Objetivo de la guía.

El objetivo de esta guía es familiarizarnos con el uso de los cuadernos de Jupyter y con los elementos básicos de la programación utilizando el lenguaje Python.

Si bien la sintaxis de Python se ve en detalle más adelante, es nuestro interés familiarizarnos con el uso de esta herramienta que utilizaremos a lo largo del curso.

Jupyter.

Jupyter Notebook, es un entorno de trabajo Web, de código abierto que nos permite la inclusión de código, texto, audio, video e imágenes en un mismo archivo para la confección de documentos interactivos con código editable y ejecutable para múltiples propósitos, utilizando el navegador como soporte.

Jupyter refiera a una organización sin fines de lucro, PROJECT JUPYTER, que es la principal promotora de este entorno diseñado originalmente para proyectos en JULIA, PYTHON y R, (de ahí el nombre) y que con el paso del tiempo quedó con Python como lenguaje principal para trabajar en este tipo de proyectos, gracias a sus múltiples aplicaciones en el mundo del Big Data y la ciencia de datos.

Su principal objetivo es apoyar la ciencia de datos interactiva y la computación en casi todas las ramas de la ciencia y la ingeniería.

Jupyter Notebook.

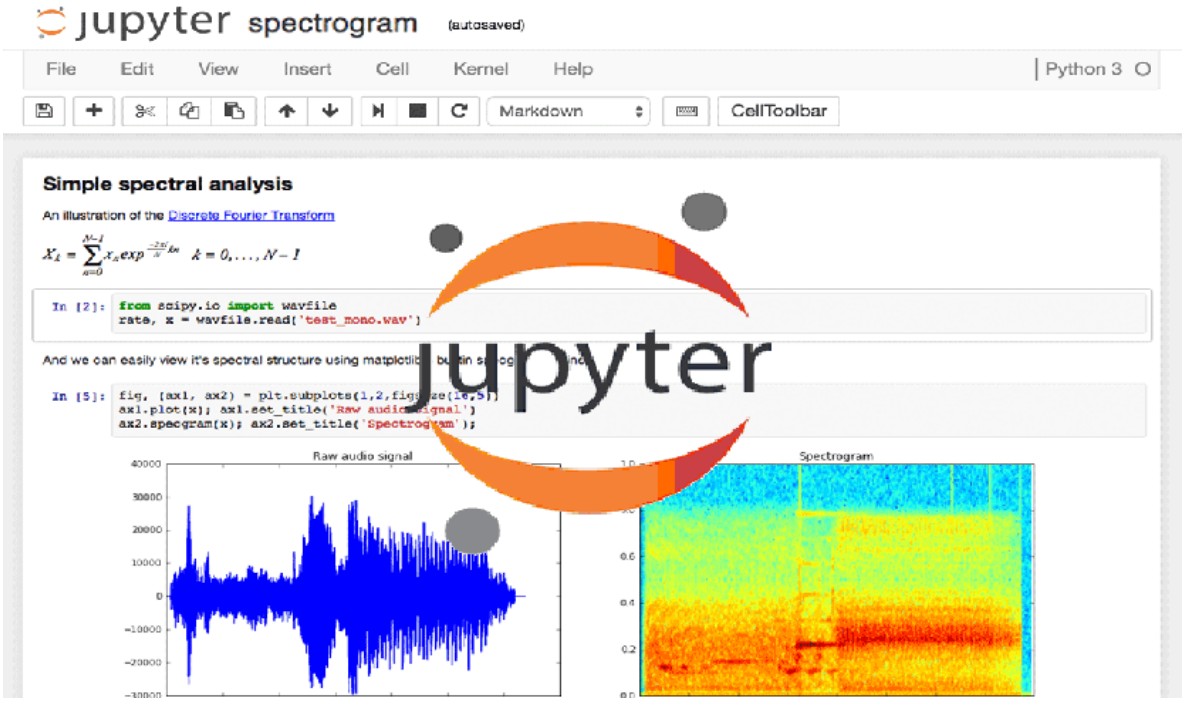
Un cuaderno de Jupyter es simplemente un archivo “ipynb” hecho por la aplicación web, el cual contiene una multitud de ‘celdas’ con código, texto Markdown (¡en formato LaTeX!), o metadatos raw.

Al ser interpretado por Jupyter, estas celdas terminan teniendo una apariencia muy similar a la de un documento de texto, y las celdas que contienen código son ejecutadas, mostrando la salida dentro de la celda.

Es muy común el uso de librerías como matplotlib para mostrar gráficos dentro de estas celdas de código, pero gracias a su versatilidad, puede ser utilizado también para mostrar videos o imágenes.

Es importante saber cuál es el objetivo de nuestro código, si vamos a hacer un SCRIPT en Python, que se deba ejecutar de manera local, por ejemplo, para cálculos predeterminados o automatización de tareas, la herramienta ideal es el IDLE que se descarga junto con la instalación.

En cambio, si nuestro código está destinado al trabajo científico, técnico, de ingeniería o si es de tipo colaborativo, trabajaremos directamente en la web de Jupyter, o en la versión local de Jupyter o directamente en el COLAB de Google.



Python.

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código.

Se trata de un lenguaje de programación multi paradigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

Introducción a Python usando Jupyter.

Veremos el uso básico de Jupyter para crear documentos interactivos mediante la programación en lenguaje Python.

Podemos utilizar la versión web del entorno https://jupyter.org/ o, si la tenemos instalada, usaremos dicha versión para trabajar con más comodidad.

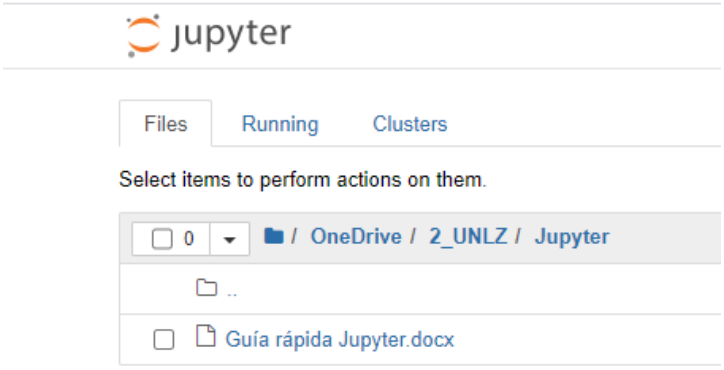
Dentro de estos documentos, estos cuadernos de Jupyter, podemos colocar texto, imágenes, videos y lo más importante, porciones de código para la solución y análisis de diferentes problemas.

Crear un cuaderno.

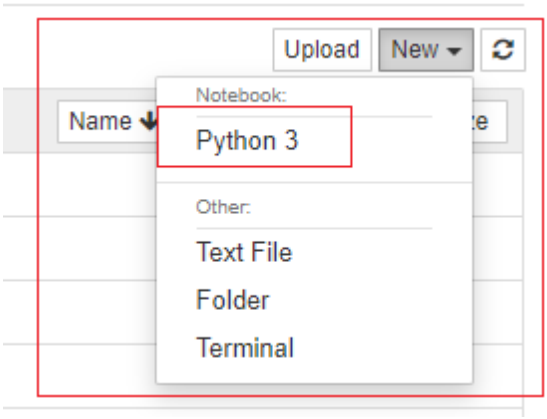
Apenas ejecutamos Jupyter Notebook se nos abre, en el navegador, la pantalla principal desde la cual podemos navegar entre las diferentes carpetas de nuestro equipo y crear nuevas carpetas o documentos.

El primer paso, es navegar hasta la carpeta en la que vayamos a crear

nuestro nuevo cuaderno.

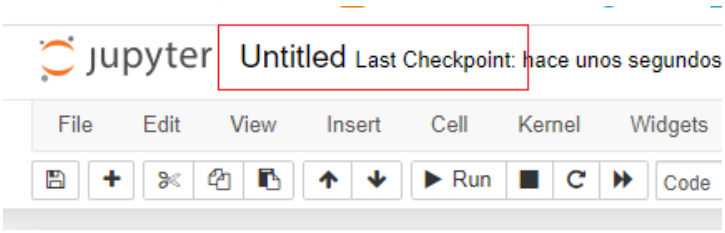


Una vez posicionados en la carpeta de trabajo, vamos a crear un nuevo cuaderno, basado en Python 3:



Se les abrirá una nueva ventana en el navegador web, con el cuaderno recién creado listo para usar.

El segundo paso es ponerle un título, pulsando con el puntero del mousse sobre la etiqueta del título:



Ya tenemos creado el cuaderno, lo hemos guardado en nuestra carpeta de trabajo y le pusimos un nombre.

Ya estamos en condiciones de empezar a conocer y aplicar los diferentes elementos del lenguaje.

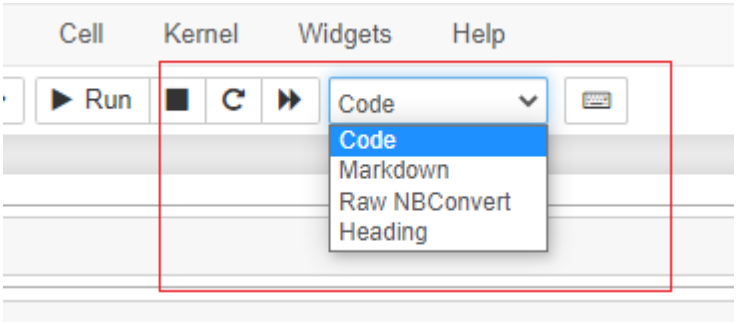
Celdas.

Las celdas son los cuadros que nos permiten ingresar los diferentes elementos que van a formar parte de nuestro documento.

Estos elementos serán en su mayoría código Python, pero también podemos poner texto, ecuaciones, gráficas, imágenes y videos.

Cada celda tiene a su izquierda una etiqueta "In [ ]" que nos indica entre corchetes el número de veces que fue ejecutado el código que contiene.

Las celdas pueden ser de diferentes tipos, por defecto, Jupyter siempre nos va a crear una celda para código, pero nosotros podemos cambiar el tipo de cada una desde el menú de opciones:



Texto.

Para colocar texto vamos a seleccionar el tipo de etiqueta "Markdown".

En las etiquetas de este tipo, podemos colocar texto en diferentes opciones, por ejemplo, encabezados, texto sencillo, listas, etc.

Veamos algunas opciones que podemos utilizar en nuestros cuadernos:

- Encabezados: ## Este es un título.
- Negrita: ** Este texto está en negrita **
- Cursiva: *Texto en cursiva*
- Listas sin numerar: * + espacio

- Listas numeradas: número + punto + espacio.
- LaTeX dentro de una línea de texto: $...$
- LaTeX separado de la línea de texto: $$.....$$

Elementos del lenguaje.

En este parte del curso trabajaremos directamente sobre una Notebook de Jupyter y para ello, usaremos la sintaxis básica del lenguaje, brevemente explicadas por el docente, con el único objetivo de hacer una pequeña introducción a temas que serán nuevamente explicados, pero más a fondo en su unidad correspondiente.

Esto nos sirve como introducción a la semántica del lenguaje y al uso de Jupyter, ya que es importante familiarizarnos con esta herramienta desde el inicio del curso porque la utilizaremos en todas las clases para practicas sobre los conceptos teóricos que vayamos incorporando.

Comentarios

En todo momento podemos incluir comentarios dentro del código que siempre serán ignorados por el intérprete.

Se utilizan, entre otras cosas, para documentar el paso a paso del trabajo, y van precedidos por el carácter #.

```
# Esto es un comentario.
# Recuerden que Python hace diferencia entre mayúsculas y minúsculas.
```

Es CASE SENSITIVE.

Cuando decimos que un lenguaje de programación es CASE SENSITIVE, nos referimos a que hace diferencia entre mayúsculas y minúsculas.

Esto es importante al momento de darle un nombre a nuestras variables y funciones.

Función PRINT.

La función PRINT nos permite mostrar por pantalla los diferentes datos con los que trabajamos en Python, por ejemplo, cadenas de texto, números, operaciones matemáticas, resultados devueltos por las funciones, etc.

```
# Función PRINT con texto.
print('Guía rápida de Python 3')

# función PRINT con un número.
print(56)
```

Variables.

Una variable es un depósito temporal de datos donde podremos guardar y tener a disposición todos los valores de nuestro código.

Para definir una variable, debemos asignarle un nombre (identificador) y un valor.

El nombre de una variable NO puede comenzar con un número.

No es necesario declarar el tipo de variable que queremos crear, esto se define al momento de asignarle un valor.

Podemos mostrar por pantalla el valor de las variables usando la función PRINT.

```
# Variables
edad = 34
apellido = 'Alfaro'

print(edad)
print(apellido)
```

```
34
Alfaro
```

Las variables se pueden reutilizar en el mismo código, es decir, un mismo nombre de variable lo puedo utilizar las veces que sea necesario incluso asignándole diferentes tipos de datos.

```
# Podemos reutilizar las variables
var1 = 12
print(var1)

var1 = 'Variable reutilizada'
print(var1)
```

```
12
Variable reutilizada
```

Aunque esto está permitido, se recomienda siempre dar a las variables, un nombre que nos ayude a recordar a simple vista que valor estamos guardando en ella.

Las variables en Python son, por defecto, variables locales.

Esto quiere decir que, si definimos una variable dentro de una porción de código, como, por ejemplo, una función, esta solo existirá dentro de la función y no serán accesibles desde otra parte del programa.

De igual manera, toda variable declarada fuera de una función, no será visible dentro de la misma.

Podemos ver el tipo de variable o tipo de dato, mediante la función TYPE.

Veamos como declarar y asignar valores a diferentes tipos de variable y mostrar por pantalla que tipo es.

```
# Función Type
# Me permite saber que tipo de dato tiene cargado una variable.

variable1 = 12
print(type(variable1))
```

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

Operadores aritméticos.

Estos son los operadores aritméticos que tenemos disponibles en Python.

Podemos probar los ejemplos directamente en el editor.

Operador	Nombre	Ejemplo	Resultado
+	Suma	15 + 20	35
-	Resta	15 - 7	8
-	Negativo	-5	-5
*	Multiplicación	8 * 4	32
**	Exponente	3 ** 2	9
/	División	24 / 3	8
//	División entera	17 // 2	7
%	Resto	29 % 3	2

Tipo de dato LISTA.

El tipo lista en Python me permite almacenar en su interior, diferentes tipos de datos, separados por coma.

Cada elemento de la lista tiene asociado un índice, que es la posición que ocupa dentro de la misma.

Por defecto, siempre el índice del primer elemento es el cero.

Este índice es que deberemos utilizar para referirnos a cada uno de los elementos cada vez que necesitemos acceder a ellos.

```
# Tipo de dato LISTA

numeros = [3,6,45,21,123,0,14,67]
print(numeros[4])
```

```
123
```

Dentro de una misma lista podemos tener diferentes tipos de datos, por ejemplo:

```
# Tipo de dato LISTA

numeros = [3,6,45,'Guillermo',123,0,'Alfaro',67]
print(numeros[3])
```

```
Guillermo
```

** Podemos hacer operaciones de todo tipo entre los elementos de una lista.

LEN

La función LEN nos permite conocer la longitud de una lista.

```
# Función LEN. Permite conocer el largo de una lista.
colores = ['rojo','verde','azul','celeste']
print('La lista tiene : ',len(colores),' colores.')
```

```
La lista tiene :  4  colores.
```

Métodos de las listas.

1- Método APPEND.
   a. Permite agregar un nuevo elemento al final de la lista.

```
# Método APPEND
colores.append('Amarillo')
print('La lista tiene : ',len(colores),' colores.')
```

```
La lista tiene :  5  colores.
```

2- Método COUNT.
   a. Este método nos permite saber cuántas veces aparece un determinado elemento dentro de una lista.

```
# Método COUNT
print('El color azul se aparece: ',colores.count('azul'),' veces.')
```

```
El color azul se aparece:  2  veces.
```

3- Método INDEX.
   a. Este método recibe un índice como argumento, y devuelve la posición de la primera aparición de ese elemento dentro de la lista.

```
: # Método INDEX
  lista = [1,2,3,4,5,6,7,8,9,4]
  print('El número 4 aparece por primera vez en la posición:  ',lista.index(4))
```

```
El número 4 aparece por primera vez en la posición:   3
```

4- Método INSERT.
   a. Permite insertar un nuevo elemento dentro de una lista, con un índice determinado.

```
lista.insert(3,'A')
print(lista)
```

```
[1, 2, 3, 'A', 4, 5, 6, 7, 8, 9, 4]
```

5- Método POP.
    a. Devuelve el último elemento de una lista e inmediatamente lo elimina.

** Hacer un ejemplo del uso de este método, guardando el elemento eliminado en una variable.

6- Método REMOVE.
    a. Borra un determinado elemento de una lista. Si el elemento aparece más de una vez, solo borra la primera aparición, y si el elemento no está en la lista, dará un mensaje de error.

```
lista.remove('A')
print(lista)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

7- Método REVERSE.
    a. Invierte el orden de los elementos de una lista.

```
lista.reverse()
print(lista)
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

8- Método SORT.
    a. Ordena todos los elementos de una lista.

```
numeros = [6,0,1,5,23,87,32,9,14]
numeros.sort(reverse=True)
print(numeros)
```

```
[87, 32, 23, 14, 9, 6, 5, 1, 0]
```

Operadores relacionales.

Son aquellos que utilizamos para evaluar una determinada proposición o condición booleana, es decir, aquella que al ser evaluada solo arroja dos posibles resultados, verdadero o falso.

Símbolo	Significado	Ejemplo	Resultado
`==`	Igual que	`5 == 7`	`False`
`!=`	Distinto que	`rojo != verde`	`True`
`<`	Menor que	`8 < 12`	`True`
`>`	Mayor que	`12 > 7`	`True`
`<=`	Menor o igual que	`12 <= 12`	`True`
`>=`	Mayor o igual que	`4 >= 5`	`False`

Operadores lógicos.

Este tipo de operadores se utilizan para evaluar más de una condición o proposición booleana.

Nos permiten encadenas estas proposiciones, para que sean evaluadas en su totalidad como si fuese una sola condición.

Operador	Ejemplo	Explicación	Resultado
and	5 == 7 and 7 < 12	False and False	False
and	9 < 12 and 12 > 7	True and True	True
and	9 < 12 and 12 > 15	True and False	False
or	12 == 12 or 15 < 7	True or False	True
or	7 > 5 or 9 < 12	True or True	True
xor	4 == 4 xor 9 > 3	True o True	False
xor	4 == 4 xor 9 < 3	True o False	True

Estructura condicional IF.

Se utiliza para tomar decisiones en base a una condición dada.

Por ejemplo, evaluamos un número, si es par, ejecutamos ciertas acciones, en cambio, si es impar ejecutamos otras acciones diferentes.

```
# Estructura de decisión
if condición :
    código a ejecutar si la condición es verdadera
elif otra condición:
    código a ejecutar si la otra condición es verdadera.
else:
    código a ejecutar en caso de no existir condición verdadera.
```

** Escribamos el ejemplo del número par.

```
# Ejemplo del número par.
a = 134
r = a%2

if r == 0:
    print('El numero ',a,' es PAR.')
else:
    print('El numero ',a,' es IMPAR')
```

```
El numero  134  es PAR.
```

Estructura FOR.

En Python, la sentencia FOR tiene un uso que difiere un poco del que se le da en otros lenguajes.

En nuestro caso la sentencia FOR itera sobre los elementos de cualquier secuencia, por ejemplo, una lista, en el orden en el que aparecen dichos elementos.

Por ejemplo:

```
numeros = [6,0,1,5,23,87,32,9,14]
for numero in numeros:
    print(numero*10)
```

```
60
0
10
50
230
870
320
90
140
```

En el caso anterior vimos como recorría una lista valor, por valor.

También podemos aprovechar esta estructura para generar nuestras propias listas.

Por ejemplo, si queremos generar los números impares múltiplos de 7 entre el 3000 y el 3678, podríamos hacer algo así.

```
multiplos = []
for numero in range(3000,3679):
    if numero%7 == 0:
        multiplos.append(numero)

print(multiplos)
```

```
[3003, 3010, 3017, 3024, 3031, 3038, 3045, 3
3150, 3157, 3164, 3171, 3178, 3185, 3192, 31
297, 3304, 3311, 3318, 3325, 3332, 3339, 334
44, 3451, 3458, 3465, 3472, 3479, 3486, 3493
1, 3598, 3605, 3612, 3619, 3626, 3633, 3640,
```

Un último ejemplo, es el de usar la estructura FOR indicándole un determinado incremento entre valor y valor.

```
for numero in range (0,20,2):
    print(numero)
```

```
0
2
4
6
8
10
12
14
16
18
```

Funciones.

Para definir funciones, utilizamos la palabra reservada DEF y a continuación, el nombre que le damos a la función y a lista de parámetros.
Las sentencias que forman parte del cuerpo de la función, comienzan a partir de la siguiente línea, y deben estar identadas (sangría).

Escribamos una función básica a la que le pasamos un entero por argumento, y nos devuelve dicho número, elevado al cuadrado.

```
def cuadrado(valor):
        resultado = valor**2
        print('El cuadrado de ',valor,' es ',resultado)

cuadrado(239671)
```

```
El cuadrado de  239671  es  57442188241
```

Números aleatorios.

Para trabajar con ellos, deberemos importarlos.

```
import random
```

Para utilizarlo, nos valemos del método randrange(x,y,z) donde:

- x: inicio del rango de números.
- y: fin del rango de números (no está incluido)
- z: incremento.

Solo es necesario el primer argumento, veamos algunos ejemplos de uso:

```
import random
print('Numero al azar entre 100 y 150: ',random.randrange(100,150))
```

```
Numero al azar entre 100 y 150:  148
```