido, correo, edad)
u1 = ('juan','gonzalez','jgonz@uc.cl',19)
u2 = ('maría','mardoqueo','mjmard@uc.cl',18)

# Próximos feriados (día, mes, año)
f1 = ( 1,5,2015); f2 = (21,5,2015)
f3 = (29,6,2015); f4 = (16,7,2015)
f5 = (15,8,2015); f6 = (18,9,2015)
```

Listas vs Tuplas

Lista: Secuencia homogénea de elementos (**t[i]** es un elemento cualquiera).

Listas vs Tuplas

Lista: Secuencia homogénea de elementos (**t[i]** es un elemento cualquiera).

```
1 # coordenadas paralelepípedo (x,y,z)
2 p1 = (0,0,0); p2 = (0,0,1); p3 = (0,1,0)
3 p4 = (0,1,1); p5 = (1,0,0); p6 = (1,0,1)
4 p7 = (1,1,0); p8 = (1,1,1)
5
6 # lista con puntos del paralelepípedo
7 l = [p1,p2,p3,p4,p5,p6,p7,p8]
```

Listas vs Tuplas

Lista: Secuencia homogénea de elementos (**t[i]** es un elemento cualquiera).

```
1 # coordenadas paralelepípedo (x,y,z)
2 p1 = (0,0,0); p2 = (0,0,1); p3 = (0,1,0)
3 p4 = (0,1,1); p5 = (1,0,0); p6 = (1,0,1)
4 p7 = (1,1,0); p8 = (1,1,1)
5
6 # lista con puntos del paralelepípedo
7 l = [p1,p2,p3,p4,p5,p6,p7,p8]
```

- **l[3]** es un punto cualquiera.
- **l[3][1]** es la coordenada **y** de **l[3]**.

Listas vs Tuplas

Lista: Secuencia homogénea de elementos (**t[i]** es un elemento cualquiera).

```
1 # coordenadas paralelepípedo (x,y,z)
2 p1 = (0,0,0); p2 = (0,0,1); p3 = (0,1,0)
3 p4 = (0,1,1); p5 = (1,0,0); p6 = (1,0,1)
4 p7 = (1,1,0); p8 = (1,1,1)
5
6 # lista con puntos del paralelepípedo
7 l = [p1,p2,p3,p4,p5,p6,p7,p8]
```

- **l[3]** es un punto cualquiera.
- **l[3][1]** es la coordenada **y** de **l[3]**.

... esto lo garantiza la *inmutabilidad* de las tuplas.

Listas vs Tuplas

Otras ventajas:

- Son más rápidas.
- Son *read only*.

Tuplas: packing y unpacking

Packing: Reunir varias variables (o valores) en una tupla.

```
1 # datos usuario
2 nombre = "margarita"
3 apellido = "castro"
4 correo = "mjcastro@uc.cl"
5
6 # packing
7 usuario = nombre, apellido, correo
```

Tuplas: packing y unpacking

Packing: Reunir varias variables (o valores) en una tupla.

```
1 # datos usuario
2 nombre = "margarita"
3 apellido = "castro"
4 correo = "mjcastro@uc.cl"
5
6 # packing
7 usuario = nombre, apellido, correo
```

Unpacking: Pasar valores de una tupla a varias variables.

```
1 # punto
2 P = 3,6,5
3
4 # unpacking
5 x,y,z = P
```

Tuplas: Retornos de funciones

¿Python permite retornos múltiples?

Tuplas: Retornos de funciones

¿Python permite retornos múltiples?

```
1  # Recibe segundos y retorna el
2  # equivalente en H,M,S
3  def obtenerHMS(s):
4    H = s //3600
5    M = (s % 3600) //60
6    S = s % 60
7    return H,M,S
8
9  # llamado a la función
10 h,m,s = obtenerHMS(4523)
11 print(h,":",m,":",s)    # >>> '1 : 15 : 23'
```

Tuplas: Cambiar valores

Mini-desafío:

Considere el siguiente código

```
1 a = int(input())
2 b = int(input())
```

... cree un programa que intercambie los valores de `a` y `b` (deje en `a` el valor de `b` y en `b` el valor de `a`).

Tuplas: Cambiar valores

Solución: Necesitamos usar una variable auxiliar...

```
1 # valores iniciales
2 a = 3; b = 5
3 # intercambiarlos
4 x = a
5 a = b
6 b = x
7 print("a",a,"b",b)
```

Tuplas: Cambiar valores

Solución: Necesitamos usar una variable auxiliar...

```
# valores iniciales
a = 3; b = 5
# intercambiarlos
x = a
a = b
b = x
print("a",a,"b",b)
```

Sin embargo, las tuplas permiten hacerlo en una línea.

```
# valores iniciales
a = 3; b = 5
# intercambiarlos
(b,a) = (a,b)
print("a",a,"b",b)
```

Tuplas: Ejemplo

"Cree una función que reciba una lista de puntos como tuplas (x,y,z), y retorne el id del par más cercano"

Estrategia:

Tuplas: Ejemplo

"Cree una función que reciba una lista de puntos como tuplas (x,y,z), y retorne el id del par más cercano"

Estrategia:

- Función para calcular distancia entre pares de puntos:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Tuplas: Ejemplo

"Cree una función que reciba una lista de puntos como tuplas (x,y,z), y retorne el id del par más cercano"

Estrategia:

- Función para calcular distancia entre pares de puntos:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

- Crear función que resuelva el problema.

Tuplas: Ejemplo

"Cree una función que reciba una lista de puntos como tuplas (x,y,z), y retorne el id del par más cercano"

Estrategia:

- Función para calcular distancia entre pares de puntos:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

- Crear función que resuelva el problema.
  - Calcular distancia entre todo par de puntos (loops anidados).
  - Recordar la mínima distancia encontrada hasta el momento.
  - Al finalizar el loop, retornamos el mínimo encontrado.

Tuplas: Ejemplo

Función para encontrar la distancia entre 2 puntos.

Tuplas: Ejemplo

Función para encontrar la distancia entre 2 puntos.

```
1 def calcular_distancia(p1,p2):
2   x = p1[0]-p2[0]
3   y = p1[1]-p2[1]
4   z = p1[2]-p2[2]
5   return (x**2+y**2+z**2)**0.5
```

Tuplas: Ejemplo

Función que resuelve el problema planteado.

Tuplas: Ejemplo

Función que resuelve el problema planteado.

```
7  def min_distancia(l):
8    par_min = ()
9    val_min = float("inf")      # valor infinito
10   for i in range(len(l)):
11     for j in range(i+1,len(l)):
12       d = calcular_distancia(l[i],l[j])
13       if(d < val_min):
14         val_min = d
15         par_min = i,j
16   return par_min
```

Tuplas: Ejemplo

Para probar nuestro función.

```
18 # test
19 l1 = [(1,1,1),(3,3,3),(0,3,9)]
20 l2 = [(8,3,1),(6,3,1),(3,7,2)]
21 l3 = [(1,4,1),(3,7,3),(9,2,2)]
22 l4 = [(9,1,3),(1,1,2),(0,5,2)]
23 l = l1+l2+l3+l4
24 i,j = min_distancia(l)
25 print(l[i],l[j])          # >>> (1,1,1) (1,1,2)
```

Ejercicios

1) Cree una función que reciba una lista de puntos (tuplas: x, y, z) y retorne el perímetro de la figura. Considere que cada par de puntos consecutivos están conectados, y **l[len(n) - 1]** está conectado a **l[0]**.

2) Cree una función que reciba una lista de usuarios (tuplas: nombre, apellido, correo, edad) y muestre:

- El número de usuarios.
- El nombre del usuario más viejo.
- El nombre del usuario más joven.
- El nombre del usuario con apellido más largo.
- La edad promedio de los usuarios.

Ejercicios

3) Cree una función que reciba una tupla **t**, un id **i** y una variable **c**. Su función debe retornar una tupla igual a **t**, pero con $\mathbf{t[i]} = \mathbf{c}$ (si $\mathbf{i} \geq \mathbf{len(t)}$, agregue **c** al final de **t**).

4) Cree una función que reciba dos tuplas y retorne el número de elementos distintos en ellas.

5) Cree una función que reciba una lista de puntos (tuplas: x, y, z) y retorne la lista ordenada según la distancia entre cada punto y el origen. **Warning**: Este es difícil...