


Python - Data Estructurada


PyLadies

Jupyter - Instalación

Ingresar a la ruta: <https://docs.conda.io/en/latest/miniconda.html>

 Conda
latest

[Conda](#)
[Conda-build](#)

 **Miniconda**

[Windows installers](#)
[MacOSX installers](#)
[Linux installers](#)
[Installing](#)
[Other resources](#)

[Help and support](#)
[Contributing](#)
[Conda license](#)

Windows installers

Windows

Python version	Name	Size	SHA256 hash
Python 3.7	Miniconda3 Windows 64-bit	51.5 MiB	<code>f18060cc0bb50ae75e4d602b7ce35197c8e31e81288d069b758594f1bb46ab45</code>
	Miniconda3 Windows 32-bit	54.0 MiB	<code>7c30778941d2bba03531ba269a78a108b01fa366530290376e7c3b467f3c66ba</code>
Python 2.7	Miniconda2 Windows 64-bit	50.9 MiB	<code>8647c54058f11842c37854edeff4d20bc1fbdad8b88d9d34d76fda1630e64846</code>
	Miniconda2 Windows 32-bit	48.7 MiB	<code>0d106228d6a4610b599df965dd6d9bb659329a17e3d693e3274b20291a7c6f94</code>

MacOSX installers

MacOSX

Python version	Name	Size	SHA256 hash
Python 3.7	Miniconda3 MacOSX 64-bit bash	49.4 MiB	<code>5cf91dde8f6024061c8b9239a1b4c34380238297adbdb9ef2061eb9d1a7f69bc</code>
	Miniconda3 MacOSX 64-bit pkg	59.8 MiB	<code>9927f1de5151a1a6431b02846fbca089e8b97a55a244f02ffc3207522092907b</code>
Python 2.7	Miniconda2 MacOSX 64-bit bash	39.4 MiB	<code>0db8f4037e40e13eb1d2adc89e054dfb165470cc77be45ef2bf9cb31c8b72f39</code>
	Miniconda2 MacOSX 64-bit pkg	47.8 MiB	<code>fcc30b2e18f7a292b34b2e24ad855786a66423f860157fa2b77e48b6392f0abb</code>

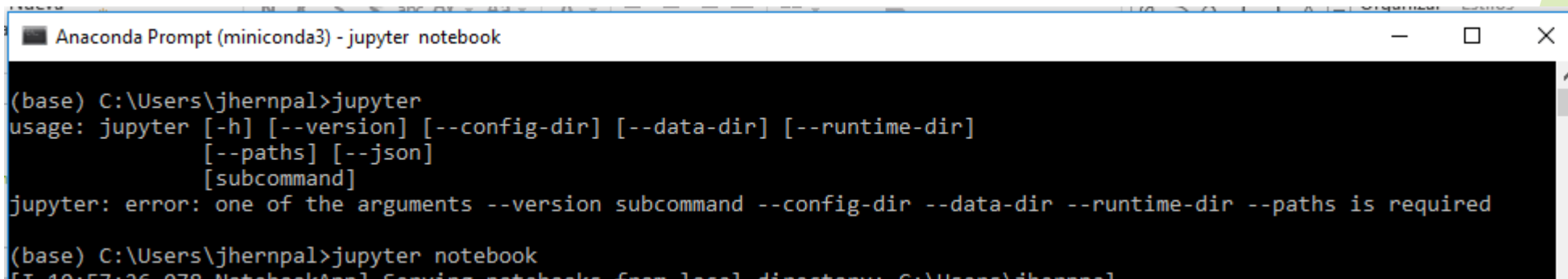
Jupyter - Instalación

Ingresar a consola de miniconda y ejecutar el comando:

“conda install jupyter”

Después de la instalación se deberá ejecutar el siguiente comando:

“jupyter notebook”



```
Anaconda Prompt (miniconda3) - jupyter notebook

(base) C:\Users\jhernpal>jupyter
usage: jupyter [-h] [--version] [--config-dir] [--data-dir] [--runtime-dir]
              [--paths] [--json]
              [subcommand]
jupyter: error: one of the arguments --version subcommand --config-dir --data-dir --runtime-dir --paths is required

(base) C:\Users\jhernpal>jupyter notebook
[E 10:57:26.078 NotebookApp] Serving notebooks from local directory: C:\Users\jhernpal
```

Tuple

Una tupla es una secuencia inmutable de longitud fija de objetos Python. La forma más fácil de crear uno es con una secuencia de valores separados por comas

```
In [1]: tup = 4, 5, 6
```

```
In [2]: tup
```

```
Out[2]: (4, 5, 6)
```

Tuple

Cuando define tuplas en expresiones más complicadas, a menudo es necesario encerrar los valores entre paréntesis, como en este ejemplo de creación de una tupla de tuplas:

```
In [3]: nested_tup = (4, 5, 6), (7, 8)
```

```
In [4]: nested_tup
```

```
Out[4]: ((4, 5, 6), (7, 8))
```

Tuple

Puede convertir cualquier secuencia o iterador en una tupla invocando la tupla:

```
In [5]: tuple([4, 0, 2])
```

```
Out[5]: (4, 0, 2)
```

```
In [6]: tup = tuple('string')
```

```
In [7]: tup
```

```
Out[7]: ('s', 't', 'r', 'i', 'n', 'g')
```

Tuple

Se puede acceder a los elementos entre corchetes [] como con la mayoría de los otros tipos de secuencia. Como en C, C ++, Java y muchos otros lenguajes, las secuencias están indexadas en 0 en Python:

```
In [8]: tup[0]  
Out[8]: 's'
```

Tuple

Una vez que se crea la tupla no es posible modificar qué objeto está almacenado en cada ranura:

```
In [9]: tup = tuple(['foo', [1, 2], True])

In [10]: tup[2] = False
-----
TypeError                                Traceback (most recent call last)
<ipython-input-10-b89d0c4ae599> in <module>()
----> 1 tup[2] = False
TypeError: 'tuple' object does not support item assignment
```


Tuple

Si un objeto dentro de una tupla es mutable, como una lista, puede modificarlo in situ:

```
In [11]: tup[1].append(3)
```

```
In [12]: tup
```

```
Out[12]: ('foo', [1, 2, 3], True)
```

Puede concatenar tuplas usando el operador + para producir tuplas más largas:

```
In [13]: (4, None, 'foo') + (6, 0) + ('bar',)
```

```
Out[13]: (4, None, 'foo', 6, 0, 'bar')
```

Tuple

Multiplicar una tupla por un número entero, como con las listas, tiene el efecto de concatenar juntas tantas copias de la tupla:

```
In [14]: ('foo', 'bar') * 4  
Out[14]: ('foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar')
```

UNPACKING TUPLES

Multiplicar una tupla por un número entero, como con las listas, tiene el efecto de concatenar juntas tantas copias de la tupla:

```
In [15]: tup = (4, 5, 6)
```

```
In [16]: a, b, c = tup
```

```
In [17]: b
```

```
Out[17]: 5
```

Incluso las secuencias con tuplas anidadas se pueden desempaquetar:

```
In [18]: tup = 4, 5, (6, 7)
```

```
In [19]: a, b, (c, d) = tup
```

```
In [20]: d
```

```
Out[20]: 7
```

UNPACKING TUPLES

Incluso las secuencias con tuplas anidadas se pueden desempaquetar:

```
In [18]: tup = 4, 5, (6, 7)
```

```
In [19]: a, b, (c, d) = tup
```

```
In [20]: d
```

```
Out[20]: 7
```

Con esta funcionalidad, puede intercambiar fácilmente nombres de variables, una tarea que en muchos idiomas podría verse así:

```
tmp = a  
a = b  
b = tmp
```

UNPACKING TUPLES

Pero, en Python, el intercambio se puede hacer así:

```
In [21]: a, b = 1, 2
```

```
In [22]: a
```

```
Out[22]: 1
```

```
In [23]: b
```

```
Out[23]: 2
```

```
In [24]: b, a = a, b
```

```
In [25]: a
```

```
Out[25]: 2
```

```
In [26]: b
```

```
Out[26]: 1
```

UNPACKING TUPLES

Un uso común del desempaqueado de variables es iterar sobre secuencias de tuplas o listas:

```
In [27]: seq = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]

In [28]: for a, b, c in seq:
.....:     print('a={0}, b={1}, c={2}'.format(a, b, c))
a=1, b=2, c=3
a=4, b=5, c=6
a=7, b=8, c=9
```

Métodos de la Tuplas

Dado que el tamaño y el contenido de una tupla no se pueden modificar, es muy ligero en los métodos de instancia. Uno particularmente útil (también disponible en las listas) es `count`, que cuenta el número de apariciones de un valor:

```
In [34]: a = (1, 2, 2, 2, 3, 4, 2)
```

```
In [35]: a.count(2)
```

```
Out[35]: 4
```

Lista: A diferencia de las tuplas, las listas son de longitud variable y su contenido se puede modificar en el lugar. Puede definirlos usando corchetes `[]` o usando la función de tipo de lista:

```
In [36]: a_list = [2, 3, 7, None]
```

```
In [37]: tup = ('foo', 'bar', 'baz')
```

```
In [38]: b_list = list(tup)
```

```
In [39]: b_list
```

```
Out[39]: ['foo', 'bar', 'baz']
```

```
In [40]: b_list[1] = 'peekaboo'
```

```
In [41]: b_list
```

```
Out[41]: ['foo', 'peekaboo', 'baz']
```

List

Las listas y las tuplas son semánticamente similares (aunque las tuplas no se pueden modificar) y se pueden usar indistintamente en muchas funciones.

La función de lista se usa con frecuencia en el procesamiento de datos como una forma de materializar un iterador o una expresión generadora:

```
In [42]: gen = range(10)

In [43]: gen
Out[43]: range(0, 10)

In [44]: list(gen)
Out[44]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```


Agregar y Remover Elementos

Los elementos se pueden agregar al final de la lista con el método `append`:

```
In [45]: b_list.append('dwarf')  
  
In [46]: b_list  
Out[46]: ['foo', 'peekaboo', 'baz', 'dwarf']
```

Usando `insertar` puede insertar un elemento en una ubicación específica en la lista:

```
In [47]: b_list.insert(1, 'red')  
  
In [48]: b_list  
Out[48]: ['foo', 'red', 'peekaboo', 'baz', 'dwarf']
```

Agregar y Remover Elementos

La operación inversa para insertar es pop, que elimina y devuelve un elemento en un índice particular:

```
In [49]: b_list.pop(2)
Out[49]: 'peekaboo'

In [50]: b_list
Out[50]: ['foo', 'red', 'baz', 'dwarf']
```

Los elementos se pueden eliminar por valor con remove, que localiza el primer valor y lo elimina del último:

```
In [51]: b_list.append('foo')

In [52]: b_list
Out[52]: ['foo', 'red', 'baz', 'dwarf', 'foo']

In [53]: b_list.remove('foo')

In [54]: b_list
Out[54]: ['red', 'baz', 'dwarf', 'foo']
```

Agregar y Remover Elementos

Si el rendimiento no es una preocupación, mediante el uso de agregar y eliminar, puede usar una lista de Python como una estructura de datos "multiset" perfectamente adecuada.

Compruebe si una lista contiene un valor con la palabra clave en:

```
In [55]: 'dwarf' in b_list  
Out[55]: True
```

La palabra clave no se puede utilizar para negar en:

```
In [56]: 'dwarf' not in b_list  
Out[56]: False
```

Concatenando y Combinando Listas

Similar a las tuplas, agregar dos listas junto con + las concatena:

```
In [57]: [4, None, 'foo'] + [7, 8, (2, 3)]  
Out[57]: [4, None, 'foo', 7, 8, (2, 3)]
```

Similar a las tuplas, agregar dos listas junto con + las concatena:

```
In [58]: x = [4, None, 'foo']  
  
In [59]: x.extend([7, 8, (2, 3)])  
  
In [60]: x  
Out[60]: [4, None, 'foo', 7, 8, (2, 3)]
```

Concatenando y Combinando Listas

Tenga en cuenta que la concatenación de listas por adición es una operación comparativamente costosa ya que se debe crear una nueva lista y copiar los objetos. Por lo general, es preferible utilizar `extend` para agregar elementos a una lista existente, especialmente si está creando una lista grande. Así:

```
everything = []  
for chunk in list_of_lists:  
    everything.extend(chunk)
```

Es más rápido que la alternativa concatenativa:

```
everything = []  
for chunk in list_of_lists:  
    everything = everything + chunk
```

Sorting

Puede ordenar una lista in situ (sin crear un nuevo objeto) llamando a su función de clasificación:

```
In [61]: a = [7, 2, 5, 1, 3]

In [62]: a.sort()

In [63]: a
Out[63]: [1, 2, 3, 5, 7]
```

Puede ordenar una lista in situ (sin crear un nuevo objeto) llamando a su función de clasificación:

```
In [61]: a = [7, 2, 5, 1, 3]

In [62]: a.sort()

In [63]: a
Out[63]: [1, 2, 3, 5, 7]
```

Sorting

Sort tiene algunas opciones que ocasionalmente serán útiles. Una es la capacidad de pasar una clave de clasificación secundaria, es decir, una función que produce un valor para clasificar los objetos. Por ejemplo, podríamos ordenar una colección de cadenas por su longitud:

```
In [64]: b = ['saw', 'small', 'He', 'foxes', 'six']
```

```
In [65]: b.sort(key=len)
```

```
In [66]: b
```

```
Out[66]: ['He', 'saw', 'six', 'small', 'foxes']
```

Bisect

Este módulo proporciona soporte para mantener una lista ordenada sin tener que ordenar la lista después de cada inserción. Para largas listas de artículos con costosas operaciones de comparación, esto puede ser una mejora con respecto al enfoque más común. El módulo se llama bisect porque usa un algoritmo de bisección básico para hacer su trabajo. El código fuente puede ser más útil como ejemplo de trabajo del algoritmo (¡las condiciones límite ya son correctas!).

```
In [67]: import bisect

In [68]: c = [1, 2, 2, 2, 3, 4, 7]

In [69]: bisect.bisect(c, 2)
Out[69]: 4

In [70]: bisect.bisect(c, 5)
Out[70]: 6

In [71]: bisect.insort(c, 6)

In [72]: c
Out[72]: [1, 2, 2, 2, 3, 4, 6, 7]
```


Slicing

Puede seleccionar secciones de la mayoría de los tipos de secuencia utilizando la notación de corte, que en su forma básica consiste en start: stop pasado al operador de indexación []:

```
In [73]: seq = [7, 2, 3, 7, 5, 6, 0, 1]
```

```
In [74]: seq[1:5]
```

```
Out[74]: [2, 3, 7, 5]
```

Los sectores también se pueden asignar con una secuencia:

```
In [75]: seq[3:4] = [6, 3]
```

```
In [76]: seq
```

```
Out[76]: [7, 2, 3, 6, 3, 5, 6, 0, 1]
```

Slicing

Si bien se incluye el elemento en el índice de inicio, el índice de detención no se incluye, por lo que el número de elementos en el resultado es stop - start.

Se puede omitir el inicio o la detención, en cuyo caso se establece de manera predeterminada el inicio de la secuencia y el final de la secuencia, respectivamente:

```
In [77]: seq[:5]  
Out[77]: [7, 2, 3, 6, 3]
```

```
In [78]: seq[3:]  
Out[78]: [6, 3, 5, 6, 0, 1]
```

Los índices negativos cortan la secuencia en relación con el final:

```
In [79]: seq[-4:]  
Out[79]: [5, 6, 0, 1]
```

```
In [80]: seq[-6:-2]  
Out[80]: [6, 3, 5, 6]
```

Slicing

Un paso también se puede usar después de un segundo colon para, por ejemplo, tomar cualquier otro elemento:

```
In [81]: seq[::2]  
Out[81]: [7, 3, 3, 6, 1]
```

Un uso inteligente de esto es pasar -1, que tiene el efecto útil de revertir una lista o tupla:

```
In [82]: seq[::-1]  
Out[82]: [1, 0, 6, 5, 3, 6, 3, 2, 7]
```

Slicing

⁰	¹	²	³	⁴	⁵
H	E	L	L	O	!

0 1 2 3 4 5 6
-6 -5 -4 -3 -2 -1

⁰	¹	²	³	⁴	⁵
H	E	L	L	O	!

`string[2:4]`

⁰	¹	²	³	⁴	⁵
H	E	L	L	O	!

`string[-5:-2]`

Condicional if

La sentencia condicional if se usa para tomar decisiones, este evalúa básicamente una operación lógica, es decir una expresión que de como resultado True o False, y ejecuta la pieza de código siguiente siempre y cuando el resultado sea verdadero.

```
numero = 10

if numero < 0:
    numero = 0
    print ('El número ingresado es negativo cambiado a cero.\n')
elif numero == 0:
    print ('El número ingresado es 0.\n')
elif numero == 1:
    print ('El número ingresado es 1.\n')
else:
    print ('El número ingresado es mayor que uno.\n')
```

Condicional If

Expresión ==

Esta expresión usa el operador == para validar la misma.

Expresión is

Esta expresión usa el operador is para validar la misma.

Expresión in

Esta expresión usa el operador in para validar la misma.

Ejemplos

```
In [36]: dato1, dato2, dato3, dato4 = 21, 10, 5, 20;
```

```
In [38]: if (dato1 == dato2):  
    print ('dato1 y dato2 son iguales.')  
else:  
    print ('dato1 y dato2 no son iguales.')
```

'dato1' y 'dato2' no son iguales.

```
In [39]: if (dato1 != dato2):  
    print ('dato1 y dato2 son distintas.')  
else:  
    print ('dato1 y dato2 no son distintas.')
```

'dato1' y 'dato2' son distintas.

Ejemplos

```
In [42]: if (dato1 < dato2):  
        print ("'dato1' es menor que 'dato2'.")  
    else:  
        print ("'dato1' no es menor que 'dato2'.")
```

'dato1' no es menor que 'dato2'.

```
In [43]: if (dato1 > dato2):  
        print ("'dato1' es mayor que 'dato2'.")  
    else:  
        print ("'dato1' no es mayor que 'dato2'.")
```

'dato1' es mayor que 'dato2'.

```
In [44]: if (dato3 <= dato4):  
        print ("'dato3' es menor o igual que 'dato4'.")  
    else:  
        print ("'dato3' no es menor o igual que 'dato4'.")
```

'dato3' es menor o igual que 'dato4'.

Ejemplo

```
In [45]: if (dato4 >= dato3):  
        print ("'dato4' es mayor o igual que 'dato3'.")  
    else:  
        print ("'dato4' no es mayor o igual que 'dato3'.")
```

'dato4' es mayor o igual que 'dato3'.