Tuplas y Sets

March 30, 2022

Ejecuta el siguiente bloque de código siempre antes de ejecutar el resto del notebook.

[ ]:
```
from IPython.core.magic import  Magics, magics_class, cell_magic, line_magic

@magics_class
class Helper(Magics):

  def __init__(self, shell=None,  **kwargs):
    super().__init__(shell=shell, **kwargs)

  @cell_magic
  def debug_cell_with_pytutor(self, line, cell):
    import urllib.parse
    url_src = urllib.parse.quote(cell)
    str_begin = '<iframe width="1000" height="500" frameborder="0" src="https://
↪pythontutor.com/iframe-embed.html#code='
    str_end   = '&cumulative=false&py=3&curInstr=0"></iframe>'
    import IPython
    from google.colab import output
    display(IPython.display.HTML(str_begin+url_src+str_end))

get_ipython().register_magics(Helper)
```

#Estructuras de datos 2

0.1 Tuplas

La tupla es una colección de objetos de Python muy parecida a una lista. La secuencia de valores almacenados en una tupla puede ser de cualquier tipo, incluyendo también todas las estructuras de datos, además, están indexados por enteros, nuevamente partiendo por el 0.

Al definir la tupla, sus valores se separan mediante "comas". Aunque no es necesario, es más común definir una tupla cerrando la secuencia de valores entre paréntesis. Esto ayuda a entender las tuplas de Python más fácilmente.

```
tupla = (1,2,3,'a','b','c')
```

> **Nota**: La creación de tuplas en Python sin el uso de paréntesis se conoce como empaquetamiento de tuplas.

[ ]:
```
tupla = ("manzana", 1234, "fresa", [1,2,3,4,5], (5,9,7,5,4))
print(tupla)
```

```
('manzana', 1234, 'fresa', [1, 2, 3, 4, 5], (5, 9, 7, 5, 4))
```

###Longitud de la tupla Para determinar cuántos elementos tiene una tupla, utilice la función `len()`:

[ ]:
```
print(len(tupla))
```

```
5
```

Para crear una tupla de un único dato, es necesario añadir una coma al final del dato, de lo contrario, Python no lo reconocerá como tupla. Es posible crear tuplas vacías, pero no son de mucha utilidad.

[ ]:
```
print(type(()))
```

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

[ ]:
```
print(type((1)))
```

```
<class 'int'>
```

[ ]:
```
print(type((1,)))
```

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

0.1.1 Inmutabilidad

Las tuplas son inmodificables, lo que significa que no podemos cambiar, añadir o eliminar elementos después de la creación de la tupla. Sin embargo podemos acceder a los datos internos de ella usando la notación de corchetes `[ ]`

[ ]:
```
print(tupla[1])
```

```
1234
```

[ ]:
```
tupla[1] = 5432
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-ed49bfb97abe> in <module>()
----> 1 tupla[1] = 5432

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

[ ]:
```
del tupla[1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-093aa513d8c6> in <module>()
----> 1 del tupla[1]

TypeError: 'tuple' object doesn't support item deletion
```

0.1.2 Iteración en tuplas

Podemos recorrer las tuplas con ciclos del mismo modo que se recorren las listas.

[ ]:
```
%%debug_cell_with_pytutor
#creemos una tupla
tupla = (1,2,3,4,5,9,6,4,7,11,55,24,85,34)
#recorramos con range
for i in range(len(tupla)):
    print(tupla[i], end=" ")
```

[ ]:
```
%%debug_cell_with_pytutor
#creemos una tupla
tupla = (1,2,3,4,5,9,6,4,7,11,55,24,85,34)
#recorramos iterando directamente sobre la tupla
for i in tupla:
    print(i, end=" ")
```

```
1 2 3 4 5 9 6 4 7 11 55 24 85 34
```

0.1.3 ¿Para qué es útil una tupla?

De momento puede parecer que una tupla no tiene mucha utilidad si no se puede modificar su contenido, sin embargo, esta característica las hace brillar en dos de los temas que veremos más adelante, como claves de diccionarios ya que al no poderse modificar, podemos garantizar que un dato siempre sea representado por la misma clave y para retornar más de un resultado en una función, su uso más común. En este enlace discuten un poco más sobre su utilidad. De momento no nos preocuparemos por explicar los usos que mencionamos, ya que muy pronto entraremos más en detalle en ellos.

0.2 Sets

Los conjuntos o sets se utilizan para almacenar múltiples elementos en una sola variable. Un conjunto es una colección no ordenada, no modificable y no indexada, por lo tanto no es posible usar la notación de corchetes `[ ]` para acceder a su contenido.

> **Nota**: * Los elementos de un conjunto son inmutables, pero se pueden eliminar elementos y añadir otros nuevos. * Los conjuntos no están ordenados, por lo que no se puede asegurar en qué orden aparecerán los elementos. * Un set no puede almacenar listas u otros sets en su interior, si requerimos almacenar un paquete de datos en una sola posición, debemos usar tuplas

```
conjunto = {} # set vacío
conjunto = {1,3,'banana'}
```

[ ]:
```
conjunto = {1121,'a',85,(1,2,3),[45.51,85,6]}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-4ae67b010b1c> in <module>()
----> 1 conjunto = {1121,'a',85,(1,2,3),[45.51,85,6]}

TypeError: unhashable type: 'list'
```

[ ]:
```
conjunto = {1121,'a',85,(1,2,3),{45.51,85,6}}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-fce6b9b0cd4a> in <module>()
----> 1 conjunto = {1121,'a',85,(1,2,3),{45.51,85,6}}

TypeError: unhashable type: 'set'
```

[ ]:
```
conjunto = {1121,'a',85,(1,2,3),(45.51,85,6)}
print(type(conjunto),conjunto)
```

```
<class 'set'> {1121, (45.51, 85, 6), 'a', (1, 2, 3), 85}
```

0.2.1 No se permiten duplicados

Los conjuntos no pueden tener dos elementos con el mismo valor. Los duplicados simplemente se ignorarán

[ ]:
```
c = {1,2,3,1}
print(c)
```

```
{1, 2, 3}
```

0.2.2 Obtener la longitud de un set

Para determinar cuántos elementos tiene un conjunto, utilice la función `len()`.

[ ]:
```
print(f'{c} tiene {len(c)} elementos')
print(f'{conjunto} tiene {len(conjunto)} elementos')
```

```
{1, 2, 3} tiene 3 elementos
{1121, (45.51, 85, 6), 'a', (1, 2, 3), 85} tiene 5 elementos
```

###Añadir elementos a un conjunto #### Utilizando el método `add` Los elementos pueden ser añadidos al Conjunto utilizando la función incorporada `add()`. Sólo se puede añadir un elemento a

la vez al conjunto utilizando el método add(), los bucles se utilizan para añadir múltiples elementos a la vez con el uso del método `add()`.

[ ]:
```
%%debug_cell_with_pytutor
# Creamos un  set
set1 = set()
print("Set en blanco: ")
print(set1)

# Añadimos elementos y una tupla al set
set1.add(8)
set1.add(9)
set1.add((6,7))
print("\nSet luego de añadir 3 elementos: ")
print(set1)

# Añadimos elementos al set
# usando ciclos
for i in range(1, 6):
    set1.add(i)
print("\nSet luego de añadir los números del 1 al 5: ")
print(set1)
```

Uso del método `update` Para añadir dos o más elementos se utiliza el método `Update()`. El método `update()` acepta como argumentos listas, cadenas, tuplas y otros conjuntos. En todos estos casos, se evitan los elementos duplicados.

[ ]:
```
# Añadimos elementos al set
# usando la función update
set1 = set([ 4, 5, (6, 7)])
print('Set inicial\n',set1,sep="")
set1.update([10, 11])
print("\nSet luego de añadir elementos con update: ")
print(set1)
```

```
Set inicial
{4, 5, (6, 7)}

Set luego de añadir elementos con update:
{4, 5, (6, 7), 10, 11}
```

0.2.3 Eliminación de elementos del conjunto

####Utilizando el método `remove` o el método `discard` Los elementos pueden ser eliminados del conjunto utilizando la función incorporada `remove()` pero se produce un `KeyError` si el elemento no existe en el conjunto. Para eliminar elementos de un conjunto sin `KeyError`, utilice `discard()`, si el elemento no existe en el conjunto, permanece sin cambios.

[ ]:
```
# creamos un set
set1 = set([1, 2, 3, 4, 5, 6,
            7, 8, 9, 10, 11, 12])
print("Set inicial: ")
print(set1)
```

```
Set inicial:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

[ ]:
```
# Removemos elementos
# usando remove()
set1.remove(5)
set1.remove(6)
print("\nSet luego de eliminar 2 elementos: ")
print(set1)
```

```
Set luego de eliminar 2 elementos:
{1, 2, 3, 4, 7, 8, 9, 10, 11, 12}
```

[ ]:
```
#generemos un keyerror
set1.remove(15)
print(set1)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-8-5dbb28a3a77c> in <module>()
      1 #generemos un keyerror
----> 2 set1.remove(15)
      3 print(set1)

KeyError: 15
```

[ ]:
```
# usando discard()
set1.discard(8)
set1.discard(9)
print("\nSet luego de eliminar 2 elementos: ")
print(set1)
```

```
Set luego de eliminar 2 elementos:
{1, 2, 3, 4, 7, 10, 11, 12}
```

[ ]:
```
# usando un ciclo
for i in range(1, 5):
    set1.remove(i)
print("\nSet luego de eliminar elementos con un ciclo: ")
```

```
print(set1)
```

```
Set luego de eliminar elementos con un ciclo:
{7, 10, 11, 12}
```

0.2.4 Acceder a un conjunto

Como no se puede acceder a los elementos de un conjunto haciendo referencia a un índice, ya que los conjuntos no están ordenados y los elementos no tienen índice. Sin embargo, es posible recorrer los elementos del conjunto mediante un ciclo `for` o preguntar si un valor específico está incluído en un conjunto utilizando el operador `in`.

[ ]:
```
%%debug_cell_with_pytutor
# Creating a set
set1 = set() #set vacío
for i in range(0,100,10):
    set1.add(i)
print("set inicial")
print(set1)

# acceso mediante ciclo for
print("\nElementos del set: ")
for i in set1:
    print(i, end=" ")
print()
# Verificar si un elemento pertenece
#usando el operador in
print("10 en set?: ",10 in set1)
print("15 en set?: ",15 in set1)
```