El futuro digital es de todos

MinTIC

Tuplas

OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

Tuplas

Las tuplas son inmutables

Una tupla es una secuencia de valores muy parecida a una lista. Los valores almacenados en una tupla pueden ser de cualquier tipo, y están indexados por números enteros. La diferencia importante es que las tuplas son inmutables. Las tuplas también son comparables y hashables, por lo que podemos ordenar listas de ellas y utilizar las tuplas como valores clave en los diccionarios de Python.

Sintácticamente, una tupla es una lista de valores separada por comas:

```
In [1]: t = 'a', 'b', 'c', 'd', 'e'
```

Aunque no es necesario, es común encerrar las tuplas entre paréntesis para ayudarnos a identificar rápidamente las tuplas cuando miramos el código de Python:

```
In [2]: t = ('esto es una cadena', 'b', 'c', 'd', 'e')
        print(t)
        t[0]

        ('esto es una cadena', 'b', 'c', 'd', 'e')

Out[2]: 'esto es una cadena'
```

Para crear una tupla con un solo elemento, hay que incluir la coma final:

```
In [3]: t1 = ('a',)
        type(t1)

Out[3]: tuple
```

```
In [3]: t1 = ('a',)
        type(t1)
```

```
Out[3]: tuple
```

Sin la coma Python trata ('a') como una expresión con una cadena entre paréntesis que evalúa a una cadena:

```
In [ ]: t2 = ('a')
        type(t2)
```

Otra forma de construir una tupla es la tupla de función incorporada. Sin ningún argumento, crea una tupla vacía:

```
In [6]: t = tuple()
        print(type(t))
```

```
<class 'tuple'>
```

Si el argumento es una secuencia (cadena, lista o tupla), el resultado de la llamada a la tupla es una tupla con los elementos de la secuencia:

```
In [7]: t = tuple('lupines')
        print(t)
```

```
('l', 'u', 'p', 'i', 'n', 'e', 's')
```

Debido a que tupla es el nombre de un constructor, debes evitar usarlo como un nombre de variable.

La mayoría de los operadores de listas también trabajan con tuplas. El operador de corchetes indexa un elemento:

```
In [15]: t = ('a', 'b', 'c', 'd', 'e')
         print(t)

         ('a', 'b', 'c', 'd', 'e')
```

Y el operador de rebanadas selecciona una serie de elementos.

```
In [16]: print(t[1:3])

         ('b', 'c')
```

Pero si intentas modificar uno de los elementos de la tupla, obtienes un error:

```
In [17]: t[0] = 'A'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-7e674cdf20e6> in <module>
----> 1 t[0] = 'A'

TypeError: 'tuple' object does not support item assignment
```

No se pueden modificar los elementos de una tupla, pero se puede reemplazar una tupla por otra:

```
In [18]: t = ('A',) + t[1:]
         print(t)

('A', 'b', 'c', 'd', 'e')
```

Comparando tuplas

Los operadores de comparación trabajan con tuplas y otras secuencias. Python comienza comparando el primer elemento de cada secuencia. Si son iguales, pasa al siguiente elemento, y así sucesivamente, hasta encontrar elementos que difieren. Los elementos subsiguientes no se consideran (aunque sean realmente grandes).

```
In [19]: (0, 1, 2) < (0, 3, 4)
Out[19]: True

In [20]: (0, 1, 2000000) < (0, 3, 4)
Out[20]: True
```

La función de **sort** funciona de la misma manera. Se ordena principalmente por el primer elemento, pero en el caso de un empate, se clasifica por el segundo elemento, y así sucesivamente.

Esta característica se presta a un patrón llamado DSU.

Decorate: ordenar una secuencia construyendo una lista de tuplas con una o más claves de clasificación que preceden a los elementos de la secuencia,
Sort: Ordena la lista de tuplas usando la clasificación incorporada de Python, y
Undecorate: desordena la lista extrayendo los elementos ordenados de la secuencia.

Por ejemplo, supongamos que tienes una lista de palabras y quieres ordenarlas de más a menos:

In [23]:
```
txt = "but soft what light in yonder window breaks"
palabras = txt.split()
palabras
```

Out[23]:
```
['but', 'soft', 'what', 'light', 'in', 'yonder', 'window', 'breaks']
```

In [26]:
```
l = list()
for subcadena in palabras:
    l.append((len(subcadena), subcadena))
print(l)
```

```
[(3, 'but'), (4, 'soft'), (4, 'what'), (5, 'light'), (2, 'in'), (6, 'yonder'), (6, 'window'), (6, 'breaks')]
```

In [28]:
```
l.sort(reverse=True)
print(l)

res = list()

for longitud, palabra in l:
    res.append(palabra)

print(res)
```

```
[(6, 'yonder'), (6, 'window'), (6, 'breaks'), (5, 'light'), (4, 'what'), (4, 'soft'), (3, 'but'), (2, 'in')]
['yonder', 'window', 'breaks', 'light', 'what', 'soft', 'but', 'in']
```

El primer bucle construye una lista de tuplas, donde cada tupla es una palabra precedida por su longitud.

La función **sort** compara el primer elemento, la longitud, primero, y sólo considera el segundo elemento para romper los empates. El argumento de la palabra clave reverse=True le dice a **sort** que vaya en orden decreciente.

El segundo bucle atraviesa la lista de tuplas y construye una lista de palabras en orden descendente de longitud. Las palabras de cuatro caracteres se ordenan en orden alfabético inverso, así que "what" aparece antes de **"soft"** en la siguiente lista. El resultado del programa es el siguiente:

['yonder', 'window', 'breaks', 'light', 'what', 'soft', 'but', 'in']

Por supuesto, la línea pierde mucho de su impacto poético cuando se convierte en una lista Python y se ordena en orden descendente de longitud de palabra.

Asignación de tupla

Una de las características sintácticas únicas del lenguaje Python es la capacidad de tener una tupla en el lado izquierdo de una declaración de asignación. Esto permite asignar más de una variable a la vez cuando el lado izquierdo es una secuencia.

En este ejemplo tenemos una lista de dos elementos (que es una secuencia) y asignamos el primer y segundo elemento de la secuencia a las variables x e y en una única sentencia.

```
In [3]: m = ['have', 'fun']
        x, y = m

In [4]: print(x)
        print(y)

have
fun
```

No es magia, Python traduce aproximadamente la sintaxis de la asignación de tupla como la siguiente:

```
In [5]:  m = ['have', 'fun']
         x = m[0]
         y = m[1]

In [6]:  print(x)
         print(y)

         have
         fun
```

Cuando usamos una tupla en el lado izquierdo de la declaración de asignación, omitimos los paréntesis, pero la siguiente es una sintaxis igualmente válida:

```
In [7]: m = [ 'have', 'fun' ]
        (x, y) = m

In [8]: print(x)
        print(y)

have
fun
```

Una aplicación particularmente inteligente de la asignación de tupla nos permite intercambiar los valores de dos variables en una sola declaración:

```
In [9]: a, b = b, a
```

Ambos lados de esta declaración son tuplas, pero el lado izquierdo es una tupla de variables; el lado derecho es una tupla de expresiones. Cada valor del lado derecho se asigna a su respectiva variable del lado izquierdo. Todas las expresiones del lado derecho se evalúan antes de cualquiera de las asignaciones.

El número de variables de la izquierda y el número de valores de la derecha deben ser iguales:

```
In [10]: a, b = 1, 2, 3
```

En general, el lado derecho puede ser cualquier tipo de secuencia (cadena, lista o tupla). Por ejemplo, para dividir una dirección de correo electrónico en un nombre de usuario y un dominio, se podría escribir:

```
In [11]: addr = 'monty@python.org'
         uname, domain = addr.split('@')
```

El valor de retorno de la división es una lista con dos elementos; el primer elemento se asigna a uname, el segundo a dominio.

```
In [12]: print(uname)
         print(domain)

         monty
         python.org
```

Diccionarios y tuplas

Los diccionarios tienen un método llamado elementos que devuelve una lista de tuplas, en la que cada tupla es un par clave-valor:

```
In [13]: d = {'a':10, 'b':1, 'c':22}
         t = list(d.items())
         print(t)

         [('a', 10), ('b', 1), ('c', 22)]
```

Como es de esperar de un diccionario, los artículos no están en un orden particular.

Sin embargo, como la lista de tuplas es una lista, y las tuplas son comparables, podemos ahora ordenar la lista de tuplas. Convertir un diccionario en una lista de tuplas es una forma de obtener el contenido de un diccionario ordenado por clave:

```
In [14]:
d = {'a':10, 'b':1, 'c':22}
t = list(d.items())
print(t)
```

```
[('a', 10), ('b', 1), ('c', 22)]
```

```
In [15]:
t.sort()
print(t)
```

```
[('a', 10), ('b', 1), ('c', 22)]
```

La nueva lista está ordenada en orden alfabético ascendente por el valor clave.

Usando tuplas como claves en los diccionarios

Debido a que las tuplas son hashable y las listas no, si queremos crear una clave compuesta para usar en un diccionario debemos usar una tupla como clave.

Nos encontraríamos con una clave compuesta si quisiéramos crear un directorio telefónico que mapee desde pares de apellidos y nombres a números de teléfono.

Método

count()

Este método recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la tupla.

```
In [49]: valores = ("Python", True, "Zope", 5)
         print("True ->", valores.count(True))

         True -> 1

In [50]: print("'Zope' ->", valores.count('Zope'))

         'Zope' -> 1

In [51]: print("5 ->", valores.count(5))

         5 -> 1
```

index()

Comparte el mismo método **index()** del tipo lista. Este método recibe un elemento como argumento, y devuelve el índice de su primera aparición en la tupla.

```
In [52]: valores = ("Python", True, "Zope", 5)

In [53]: print(valores.index(True))

         1

In [54]: print(valores.index(5))

         3
```

Seguimiento del número de la numeración:

Una tarea común es iterar sobre una secuencia mientras cuidas el seguimiento de la numeración de un elemento.

Podría usar un bucle while con un contador o un bucle **for** usando la función **range()** y la función **len():**

In [59]:
```
tecnologias = ('Zope', 'Plone', 'Pyramid')
for i in range(0, len(tecnologias)):
    print(i, tecnologias[i])
```

```
0 Zope
1 Plone
2 Pyramid
```

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022