

# Toolkit

Workshop Visualización de Datos con Python

El objetivo de este documento es presentar los elementos que utilizaremos para el workshop de visualización de datos. Las herramientas se encuentran contenidas en un sistema de administración llamado **Anaconda**. El toolkit que se presenta es un sistema que permite generar documentación ordenada de manera afable, así como la mantención de nuestro sistema de trabajo. El sistema está centrado Anaconda, Python, Markdown, Jupyter y el Terminal. Cada herramienta presenta una función específica respecto al toolkit.

## Anaconda: nuestro sistema de administración

Anaconda es una distribución de código abierto, diseñada específicamente para tareas asociadas al análisis de datos y computación científica. El objetivo es el sintetizar y unificar el conjunto de buenas prácticas en la mantención, administración y desarrollo de los proyectos.

Anaconda presenta dos versiones para descargar:

1. Anaconda Distribution: Versión completa con una amplia variedad de paquetes de Python y R ya instalados. Pesa alrededor de 3GB. → Link. **Escojan la versión 3.6**
2. Miniconda: Versión reducida con las instalaciones básicas. Se recomienda instalar esta versión no solo dado lo reducido, también les ayudará a soltar los dedos con la instalación de módulos y generación de ambientes virtuales. → Link. **Escojan la versión 3.6.**
  - Si van a instalar miniconda, informen para asistirlos.

Anaconda presenta su versión desde la línea de comando, llamada **conda**. A lo largo del curso enfatizaremos el uso de la línea de comando. A continuación se presenta una lista de comandos que pueden ser útiles para la administración:

Comando	Objetivo
<code>conda info</code>	Verificar versión de anaconda
<code>conda install PAQUETE</code>	Instalar un paquete en anaconda
<code>conda list</code>	Enlistar todos los paquetes instalados en nuestra distribución
<code>conda remove PAQUETE</code>	Eliminar un paquete de nuestra distribución
<code>conda update PAQUETE</code>	Actualizar un paquete de nuestra distribución

## Python: nuestro motor

Python es un lenguaje por Guido van Rossum en 1991, con la intención de hacer un lenguaje con mínimas distracciones sintácticas. Dada la economización del lenguaje, en Python se prioriza la existencia de *una solución que debe ser universal*, esta solución se le conoce informalmente como **pythonic solution**.

Algunas características de Python:

# Toolkit

Workshop Visualización de Datos con Python

---

- Interpretado: Ejecuta las implementaciones del código de forma directa, sin necesidad de ser compilado de forma previa.
- Dinámico: La definición de la tipificación se realiza durante la ejecución y no durante compilación. Esto significa que el programador pasa menos tiempo definiendo aspectos abstractos de su código, y más tiempo en lo substancial.
- Orientado a objetos: Enfoca la tarea del programador a cómo manipular objetos por sobre acciones, y datos por sobre lógica. Esto va de la mano con la definición dinámica.
- De alto nivel: Busca abstraerse de los detalles del computador y se acerca más al lenguaje humano en la forma de procesar.

Todo lo anterior se resume en que Python reduce la barrera de asimilación, desarrollándose como un lenguaje afable y ameno. La filosofía de Python se puede entender a partir del *Zen de Python*:

- Hermoso es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad importa.
- Los casos especiales no son lo suficientemente especiales como para romper las reglas.
- La funcionalidad vence a la belleza.
- Los errores nunca deben ser silenciados.
- A menos que sean silenciados explícitamente.
- Frente a la ambigüedad, abstenerse de adivinar.
- Debe existir una -y preferentemente sólo una- forma obvia de hacer las cosas.
- Aún cuando esa forma no sea obvia, a menos que seas Holandés.
- Ahora es mejor que nunca.
- Aún cuando nunca casi siempre es mejor que **ahora mismo**.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, quizás sea una buena idea.
- Los namespaces son una gran idea, hagamos más de esos!

## Python 2.x y 3.x

Un contratiempo de Python es el tiempo que se ha demorado en declarar su versión 2.x como obsoleta. Esto ha generado que la comunidad de Python tenga que operar en dos variantes. El fin del 2.7 está programado para 2020, así que es una buena idea acostumbrarse al formato 3.x.

## Diccionario Python ↔ Español

- **Argument:** Valor asignado a una función o programa cuando se ejecuta. Se ocupa de forma intercambiable con el término *parameter*

# Toolkit

Workshop Visualización de Datos con Python

- **Assertion:** Una expresión que se asume como verdadera en un punto particular del programa. Por lo general se utilizan *assertions* para identificar errores de forma explícita (Esto se conoce como *programación defensiva*).
- **Assign:** Asociación de un valor con una *variable*
- **Body:** El cuerpo de una función o clase. Se define bajo la declaración

```
1 def funcion(x):  
2     cuerpo que ejecuta acciones en x
```

- **Boolean operators:** Operadores lógicos (*and*, *or* y *not*) que permite expandir la evaluación de argumentos. No confundir con *Boolean values* (*True* y *False*).
- **Case Sensitive:**
- **Case Insensitive:**
- **Comment:** Una nota dentro de un programa que no es interpretada por el lenguaje. Sirve como guía para el humano.
- **Compose:** Función compuesta que resulta de aplicar una función dentro de otra  $\rightsquigarrow f(g(x))$ .
- **Conditional Statement:** Una declaración dentro de un programa que puede ser o no ejecutada, dependiendo del resultado de una prueba. También se les conoce como expresiones booleanas.
- **Default Value:** Valor que se define para el parámetro de una función en caso no especificar explícitamente algo.
- **Delimiter** Carácter o lista de caracteres utilizados para separar valores individuales
- **Docstring:** String de documentación. Es la representación textual que se escribe dentro de un programa/clase/función en Python. Un *docstring* puede ser visualizado de forma interactiva en el programa, llamando `objeto.__doc__`. Esto, a diferencia de los comentarios, que no pueden ser visualizados de forma interactiva.
- **Dotted Notation:** Sintaxis de acceso en variados lenguajes para acceder a componentes dentro de objetos de la siguiente forma canónica  $\rightsquigarrow \text{objeto.componente}$ .
- **Floating point number:** Número que contiene una parte real y un decimal.  $\rightsquigarrow 5.6$
- **Immutable:** Dato que no se puede cambiar después de su creación.
- **Import:** Cargar una librería o módulo en un programa.
- **In place operators:** Un operador como `+=` que sirve como un atajo para operaciones comunes. Por ejemplo, si deseamos actualizar el valor de `x`:

```
1 # forma clásica  
2 x = x + 3  
3  
4 #forma express  
5 x += 3
```

- **Integer:** Número completo  $\rightsquigarrow 435$ .
- **Library:** Un conjunto de scripts (clases/funciones/variables) que implementan una serie de tareas relacionadas.
- **Method:** Una función asociada a un objeto específico.
- **Object:** Colección de variables asociadas a funciones específicas (métodos).

# Toolkit

Workshop Visualización de Datos con Python

---

- **Parameter:** Variable declarada dentro de la función que permite pasar el valor a la llamada de ésta.
- **Pipeline:** Conexiones realizadas entre el output de un programa al input de otro.
- **Return statement:** Declaración que ocasiona el fin de la ejecución de una función y retorna un valor de forma inmediata.
- **Slice:** Subsecuencia de un conjunto mayor
- **Stack frame:** La estructura de datos que provee asignación para las variables locales de una función. Cada vez que se invoca una función, se crea un nuevo stack frame y se prioriza en el call stack.
- **Standard input:** El flujo de elementos ingresados en una función.
- **Standard output:** El flujo de elementos salientes en una función.
- **String:** Secuencia de cero o más caracteres alfanuméricos. Si un número es ingresado en un string, no se interpreta como **int** o **float**.
- **Traceback:** La secuencia de llamadas de una función que conlleva a un error.
- **Tuple:** Secuencia inmutable de valores.
- **Type of Error:** Indica la naturaleza del error en el programa.

## Markdown: un formato afable de escritura

Es un lenguaje de marcas minimalista cuyo eje central es la conversión fácil a [html](#) y otros formatos<sup>1</sup>. Markdown permite escribir utilizando un lenguaje **fácil de leer y escribir**

Elemento	Sintaxis Markdown
# Título	Título principal
## Subtítulo	Subtítulo
### Subsubtítulo	Subsubtítulo
#### Párrafo	Párrafo
_Cursiva_ o *Negrita*	<i>cursiva</i>
__Negrita__ o **Negrita**	<b>Negrita</b>
[text](http://address)]	Inserción de hipervínculo
![alttext](/images/image.jpg)	Inserción de imagen

Las listas pueden ser ordenadas o no ordenadas. Para las listas no ordenadas, precedemos cada elemento con \*:

- Esta
- es
- una
- lista

Para las listas ordenadas, precedemos cada elemento con un número:

---

<sup>1</sup>Esto se puede lograr mediante pandoc

# Toolkit

Workshop Visualización de Datos con Python

---

1. Esta
2. Es
3. Otra
4. Lista

Se pueden agregar metadatos mediante un YAML<sup>2</sup>

## Jupyter: Python + Markdown = <3

Jupyter Notebook (antes denominado iPython Notebook) es un ambiente de trabajo que busca unificar las herramientas de documentación y ejecución de código en un mismo archivo. Los notebooks permite la lectura fácil para los humanos, así como la compartimentalización del código para la máquina.

Jupyter acepta dos tipos de *chunks* (códigos): un formato de texto, preferentemente Markdown; y un *kernel*, que es el motor computacional que vincula la interpretación del código en el chunk mediante un lenguaje.

Jupyter expandió su funcionalidad a más de cuarenta lenguajes, destacando Javascript, Ruby, R, Julia, Haskell, etc... Aprender su funcionalidad básica es una buena idea para escribir documentación y otros.

### Algunos tips para entender Jupyter

1. Jupyter funciona por chunks: Cada chunk puede contener texto **ó** código. Algunas funciones sobre los chunks. Cada función debe estar en el modo de comando (esto significa que deben presionar **Esc** antes de ejecutar)

Combinación de teclas	Acción
<Enter>	Entrar en el modo edición
<Shift> + <Enter>	Ejecutar chunk y crear nuevo chunk
<Control> + <Enter>	Ejecutar chunk, no moverse
<Tab>	Invocar autocompletado (ver todas las opciones)
<Shift> + <Tab>	Ver documentación sobre la función
M	Cambiar la modalidad del chunk de código a markdown
Y	Cambiar la modalidad del chunk de markdown a código
A	Insertar chunk arriba
B	Insertar chunk abajo
DD	Eliminar chunk

2. Dado que opera en base a chunks, su formato en sí es un poco ininteligible (es básicamente un `.json`)

---

<sup>2</sup>YAML Ain't Markup Language

# Toolkit

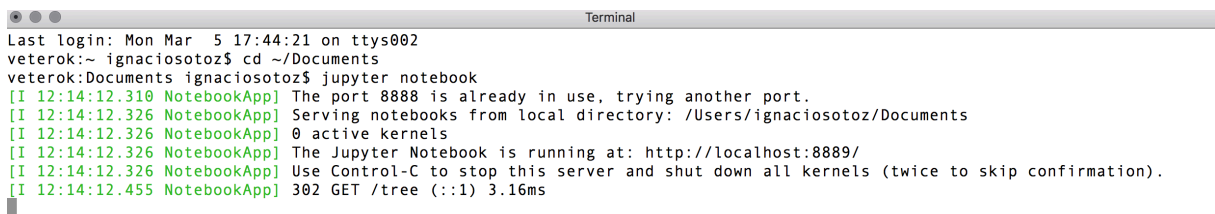
Workshop Visualización de Datos con Python

con extensión `.ipynb`). Hay algunas alternativas que pueden resultar útiles más adelante:

- Github soporta `.ipynb` y los visualiza en la página.
- Jupyter ofrece opciones para convertir `.ipynb` a `.html`, `.doc` y `.pdf`.

3. Jupyter funciona desde un navegador web por medio de un servidor. Es un buen consejo familiarizarse con los siguientes pasos:

- En el terminal, escriba `jupyter notebook`. En este paso está inicializando el servidor de manera local.
- Por lo general, `jupyter` abrirá el navegador web de forma automática. Si por X motivo el navegador se cierra; no se preocupe, jupyter sigue corriendo. Vuelva al navegador y diríjase a <http://localhost:8888/>



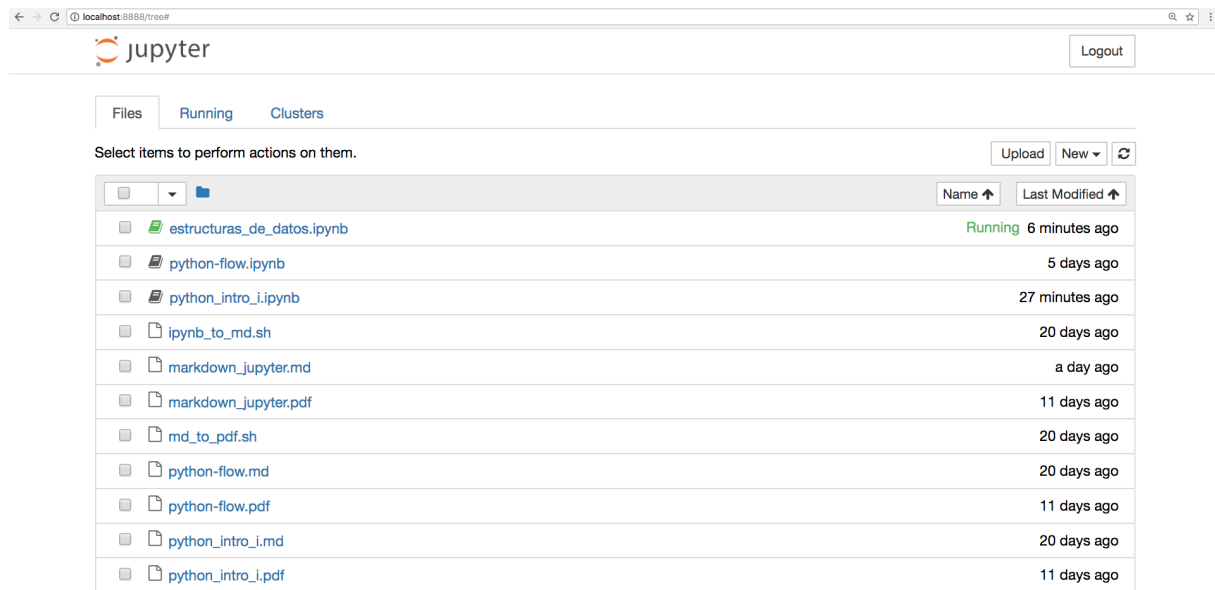
```
Terminal
Last login: Mon Mar  5 17:44:21 on ttys002
veterok:~ ignaciosotoz$ cd ~/Documents
veterok:Documents ignaciosotoz$ jupyter notebook
[I 12:14:12.310 NotebookApp] The port 8888 is already in use, trying another port.
[I 12:14:12.326 NotebookApp] Serving notebooks from local directory: /Users/ignaciosotoz/Documents
[I 12:14:12.326 NotebookApp] 0 active kernels
[I 12:14:12.326 NotebookApp] The Jupyter Notebook is running at: http://localhost:8889/
[I 12:14:12.326 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[I 12:14:12.455 NotebookApp] 302 GET /tree (::1) 3.16ms
```

**Figure 1:** Iniciando `jupyter notebook` desde el terminal

4. Dentro de `localhost:8888`, jupyter mostrará todos los documentos dentro del working directory.

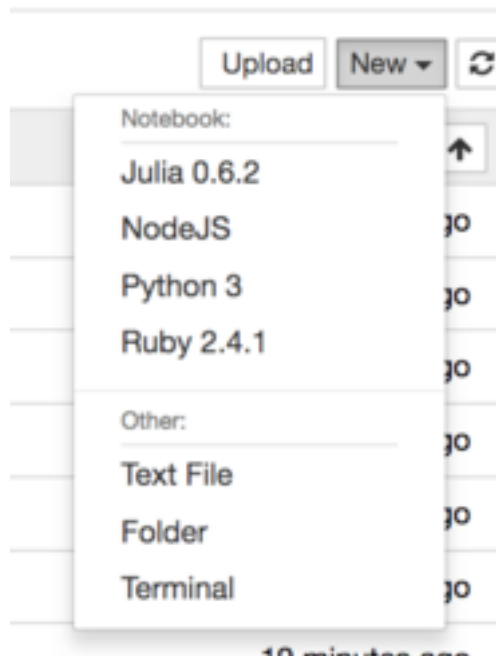
# Toolkit

Workshop Visualización de Datos con Python



**Figure 2:** Vista principal de la consola de Jupyter

5. Para iniciar un notebook, hacemos click en **New** y seleccionamos **Python 3**.



**Figure 3:** Creando un nuevo notebook



Figure 4: Vista de un notebook

6. Para ejecutar python dentro del notebook, procuramos que el costado izquierdo del chunk presente `In [ ]`. Una vez que el chunk sea ejecutado con `Shift + Enter`, obtendremos un número asociado dentro de `In [ ]` y un output con `Out [ ]`:

```
In [16]: 1 # Esta es un chunk dentro del kernel
        2
        3 class Person(object):
        4     def __init__(self, name):
        5         self.name = name
        6
        7
        8 Person

Out[16]: __main__.Person
```

Figure 5: Ejecución de Python dentro del notebook

7. Para ejecutar Markdown dentro del chunk, procuramos cambiar el modo. Si estamos en lo correcto, el contenido estará destacado siguiendo la sintaxis de Markdown.



# Toolkit

Workshop Visualización de Datos con Python

```
1 # Este es un chunk dentro de Markdown
2
3 * Podemos hacer listas:
4     - Y otras cosas dentro de las listas
5
6 * También podemos hacer __tablas__
7
8 |Lorem|Ipsum|
9 |-----|-----|
10 |Dolor|Sit|
11 |Amet|
```

Figure 6: Chunk de Markdown dentro del notebook

8. Ejecutando `Shift + Enter` obtendremos la versión HTML en el notebook.

## Este es un chunk dentro de Markdown

- Podemos hacer listas:
  - Y otras cosas dentro de las listas
- *También podemos hacer* **tablas**

Lorem	Ipsum
Dolor	Sit
Amet	

Figure 7: Chunk de Markdown renderizado

## Terminal: nuestra manera de interactuar con el computador

Addenda: A lo largo del curso utilizaremos los términos *línea de comando*, *consola*, *terminal* y *cli* de forma intercambiable.

El terminal es un medio de interacción con el computador donde el usuario envía comandos en forma de texto que deben ser interpretados por el computador. El terminal fue la forma más común de interactuar con el computador en sus inicios, en detrimento de las interfaces gráficas.

## Sí, genial. ¿Pero por qué debo utilizar una tecnología obsoleta?

1. Sumarás hackerpuntos: Tendrás mayor credibilidad en el círculo si dices que haces todo en el terminal (?)
2. Permite sistematizar y automatizar acciones frecuentes.
3. Uno decide qué hacer con el computador, no está restringido a las opciones disponibles en la interfaz gráfica.
4. Los servidores se siguen administrando via terminal. No escaparás de interactuar con un servidor.

## Algunos comandos básicos

- `pwd` ~> Ver el directorio en que me sitúo.
- `cd <DIRECTORIO>` ~> Cambiar al directorio detallado.
- `ls` ~> Listar los archivos del directorio en el que me sitúo.
- `ls -lah` ~> Listar todo los archivos del directorio, en formato humano y ordenado.
- `mkdir <DIRECTORIO>` ~> Crear un directorio.
- `touch <ARCHIVO>.<EXTENSION>` ~> Crear un archivo con nombre `archivo` y extensión `.extensión`.
- `rm <ARCHIVO>` ~> Eliminar archivos.