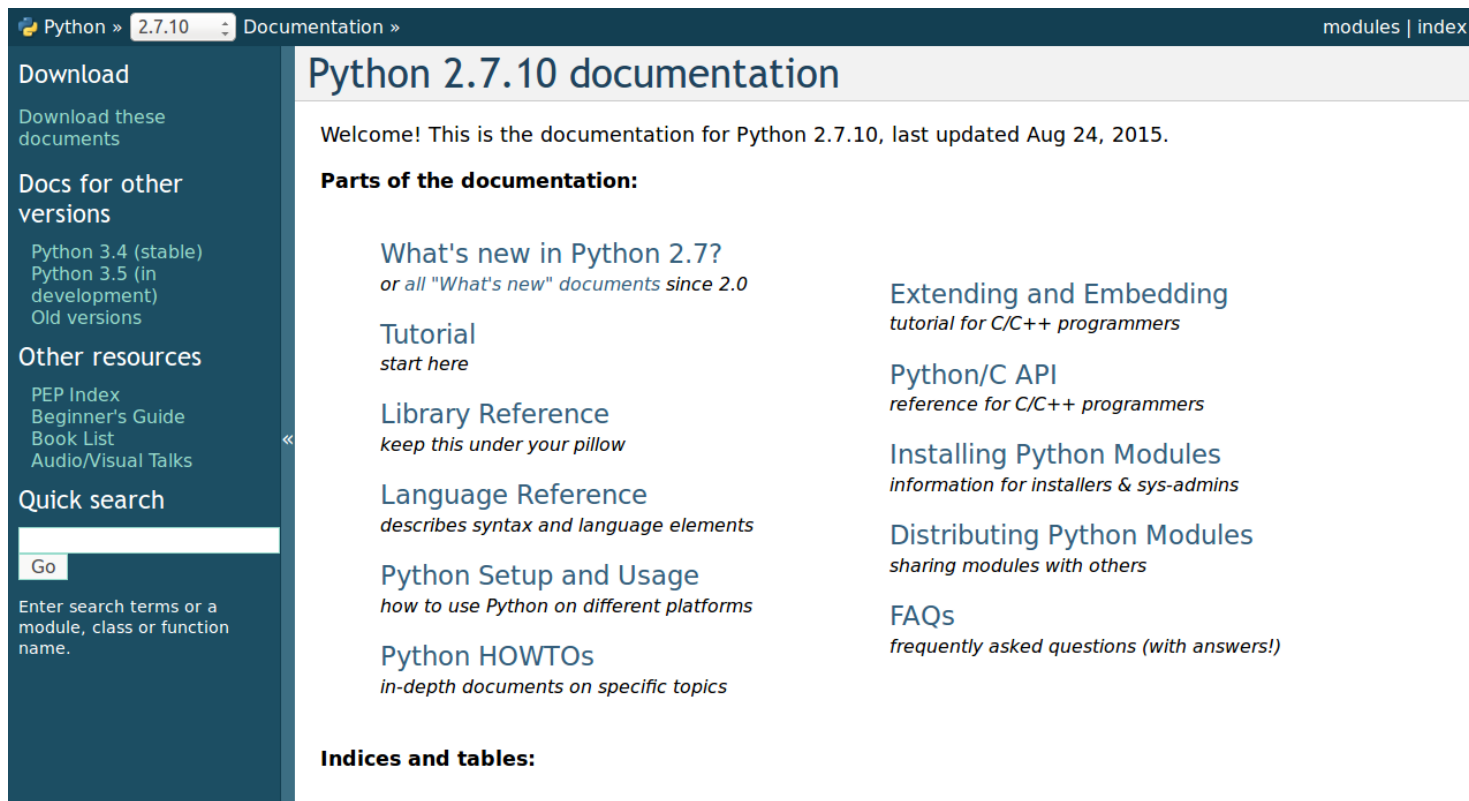


# **Python:**

# **Listas y Strings**

Herramientas Computacionales  
fl.gomez10 at uniandes.edu.co  
2015-2

# Documentación de python



The screenshot shows the Python 2.7.10 documentation page. The top navigation bar includes 'Python » 2.7.10' and 'Documentation »' with links for 'modules' and 'index'. A left sidebar contains links for 'Download', 'Docs for other versions' (including Python 3.4, 3.5, and old versions), 'Other resources' (PEP Index, Beginner's Guide, Book List, Audio/Visual Talks), and a 'Quick search' box with a 'Go' button. The main content area is titled 'Python 2.7.10 documentation' and includes a welcome message, a list of documentation parts, and a section for indices and tables.

Python » 2.7.10 Documentation » modules | index

## Python 2.7.10 documentation

Welcome! This is the documentation for Python 2.7.10, last updated Aug 24, 2015.

**Parts of the documentation:**

- [What's new in Python 2.7?](#)  
*or all "What's new" documents since 2.0*
- [Tutorial](#)  
*start here*
- [Library Reference](#)  
*keep this under your pillow*
- [Language Reference](#)  
*describes syntax and language elements*
- [Python Setup and Usage](#)  
*how to use Python on different platforms*
- [Python HOWTOs](#)  
*in-depth documents on specific topics*
- [Extending and Embedding](#)  
*tutorial for C/C++ programmers*
- [Python/C API](#)  
*reference for C/C++ programmers*
- [Installing Python Modules](#)  
*information for installers & sys-admins*
- [Distributing Python Modules](#)  
*sharing modules with others*
- [FAQs](#)  
*frequently asked questions (with answers!)*

**Indices and tables:**

<https://docs.python.org>

# Listas

Def: Una lista es un tipo de secuencia cambiante de python. (Mutable Sequence Type)

Documentación oficial disponible en

<https://docs.python.org/2/library/stdtypes.html#mutable-sequence-types>

<https://docs.python.org/2/library/functions.html#list>

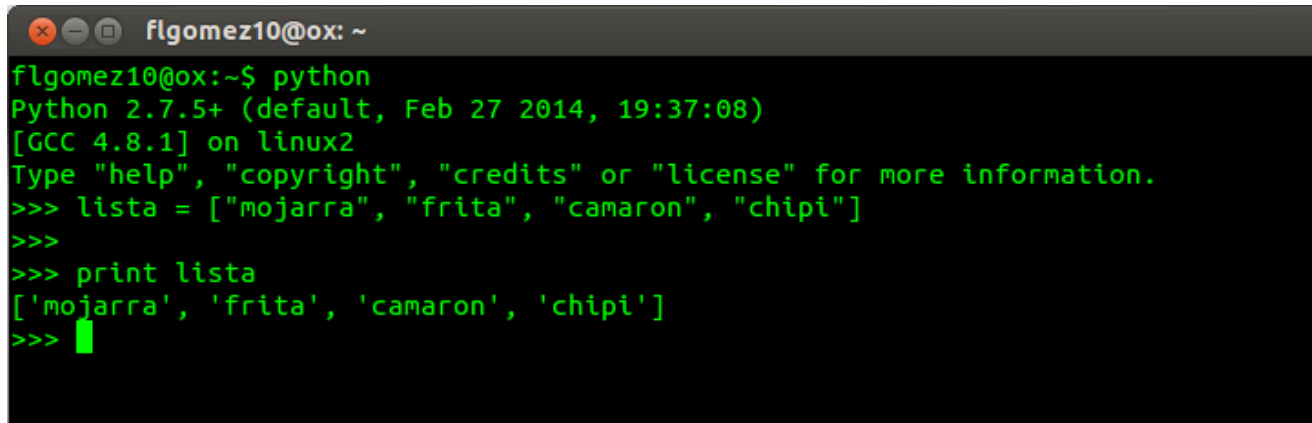
# Operaciones sobre listas

Operation	Result
<code>s[i] = x</code>	item <i>i</i> of <i>s</i> is replaced by <i>x</i>
<code>s[i:j] = t</code>	slice of <i>s</i> from <i>i</i> to <i>j</i> is replaced by the contents of the iterable <i>t</i>
<code>del s[i:j]</code>	same as <code>s[i:j] = []</code>
<code>s[i:j:k] = t</code>	the elements of <code>s[i:j:k]</code> are replaced by those of <i>t</i>
<code>del s[i:j:k]</code>	removes the elements of <code>s[i:j:k]</code> from the list
<code>s.append(x)</code>	same as <code>s[len(s):len(s)] = [x]</code>
<code>s.extend(x)</code>	same as <code>s[len(s):len(s)] = x</code>
<code>s.count(x)</code>	return number of <i>i</i> 's for which <code>s[i] == x</code>
<code>s.index(x[, i[, j]])</code>	return smallest <i>k</i> such that <code>s[k] == x</code> and <code>i &lt;= k &lt; j</code>
<code>s.insert(i, x)</code>	same as <code>s[i:i] = [x]</code>
<code>s.pop([i])</code>	same as <code>x = s[i]; del s[i]; return x</code>
<code>s.remove(x)</code>	same as <code>del s[s.index(x)]</code>
<code>s.reverse()</code>	reverses the items of <i>s</i> in place
<code>s.sort([cmp[, key[, reverse]]])</code>	sort the items of <i>s</i> in place

# Creación de listas

Crearemos una lista con los siguientes elementos:

mojarra, frita, camaron, chipi

A terminal window with a dark background and green text. The window title is 'flgomez10@ox: ~'. The text inside shows a Python prompt where a list named 'lista' is created with the elements 'mojarra', 'frita', 'camaron', and 'chipi'. The list is then printed, showing the output as ['mojarra', 'frita', 'camaron', 'chipi'].

```
flgomez10@ox: ~  
flgomez10@ox:~$ python  
Python 2.7.5+ (default, Feb 27 2014, 19:37:08)  
[GCC 4.8.1] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> lista = ["mojarra", "frita", "camaron", "chipi"]  
>>>  
>>> print lista  
['mojarra', 'frita', 'camaron', 'chipi']  
>>> 
```

# Ver elementos de la lista

```
flgomez10@ox: ~  
>>> lista = ["mojarra", "frita", "camaron", "chipi"]  
>>>  
>>> print lista  
['mojarra', 'frita', 'camaron', 'chipi']  
>>> # Para saber cuantos elementos tiene la lista utilizamos la funcion  
... # len().  
...  
>>> len(lista)  
4  
>>> # Voila! tenemos cuatro elementos.  
... # Recordando que en python contamos desde cero, El primer elemento es lista[  
0]  
...  
>>> lista[0]  
'mojarra'  
>>> lista[1]  
'frita'  
>>> lista[2]  
'camaron'  
>>> lista[3]  
'chipi'  
>>>  
>>>  
>>>
```

# Ver elementos de la lista

```
flgomez10@ox: ~  
>>> # Podemos usar indices negativos en una lista  
... # para llamar desde atras hacia adelante.  
...  
>>> lista = ["mojarra", "frita", "camaron", "chipi"]  
>>>  
>>> lista[-1]  
'chipi'  
>>> lista[-3]  
'frita'  
>>> lista[-4]  
'mojarra'  
>>> █
```

# “Slicing” o particionado

```
flgomez10@ox: ~  
>>> # Para imprimir solo un rango de elementos utilizamos ":"  
... # por ejemplo, para imprimir los tres primeros elementos  
... # de nuestra lista escribimos.  
...  
>>> lista[:3]  
['mojarra', 'frita', 'camaron']  
>>>  
>>> # Para ver desde el elemento 2 en adelante escribimos:  
...  
>>> lista[2:]  
['camaron', 'chipi']  
>>>  
>>> # Para imprimir todos los elementos de la lista:  
...  
>>> lista[:]  
['mojarra', 'frita', 'camaron', 'chipi']  
>>>  
>>> # y para imprimir desde el elemento 1 hasta el 3:  
...  
>>> lista[1:3]  
['frita', 'camaron']  
>>>  
>>> █
```



# “Slicing” o particionado

```
flgomez10@ox: ~  
>>> x = ["a","b","c","d","e","f","g","h","i","j","k"]  
>>>  
>>> x[:]  
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k']  
>>>  
>>> x[3:]  
['d', 'e', 'f', 'g', 'h', 'i', 'j', 'k']  
>>>  
>>> x[:3]  
['a', 'b', 'c']  
>>>  
>>> x[3:6]  
['d', 'e', 'f']  
>>>
```

# “Slicing” o particionado

```
flgomez10@ox: ~  
>>> # Imprimir saltando de dos en dos  
...  
>>> x[::2]  
['a', 'c', 'e', 'g', 'i', 'k']  
>>>  
>>> # imprimir desde el elemento 3 hasta el 9 saltando de 2 en dos  
...  
>>> x[3:9:2]  
['d', 'f', 'h']  
>>>  
>>> # imprimir desde atrás hacia adelante  
...  
>>> x[::-1]  
['k', 'j', 'i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']  
>>>  
>>> x[-3:-9:-2]  
['i', 'g', 'e']  
>>>
```

# Añadir elementos

```
flgomez10@ox: ~  
>>> # Para añadir un elemento al final de la lista  
... # usamos el método append  
...  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi']  
>>>  
>>> lista.append("chipi")  
>>>  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'chipi']  
>>> █
```

# Añadir elementos

```
flgomez10@ox: ~  
>>> # Para añadir un elemento al final de la lista  
... # usamos el método append  
...  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi']  
>>>  
>>> lista.append("chipi")  
>>>  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'chipi']  
>>>  
>>> # Podemos extender nuestra lista añadiendo  
... # varios elementos a la vez con el método  
... # extend  
...  
>>> lista.extend(["y", "mi", "caldero", "de", "arroz"])  
>>>  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'chipi', 'y', 'mi', 'caldero', 'de', 'a  
rroz']  
>>> █
```

# Añadir elementos

```
flgomez10@ox: ~  
>>> # También podemos insertar elementos en una posición específica  
...  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'chipi', 'y', 'mi', 'caldero', 'de', 'a  
rroz']  
>>> lista.insert(4, "seriedad")  
>>>  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'seriedad', 'chipi', 'y', 'mi', 'calder  
o', 'de', 'arroz']  
>>> █
```

# Buscar elementos

```
flgomez10@ox: ~  
>>> # El método index busca la primera posición donde aparezca  
... # nuestro criterio de búsqueda.  
...  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'seriedad', 'chipi', 'y', 'mi', 'calder  
o', 'de', 'arroz']  
>>>  
>>> lista.index("frita")  
1  
>>> lista.index("chipi")  
3  
>>>  
>>> # Y si no tenemos dicho criterio de búsqueda en nuestra lista,  
... # devuelve un error.  
...  
>>> lista.index("pulpo")  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ValueError: 'pulpo' is not in list  
>>>  
>>> # u__u  
...  
>>> █
```

# Buscar elementos

```
flgomez10@ox: ~  
>>> # Es mejor tener una respuesta lógica a la pregunta  
... # "¿está este elemento en la lista?"  
... # que tener un error.  
... # Para eso usamos "in".  
...  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'seriedad', 'chipi', 'y', 'mi', 'calder  
o', 'de', 'arroz']  
>>>  
>>> "pulpo" in lista  
False  
>>> "camaron" in lista  
True  
>>> █
```

# Eliminar elementos

```
flgomez10@ox: ~  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'seriedad', 'chipi', 'y', 'mi', 'caldero', 'de', 'arroz']  
>>> lista.remove("seriedad")  
>>>  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'chipi', 'y', 'mi', 'caldero', 'de', 'arroz']  
>>>  
>>> # Cuando se tienen elementos repetidos, el método remove  
... # elimina la primera entrada que se tenga.  
...  
>>> lista.remove("chipi")  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'y', 'mi', 'caldero', 'de', 'arroz']  
>>>  
>>> # y si no existe el elemento a eliminar, arroja un error.  
...  
>>> lista.remove("pulpo")  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ValueError: list.remove(x): x not in list  
>>> █
```



# Eliminar elementos

```
flgomez10@ox: ~  
>>> # El método "pop" requiere como argumento un número entero.  
... # pop(n) elimina el n-ésimo elemento de la lista y lo  
... # devuelve, puede verse en pantalla o asignarse a otra variable.  
...  
>>> lista  
['mojarra', 'frita', 'camaron', 'chipi', 'y', 'mi', 'caldero', 'de', 'arroz']  
>>>  
>>> lista.pop(3)  
'chipi'  
>>>  
>>> lista  
['mojarra', 'frita', 'camaron', 'y', 'mi', 'caldero', 'de', 'arroz']  
>>>  
>>> x = lista.pop(2)  
>>>  
>>> lista  
['mojarra', 'frita', 'y', 'mi', 'caldero', 'de', 'arroz']  
>>>  
>>> x  
'camaron'  
>>> █
```

# Eliminar elementos

```
flgomez10@ox: ~  
>>> # Si usamos pop() sin argumentos, elimina el último elemento de la lista.  
...  
>>> lista  
['mojarra', 'frita', 'y', 'mi', 'caldero', 'de', 'arroz']  
>>>  
>>> lista.pop()  
'arroz'  
>>>  
>>> lista  
['mojarra', 'frita', 'y', 'mi', 'caldero', 'de']  
>>> █
```

# Método “count”

```
flgomez10@ox: ~  
>>> # Ver los diferentes elementos que tiene la lista  
... # sin tener en cuenta las repeticiones.  
...  
>>> x = ["a","b","c","d","a","f","k","z","a","b","s"]  
>>>  
>>> set(x)  
set(['a', 'c', 'b', 'd', 'f', 'k', 's', 'z'])  
>>>  
>>> x.count("a")  
3  
>>> x.count("z")  
1  
>>> # Si nuestro criterio de búsqueda no está en la lista  
...  
>>> x.count("pulpo")  
0  
>>> █
```

# Funciones max() y min()

```
flgomez10@ox: ~  
>>> x  
['a', 'b', 'c', 'd', 'a', 'f', 'k', 'z', 'a', 'b', 's']  
>>>  
>>> max(x)  
'z'  
>>>  
>>> min(x)  
'a'  
>>> █
```

# Suma y repetición de listas

```
flgomez10@ox: ~  
>>> # Suma de listas  
...  
>>> x = ["a","b","c"]  
>>>  
>>> y = ["d","e","f","g"]  
>>>  
>>> x + y  
['a', 'b', 'c', 'd', 'e', 'f', 'g']  
>>>  
>>> # repetición de listas  
...  
>>> 3*x  
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']  
>>>  
>>> x*3  
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']  
>>>  
>>> 3*x + 2*y  
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'd', 'e', 'f',  
 'g']  
>>>
```

# Suma y repetición de listas

flgomez10@ox: ~

```
>>> x = ["a","b","c"]
```

```
>>>
```

```
>>> x/3
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

```
>>>
```

```
>>> # No está definida la división de listas u__u
```

```
...
```

```
>>> x - y
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

```
>>>
```

```
>>> # tampoco la resta de listas
```

```
...
```

```
>>> █
```

Fin de la brevísima introducción a las listas en Python.

¡Ahora strings y tuplas!

# Listas, strings y tuplas

```
flgomez10@ox: ~  
>>> # La función type() nos permite saber qué tipo de variable tenemos.  
... # definiremos una lista, una cadena de caracteres (string) y  
... # una tupla  
...  
>>> desayuno = ["leche", "pan", "huevos", "mermelada", "queso"]  
>>> adjetivo = "supercalifragilisticoespialidoso"  
>>> notas = (3.5, 4.3, 5.0, 4.8)  
>>>  
>>> type(desayuno)  
<type 'list'>  
>>>  
>>> type(adjetivo)  
<type 'str'>  
>>>  
>>> type(notas)  
<type 'tuple'>  
>>> # Una tupla es una lista INMUTABLE, no puede modificarse después  
... # de su creación.  
... # u_u  
... █
```



# Listas, strings y tuplas

```
flgomez10@ox: ~  
>>> # Funcionan como listas  
...  
>>> desayuno[0]  
'leche'  
>>> adjetivo[0]  
's'  
>>> notas[0]  
3.5  
>>>  
>>> adjetivo + "bla bla bla bla"  
'supercalifragilisticoespialidosobla bla bla bla'  
>>> adjetivo[::-1]  
'osodilaipseocitsiligarfilacrepus'  
>>>  
>>> notas  
(3.5, 4.3, 5.0, 4.8)  
>>> notas[2:]  
(5.0, 4.8)  
>>> 
```

# Listas ↔ Tuplas

```
flgomez10@ox: ~  
>>> notas  
(3.5, 4.3, 5.0, 4.8)  
>>> type(notas)  
<type 'tuple'>  
>>> notas = list(notas)  
>>> type(notas)  
<type 'list'>  
>>> notas  
[3.5, 4.3, 5.0, 4.8]  
>>> # Ahora es modificable!  
...  
>>> notas.append(0.0)  
>>> notas  
[3.5, 4.3, 5.0, 4.8, 0.0]  
>>> notas = tuple(notas)  
>>> notas  
(3.5, 4.3, 5.0, 4.8, 0.0)  
>>> type(notas)  
<type 'tuple'>  
>>> # Ahora nunca jamás podrás olvidar ese cero, mua ja ja ja.  
...  
>>> █
```

# Operaciones con Strings

```
flgomez10@ox: ~  
>>> cadena = ""  
>>> # este es un string con cero caracteres. Vamos a añadirle cada elemento  
... # de la lista desayuno.  
...  
>>> for elemento in desayuno:  
...     cadena = cadena + elemento  
...  
>>> print cadena  
lechepanhuevosmermeladaqueso  
>>> #Lo intentaremos de nuevo, esta vez añadiendo espacio entre los elementos.  
...  
>>> cadena = ""  
>>> for elemento in desayuno:  
...     cadena = cadena + elemento + " "  
...  
>>> cadena  
'leche pan huevos mermelada queso '  
>>> █
```

# Operaciones con Strings

```
flgomez10@ox: ~  
>>> cadena  
'leche pan huevos mermelada queso '  
>>> # Podemos crear listas a partir de strings separando palabras con  
... # split  
...  
>>> type(cadena)  
<type 'str'>  
>>> l = cadena.split()  
>>>  
>>> l  
['leche', 'pan', 'huevos', 'mermelada', 'queso']  
>>>  
>>> type(l)  
<type 'list'>  
>>> █
```

Descargar con wget el poema  
“El Cuervo” de Edgar Allan Poe  
en formato .txt desde [http://xurl.  
es/the\\_raven\\_poe.txt](http://xurl.es/the_raven_poe.txt)

o en formato largo

[https://raw.githubusercontent.com/ComputoCienciasUniandes/HerramientasComputacionalesDatos/master/data/the\\_raven\\_poe.txt](https://raw.githubusercontent.com/ComputoCienciasUniandes/HerramientasComputacionalesDatos/master/data/the_raven_poe.txt)

```
flgomez10@ox: ~  
flgomez10@ox:~$ head the_raven_poe.txt  
The Raven  
  
by Edgar Allan Poe  
(published 1845)  
  
Once upon a midnight dreary, while I pondered, weak and weary,  
Over many a quaint and curious volume of forgotten lore,  
While I nodded, nearly napping, suddenly there came a tapping,  
As of some one gently rapping, rapping at my chamber door.  
flgomez10@ox:~$
```

flgomez10@ox: ~

```
flgomez10@ox:~$ python
```

```
Python 2.7.5+ (default, Feb 27 2014, 19:37:08)
```

```
[GCC 4.8.1] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> infile = open("the_raven_poe.txt", "r")
```

```
>>> lineas = infile.readlines()
```

```
>>> lineas[0]
```

```
'The Raven\n'
```

```
>>> lineas[7]
```

```
'Over many a quaint and curious volume of forgotten lore,\n'
```

```
>>> lineas[7].split()
```

```
['Over', 'many', 'a', 'quaint', 'and', 'curious', 'volume', 'of', 'forgotten', 'lore,']
```

```
>>>
```

Fin.

<https://youtu.be/I9t4XTOWtEo>