



Formación

Curso JSON/PYTHON

*logi*RAIL



Formador

Ana Isabel Vegas



INGENIERA INFORMÁTICA con Master Máster Universitario en Gestión y Análisis de Grandes Volúmenes de Datos: Big Data, tiene la certificación PCEP en Lenguaje de Programación Python y la certificación JSE en Javascript. Además de las certificaciones SCJP Sun Certified Programmer for the Java 2 Platform Standard Edition, SCWD Sun Certified Web Component Developer for J2EE 5, SCBCD Sun Certified Business Component Developer for J2EE 5, SCEA Sun Certified Enterprise Architect for J2EE 5.

Desarrolladora de Aplicaciones FULLSTACK, se dedica desde hace + de 20 años a la CONSULTORÍA y FORMACIÓN en tecnologías del área de DESARROLLO y PROGRAMACIÓN.



training@iconotc.com

▣ **Duración:** 20 horas

▣ **Modalidad:** Remota en tiempo real.

▣ **Fechas/Horario:**

- Días 21, 22, 23, 24 y 25 Abril 2025
- Horario: 10:00 – 14:00 hs

▣ **Objetivos:** Durante el curso, los participantes aprenderán:

- Comprensión y Lectura de Código: Desarrollar la capacidad de leer y entender fragmentos de código en Python.
- Fundamentos de Python: Familiarizar a los participantes con los tipos de datos básicos y estructuras fundamentales.
- Uso de JSON: Comprender el formato JSON, cómo se estructura y cómo se integra en Python para el intercambio de datos.
- Funciones y Modularidad: Introducir la definición y el uso de funciones en Python para estructurar el código.
- Introducción a APIs: Mostrar cómo consumir APIs, gestionar peticiones y procesar respuestas en formato JSON.

▣ **Contenidos:**

- Fundamentos de Python
 - Tipos de datos: Números, cadenas, listas, diccionarios y booleanos.
 - Estructuras de control: Condicionales y bucles.
 - Operadores y expresiones básicas.
- Estructuras de Datos y Mapeo
 - Introducción a estructuras complejas: tuplas, conjuntos y diccionarios.
 - Concepto de mapeo y transformación de datos.
 - Ejercicios prácticos para reforzar la comprensión.

▣ Contenidos (continuación):

- Manejo de JSON

Conceptos básicos de JSON: Sintaxis, objetos, y arreglos.

Conversión entre JSON y estructuras de datos en Python (uso de librería Json)

Ejercicios de lectura y escritura de archivos JSON.

- Funciones en Python

Definición y declaración de funciones.

Parámetros, argumentos y retorno de valores.

Funciones anónimas (lambda) y scope de variables.

Ejercicios prácticos y casos de uso.

- APIS

Conceptos básicos: ¿Qué es una API y como se utiliza?

Realizar peticiones HTTP (usando librerías como requests)

Procesamiento de respuestas en formato JSON

Verbos HTTP: GET, POST, PUT, PATCH, DELETE

Autenticación y autorización

Testing con Postman

Real time con APIS

Revisión de principales frameworks de desarrollo

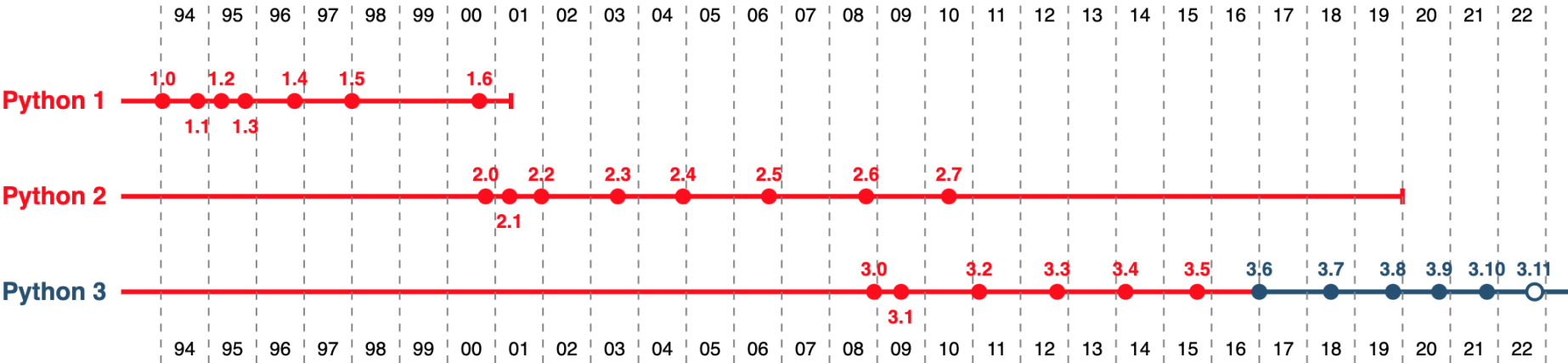
Fundamentos de Python

Tema 1

Qué es Python?

- Python es un lenguaje de programación que fue creado a finales de los años 80 por el holandés Guido van Rossum, fan del grupo humorístico Monty Python, de ahí el nombre que le puso al lenguaje de programación.

Versiones de Python



Características de Python

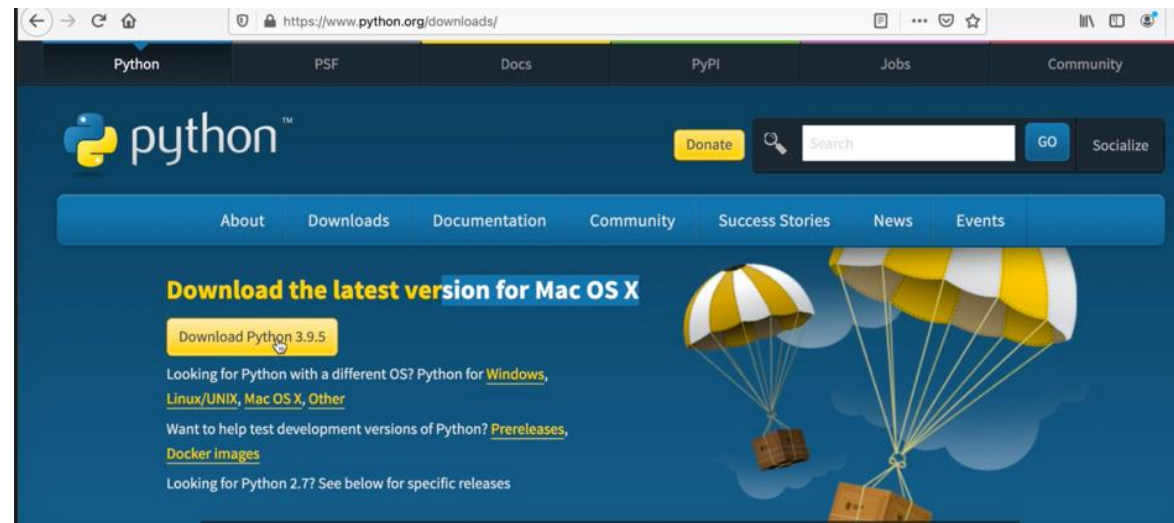
- **Simpleza:** es un lenguaje muy sencillo, de ahí el gran éxito de Python.
- **Sintaxis clara:** es obligatoria la indentación
- **Propósito general:** se pueden crear todo tipo de programas
- **Lenguaje interpretado:** no es necesaria compilación, el intérprete de Python lo ejecuta.
- **Lenguaje de alto nivel:** no hay que preocuparse del manejo de memoria y otros aspectos de bajo nivel
- **Lenguaje orientado a objetos:** Favorece la reutilización de código
- **Open source:** es un lenguaje de programación gratuito
- **Extensas librerías:** incluyen muchas funcionalidades
- **Incrustable:** es posible añadir programas Python a programas en C y C++.

Ventajas

- **Legible:** sintaxis intuitiva y estricta
- **Productivo:** ahorra mucho código
- **Portable:** para todo sistema operativo
- **Recargado:** viene con muchas librerías por defecto

Instalación en Windows

- Desde la página www.python.org/download se descarga y se instala.
- El sitio reconocerá el sistema operativo y te dará las opciones para descargar.



- Ejecuta el archivo que descargaste y sigue los pasos de instalación.

Instalación en MAC

- Puedes instalarlo descargando la versión MAC
- O también se puede instalar con el comando brew

Para instalar la Versión 2.7

```
brew install python
```

Para instalar la Versión 3.x

```
brew install python3
```

Comprobar versión instalada

- Desde la terminal de comandos:

Versión 2.7

```
python -v
```

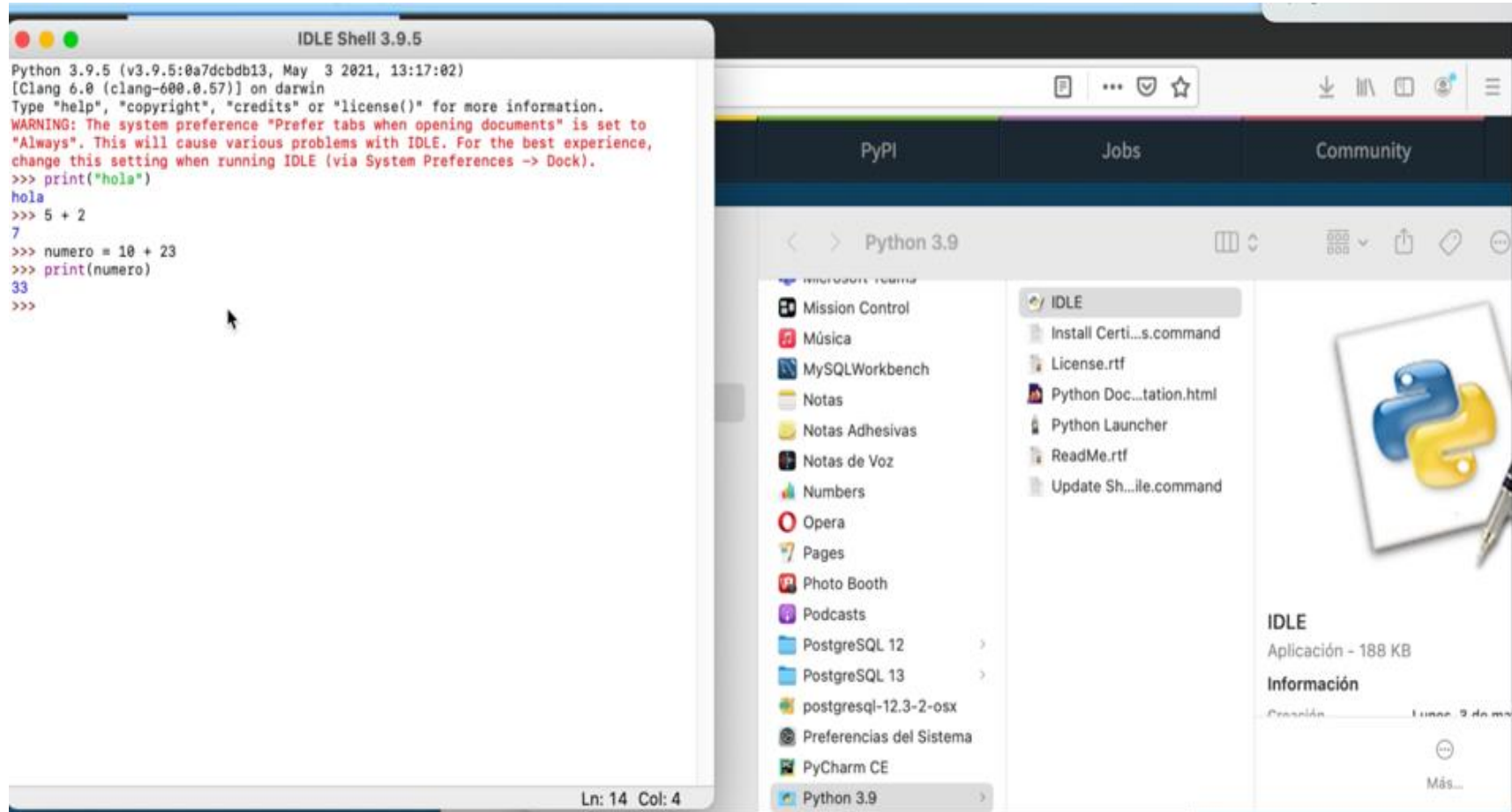
Versión 3.x

```
python3 -v
```

Principales entornos de desarrollo

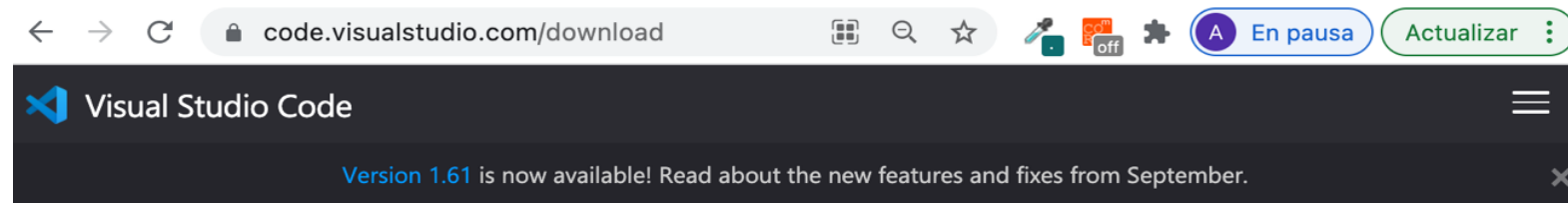
- Son muchos los entornos que podemos utilizar:
 - IDLE
 - Visual Studio Code
 - Jupiter
 - Anaconda
 - Spyder
 - PyCharm

IDLE




Para salir cerrar ventana o Ctrl + D.

Visual Studio Code



Download Visual Studio Code


Free and built on open source. Integrated Git, debugging and extensions.



↓ **Windows**

Windows 7, 8, 10, 11

User Installer	64 bit	32 bit	ARM
System Installer	64 bit	32 bit	ARM
.zip	64 bit	32 bit	ARM



↓ **.deb**


Debian, Ubuntu

.deb	64 bit	ARM	ARM 64
.rpm	64 bit	ARM	ARM 64
.tar.gz	64 bit	ARM	ARM 64

Snap Store

↓ **.rpm**

Red Hat, Fedora, SUSE

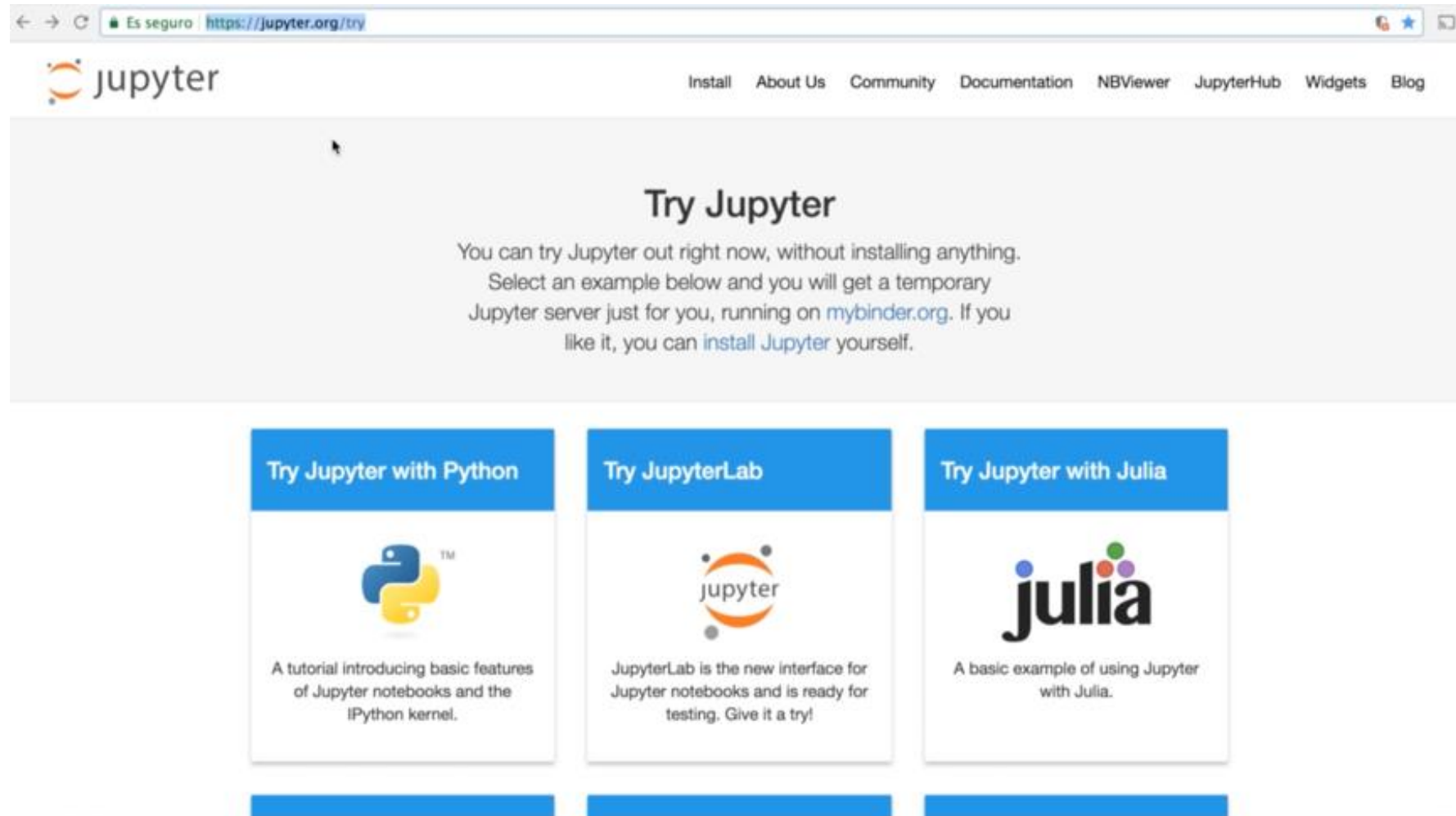


↓ **Mac**

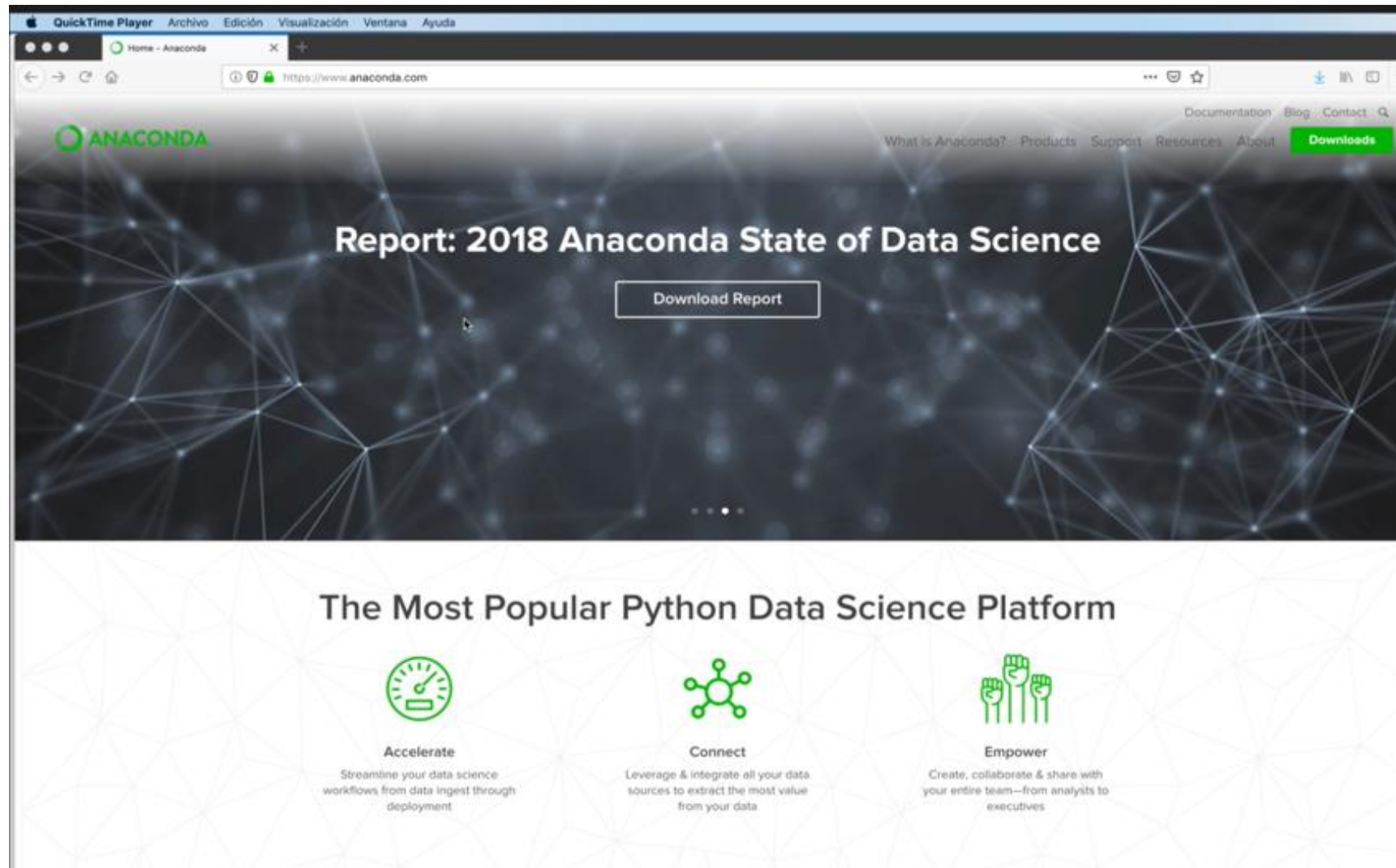
macOS 10.11+

.zip **Universal** **Intel Chip** **Apple Silicon**


Jupyter




Anaconda

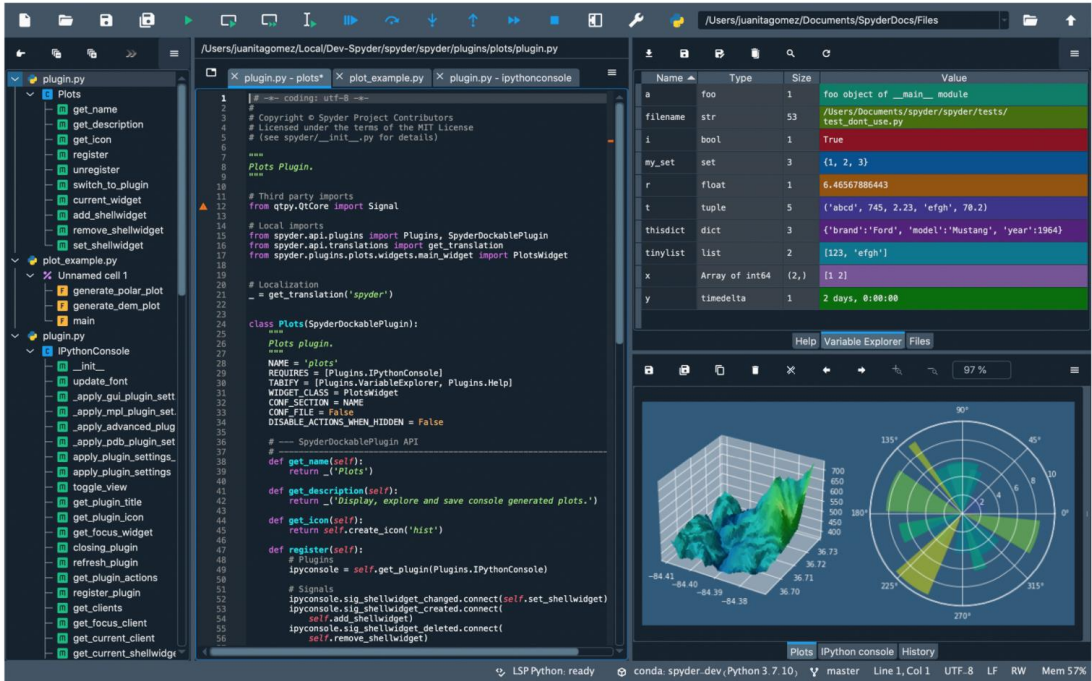


spyder-ide.org



[HOME](#)
[OVERVIEW](#)
[COMPONENTS](#)
[PLUGINS](#)
[DOWNLOAD](#)
[DONATE](#)





The screenshot displays the Spyder IDE interface. The left sidebar shows a project tree with files like `plugin.py`, `plots`, and `plugin_console`. The central code editor shows the `plugin.py` file, which defines a `Plots` class inheriting from `SpyderDockablePlugin`. The right sidebar contains the `Variable Explorer` showing a table of variables and their values, and the `Plots` panel displaying a 3D surface plot and a polar plot.

Name	Type	Size	Value
a	foo	1	foo object of __main__ module
filename	str	53	/Users/Documents/spyder/spyder/tests/test_dont_use.py
i	bool	1	True
my_set	set	3	{1, 2, 3}
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 78.2)
thisdict	dict	3	{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
tinylist	list	2	[123, 'efgh']
x	Array of int64	(2,)	[1 2]
y	timedelta	1	2 days, 0:00:00

PyCharm



Declaración de variables

- Las variables, a diferencia de los demás lenguajes de programación, no debes definir las, ni tampoco su tipo de dato, ya que al momento de iterarlas se identificará su tipo.
- Debe comenzar por letra o _
- Es case sensitive, diferencia mayúsculas de minúsculas
- Para saber el tipo de dato `type(variable)`

```
A = 3  
B = A
```

```
numero1 = 5  
numero2 = 4  
suma = numero1 + numero2
```

Tipos de datos

Tipo de dato	Descripción
Entero	Número sin decimales, tanto positivo como negativo, incluyendo el 0.
Real	Número con decimales, tanto positivo como negativo, incluyendo el 0.
Complejo	Número con parte imaginaria.
Cadenas de texto	Texto.
Booleanos	Pueden tener dos valores: True o False
Conjuntos	Colección de elementos no ordenados y no repetidos.
Lista	Vector de elementos que pueden ser de diferentes tipos de datos.
Tuplas	Lista inmutable de elementos.
Diccionario	Lista de elementos que contienen claves y valores.

Conversiones entre tipos de datos

- **Números enteros**
- Toda entrada de usuario se interpreta como una cadena de texto, para poder transformarlo en un numero, se utiliza la instrucción int.

```
sumando1 = int(input("Introduzca el primer sumando: "))  
sumando2 = int(input("Introduzca el segundo sumando: "))  
print("Resultado de la suma: ", sumando1 + sumando2)
```

Conversiones entre tipos de datos

- **Números reales**
- Para transformar una cadena de texto en un número real, utilizamos la instrucción float.

```
sumando1 = float(input("Introduzca el primer sumando (Decimal): "))  
sumando2 = float(input("Introduzca el segundo sumando (Decimal): "))  
print("Resultado de la suma: ", sumando1 + sumando2)
```

- Para redondear números reales utilizamos la instrucción round que utiliza dos parámetros:
 - El primero de ellos es el número real que quieres redondear
 - El segundo es el número de decimales al que quieres redondear.

```
dividendo = float(input("Introduzca el dividendo: "))  
divisor = float(input("Introduzca el divisor: "))  
resultado = round(dividendo / divisor,1)  
print("Resultado de la division: ", resultado)
```

Conversiones entre tipos de datos

- **Cadenas de texto**
- Para convertir un numero a cadena de texto y asi mostrarlo por pantalla utilizamos el comando str:

```
print("Has escrito el numero " + str(numero))
```


Cadenas de texto

- Las cadenas de texto son arrays de caracteres.
- Deben ir entre comillas
- Además se pueden utilizar los siguientes caracteres especiales:
 - `\n`: implica un salto de línea dentro de la cadena de texto.
 - `\\`: carácter para introducir la barra `\` en la cadena de texto.
 - `\'`: carácter para introducir una comilla simple en la cadena de texto.
 - `\"`: carácter para introducir una comilla doble en la cadena de texto.
 - `\t`: carácter para introducir una tabulación horizontal en la cadena de texto.

Imprimir variables de cadenas de texto

- El comando print lo utilizamos para mostrar información por pantalla. Podemos mostrar uno o varios elementos separados por comas.
- A través de este comando podemos utilizar 2 operadores:
 - sep; Nos permite establecer una cadena de texto que actuará como separador de los elementos a mostrar.
 - end; Será la cadena que se muestre al final.

```
# Comando print

print("Hola")
print("Juan", "Pedro", "Maria", "Luis")

#Parametro sep
print("Juan", "Pedro", "Maria", "Luis", sep=' | ')

#Parametro end
print("Juan", "Pedro", "Maria", "Luis", sep=',', end='.')
```

```
Python 3.7.7 (v3.7.7:d7c567b081
[Clang 6.0 (clang-600.0.57)] or
Type "help", "copyright", "crec
>>>
= RESTART: /Users/anaisabelvega
mplo2_print.py
Hola
Juan Pedro Maria Luis
Juan | Pedro | Maria | Luis
Juan,Pedro,Maria,Luis.
>>> |
```

Entradas de texto por teclado

- El comando input se utiliza para leer información de entrada, desde el teclado.

```
# Comando input

print('Cual es tu nombre? ')
nombre = input()
print('Hola ', nombre, 'Bienvenido al curso !!')

# Otra forma
nombre = input('Cual es tu nombre? ')
print('Hola ', nombre, 'Bienvenido al curso !!')
```

Funciones con cadenas de texto

- Para manejar cadenas de texto podemos hacer uso de las siguientes funciones:
 - Función capitalize
 - Función upper
 - Función lower
 - Función len
 - Función isalnum
 - Función isalpha
 - Función isdigit
 - Función islower
 - Función isupper
 - Funciones lstrip, rstrip, strip
 - Funciones max y min
 - Funcion replace
 - Función swapcase
 - Función split

Función capitalize

- Pone la primera letra, solo la primera, en mayúsculas.

```
cadenaejemplo = "en un lugar de la mancha..."  
print(cadenaejemplo.capitalize())
```

Función upper

- Pone en mayúsculas toda la cadena de texto.

```
cadenaejemplo = "en un lugar de la mancha..."  
print(cadenaejemplo.upper())
```

Función lower

- Pone en minúsculas toda la cadena de texto.

```
cadenaejemplo = "EN UN LUGAR DE LA MANCHA..."  
print(cadenaejemplo.lower())
```

Función len

- Permite saber el número de caracteres que componen la cadena de texto.

```
cadenaejemplo = "En un lugar de la Mancha..."  
print(len(cadenaejemplo))
```


Función isalnum

- Comprueba si todos los caracteres que componen la cadena de texto son alfanuméricos o no. Devuelve un valor booleano.

```
cadenaejemplo = "En un lugar de la mancha..."
print(cadenaejemplo.isalnum())
cadenaejemplo = "1234567890"
print(cadenaejemplo.isalnum())
cadenaejemplo = "abcdefg1234567890"
print(cadenaejemplo.isalnum())
cadenaejemplo = "abcdefg 1234567890"
print(cadenaejemplo.isalnum())
```

- los caracteres del punto y del espacio en blanco que contienen las cadenas primera y cuarta no son caracteres alfanuméricos.

Función isalpha

- Comprueba si todos los caracteres de la cadena de texto son caracteres alfabéticos.
- Ni los números, ni los puntos ni los espacios en blanco son caracteres alfabéticos, únicamente las letras componen el conjunto de caracteres alfabéticos.

```
cadenaejemplo = "Enunlugardelamancha"  
print(cadenaejemplo.isalpha())  
cadenaejemplo = "En un lugar de la mancha"  
print(cadenaejemplo.isalpha())  
cadenaejemplo = "1234567890"  
print(cadenaejemplo.isalpha())  
cadenaejemplo = "abcdefg 1234567890"  
print(cadenaejemplo.isalpha())
```

Función isdigit

- Comprueba si todos los caracteres de la cadena de texto son caracteres numéricos.

```
cadenaejemplo = "En un lugar de la mancha"  
print(cadenaejemplo.isdigit())  
cadenaejemplo = "1234567890"  
print(cadenaejemplo.isdigit())  
cadenaejemplo = "abcdefg 1234567890"
```

Función islower

- Comprueba si todos los caracteres que componen la cadena están en minúscula.

```
cadenaejemplo = "En un lugar de la mancha"  
print(cadenaejemplo.islower())  
cadenaejemplo = "en un lugar de la mancha"  
print(cadenaejemplo.islower())
```

Función isupper

- Comprueba si todos los caracteres que componen la cadena de texto están en mayúscula.

```
cadenaejemplo = "En un lugar de la mancha"  
print(cadenaejemplo.isupper())  
cadenaejemplo = "EN UN LUGAR DE LA MANCHA"  
print(cadenaejemplo.isupper())
```

Función lstrip, rstrip, strip

- Para eliminar los caracteres del comienzo de la cadena tienes que utilizar lstrip, para eliminar los caracteres del final de la cadena tienes que utilizar rstrip, y por último, para eliminar ambos a la vez tienes que utilizar strip, que hace lo mismo que los dos anteriores pero en una sola instrucción.

```
cadenaejemplo = " En un lugar de la mancha"  
print(cadenaejemplo.lstrip())  
cadenaejemplo = "En un lugar de la mancha "  
print(cadenaejemplo.rstrip())  
cadenaejemplo = " En un lugar de la mancha "  
print(cadenaejemplo.strip())
```

Funciones max y min

- permiten conocer el carácter alfabético mayor y menor de la cadena de texto.

```
cadenaejemplo = "abcdefghijklmnopqrstuvwxyz"  
print(max(cadenaejemplo))  
print(min(cadenaejemplo))
```

Función replace

- Va a permitir reemplazar caracteres de la cadena de texto por otros caracteres.
- En el siguiente ejemplo se reemplaza la A por una E.

```
cadenaejemplo = "AEIOU"  
print(cadenaejemplo.replace('A','E'))
```


Función swapcase

- Va a permitir invertir las mayúsculas y minúsculas de la cadena de texto, es decir, las mayúsculas pasarán a ser minúsculas y las minúsculas pasarán a ser mayúsculas.

```
cadenaejemplo = "En un lugar de la mancha"  
print(cadenaejemplo.swapcase())
```

Función split

- Va a permitir convertir una cadena de texto en una lista de elementos que se encuentran separados por espacios en la cadena de texto original.

```
cadenaejemplo = "En un lugar de la mancha"  
print(cadenaejemplo.split())
```

- Podemos indicar el carácter que tiene que utilizar para separar los elementos de la lista.

```
cadenaejemplo = "31/12/2017"  
print(cadenaejemplo.split("/"))
```

Operadores aritméticos

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
**	Potencia
//	División entera

Operadores de asignación

Operador	Significado
<code>+=</code>	Suma
<code>-=</code>	Resta
<code>*=</code>	Multiplicación
<code>/=</code>	División
<code>%=</code>	Módulo
<code>**=</code>	Potencia
<code>//=</code>	División entera

Operadores relacionales

Operador	Significado
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual que
!=	Distinto que

Operadores lógicos

- Los operadores lógicos que puedes utilizar son los siguientes:
 - **and**: operador lógico que realiza la operación lógica 'Y' entre dos elementos. El resultado será true si ambos elementos son true, en caso contrario será false.
 - **or**: operador lógico que realiza la operación lógica 'O' entre dos elementos. El resultado será true si uno de los dos elementos es true, en caso contrario será false.
 - **not**: operador lógico que realiza la operación lógica 'NO'. El resultado será true si el elemento es false, y será false si es true.

Operadores de identidad

- intentan identificar si los objetos son realmente el mismo objeto o si son diferentes objetos.

```
# Operadores de identidad (is, is not)
```

```
frutas1 = ["manzana", "pera"]  
frutas2 = ["manzana", "pera"]  
frutas3 = frutas1
```

```
frutas3 is frutas1
```

```
True
```

Operadores de pertenencia

- Los operadores de pertenencia permiten verificar si un valor está dentro de una lista de valores o si un objeto está dentro de una lista de objetos.

```
# Operadores de pertenencia (in, not in)
```

```
frutas1 = ["manzana", "pera", "naranja"]  
frutas2 = "pera"
```

```
frutas2 in frutas1
```

```
True
```

```
# not in
```

```
frutas2 not in frutas1
```

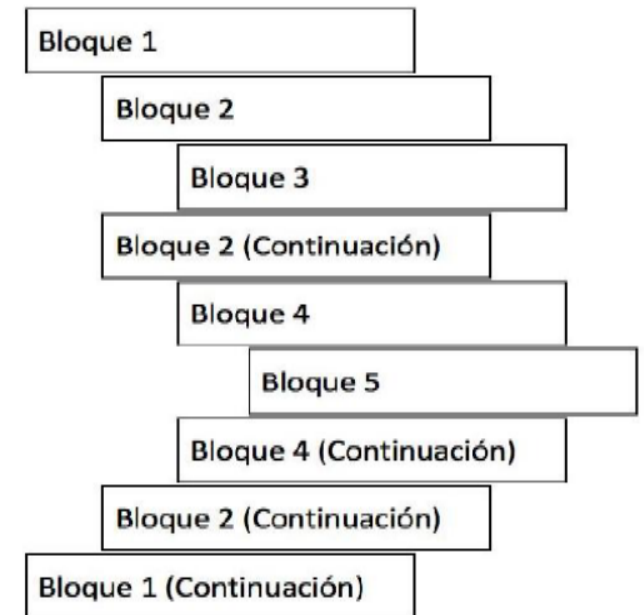
```
False
```

```
frutas3 = "melocoton"  
frutas3 not in frutas1
```

```
True
```


Bloques e indentación

- Un bloque es un grupo de sentencias de código fuente que contiene una o más sentencias. Los bloques están delimitados por su inicio y su fin, y la forma de delimitarlos es específica de cada lenguaje de programación.
- Indentación significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para así separarlo del margen izquierdo y distinguirlo más fácilmente dentro del texto. Todos los lenguajes utilizan la indentación para aumentar la legibilidad del código fuente, pero en Python no es únicamente una buena práctica estética, ya que utiliza la indentación del código fuente para delimitar los bloques dentro del código fuente, es por eso por lo que es necesario que la utilices de forma correcta.



Condicionales

- **If / elif / else**

```
if numero1 > numero2 :  
    BloqueInstrucciones1  
elif numero1 == numero2 :  
    BloqueInstrucciones2  
else :  
    BloqueInstrucciones3
```

Bucles o Ciclos

- **Bucle for**
- En Python, los bucles for se ejecutan sobre elementos iterables, como pueden ser listas, tuplas, cadenas de texto o diccionarios.

for Variable in ColeccionIterable:
BloqueInstrucciones

```
lista = [1,2,3,4,5,6,7,8,9]
for item in lista:
    print(item, end=" ")
```

Bucles o Ciclos

- **Bucle while**

while Condición:
BloqueInstrucciones

```
i = 0
while i<10:
    print(i,end=" ")
    i = i + 1
```

Estructuras de datos y mapeos

Tema 2

Listas

- Una lista es un conjunto ordenado de elementos que puede contener datos de cualquier tipo. Es el tipo de colección más flexible de todos.
- Las listas pueden contener elementos del mismo tipo o elementos de diferentes tipos.
- En Python las listas se delimitan con corchetes “[]”, con los elementos separados por comas.
- La propiedad len devuelve la longitud, es decir, el numero de elementos que contiene la lista.
- El primer elemento de una lista es el elemento 0, no el 1.

```
lista = ["ordenador","teclado","raton"]  
print(len(lista))  
print(lista[0])  
print(lista[1])  
print(lista[2])
```

Listas

- Podemos unir listas a través del operador '+’.

```
listaoriginal = ["ordenador","teclado","raton"]
listanueva = ["monitor","impresora","altavoces"]
listafinal = listaoriginal + listanueva
print(listafinal)
```

- Con el operador '+’ también podemos añadir un elemento a la lista:

```
lista = ["ordenador","teclado","raton"]
print(lista)
lista = lista + ["mesa"]
print(lista)
```

- Para eliminar un elemento de la lista utilizamos el operador del:

```
lista = ["ordenador","teclado","raton"]
print(lista)
del lista[1]
print(lista)
```

Listas

- Hasta ahora hemos visto listas de una sola dimensión, si queremos listas de dos dimensiones necesitaríamos crear una lista como elemento de otra lista.

```
lista = ["ordenador", "teclado", "raton", ["tarjeta de sonido", "microfono", "altavoces"]]  
print(lista[0])  
print(lista[1])  
print(lista[2])  
print(lista[3])  
print(lista[3][0])  
print(lista[3][1])  
print(lista[3][2])
```


Tuplas

- Las tuplas son un conjunto de elementos ordenados e inmutables.
- La diferencia con las listas reside en que en las listas puedes manipular los elementos y en las tuplas no. Las tuplas pueden contener elementos del mismo tipo o elementos de diferentes tipos, al igual que las listas.
- En Python las tuplas se delimitan por paréntesis “()”, con los elementos separados por comas.

```
tupla = ("ordenador","teclado","raton")  
print(tupla)  
print(len(tupla))  
print(tupla[0])  
print(tupla[1])  
print(tupla[2])
```

Conjuntos

- Un conjunto es una colección de elementos pero que está desordenado es decir no hay un índice

```
# Conjuntos
```

```
conjunto_colores = {"rojo", "verde", "azul"}
```

```
conjunto_colores
```

```
{'azul', 'rojo', 'verde'}
```

```
for color in conjunto_colores:  
    print(color)
```

```
azul  
rojo  
verde
```

```
conjunto_colores[0]
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-12-6ccc7b158046> in <module>  
----> 1 conjunto_colores[0]  
  
TypeError: 'set' object does not support indexing
```

```
con|
```

Diccionarios

- Los diccionarios son colecciones de elementos compuestos por una clave y un valor asociado. Las claves en los diccionarios no pueden repetirse.
- En Python los diccionarios se delimitan por corchetes "{ }", con los
- elementos separados por comas y la clave separada del valor mediante dos puntos.

```
mesestraducidos = {"Enero" : "January",  
                  "Febrero" : "February",  
                  "Marzo" : "March",  
                  "Abril" : "April",  
                  "Mayo" : "May",  
                  "Junio" : "June",  
                  "Julio" : "July",  
                  "Agosto" : "August",  
                  "Septiembre" : "September",  
                  "Octubre" : "October",  
                  "Noviembre" : "November",  
                  "Diciembre" : "December"}  
print(mesestraducidos["Noviembre"])  
print(mesestraducidos["Mayo"])
```

Manejo de JSON

Tema 3

JSON

- JSON es una especie de puente universal capaz de mover datos entre partes aparentemente incompatibles.
- JSON es la respuesta a una necesidad bastante básica: la necesidad de transferir datos que son el contenido de un objeto o conjunto de objetos.
- Podemos representar:
 - números decimales, hexadecimales, octales y binarios
 - números reales y notación científica
 - textos no soporta comillas simples, solo comillas dobles
 - valores booleanos

Objetos y arrays JSON

- Objetos:

```
{"name": "John Doe", "age": 42}
```

- Arrays:

```
[1, 2.34, True, 'False', None, ['a', 0]]
```

- Combinación de objetos con arrays:

```
{"me": "Python", "pi": 3.141592653589, "data": [1, 2, 4, 8], "friend": "JSON", "set": null}
```

Módulo JSON

- Python nos proporciona el módulo JSON para realizar mapeos y conversiones entre los distintos tipos de Python.
- Las principales funciones son:
 - `dumps()`, convierte los datos de Python en una cadena JSON
 - `loads()`, hace lo contrario, coge la cadena JSON y lo convierte en un dato Python.

Funciones en Python

Tema 4

Funciones

- Sintaxis:

```
def NombreFuncion (parámetros):  
    BloqueInstrucciones  
    return ValorRetorno
```

```
def Saludar():  
    print("¡Hola Time of Software!")  
Saludar()
```

Retornar más de un elemento

- Se pueden devolver más de un elemento con return

```
def SumarRestar(param1, param2):  
    return param1 + param2, param1 - param2  
  
numero1 = int(input("Introduce el primer numero: "))  
numero2 = int(input("Introduce el segundo numero: "))  
resultadosuma, resultadoresta = SumarRestar(numero1, numero2)  
print("El resultado de la suma es: ", resultadosuma)  
print("El resultado de la resta es: ", resultadoresta)
```

Funciones con número variable de argumentos

- En estas funciones los parámetros se reciben como una lista, por lo que tendrás que iterarla para poder procesarlos todos.

```
def Sumar(*valores):  
    resultado = 0  
    for item in valores:  
        resultado = resultado + item  
    return resultado
```

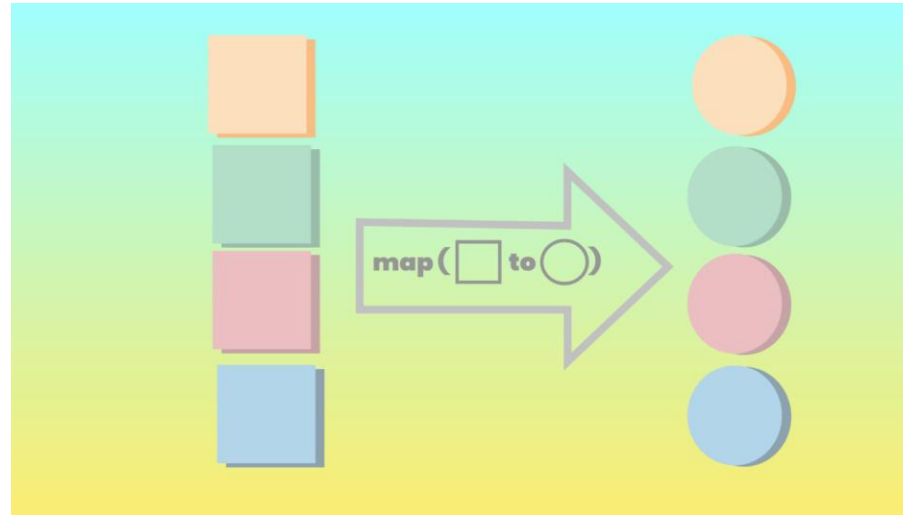
```
resultado = Sumar(23,56,3,89,78,455)  
print("El resultado de la suma es: ", resultado)
```

Funciones anidadas

```
def SumarRestar(param1, param2):  
    return Sumar(param1,param2), Restar(param1,param2)  
  
def Sumar(sumando1, sumando2):  
    return sumando1 + sumando2  
  
def Restar(minuendo, sustraendo):  
    return minuendo - sustraendo  
  
numero1 = int(input("Introduce el primer numero: "))  
numero2 = int(input("Introduce el segundo numero: "))  
resultadosuma, resultadoresta = SumarRestar(numero1,numero2)  
print("El resultado de la suma es: ", resultadosuma)  
print("El resultado de la resta es: ", resultadoresta)
```

Función map

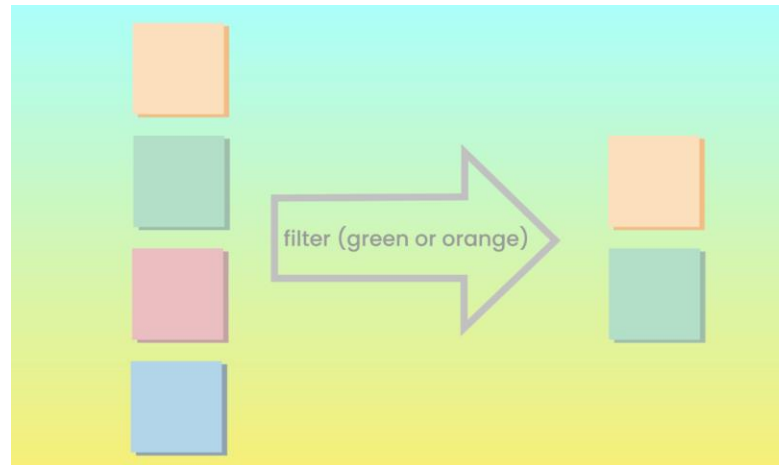
- La función map nos permite aplicar una función sobre cada uno de los elementos de un colección (Listas, tuplas, etc...).



```
map(función a aplicar, objeto iterable)
```

Función filter

- La función filter, es quizás, una de las funciones más utilizadas al momento de trabajar con colecciones. Como su nombre lo indica, esta función nos permite realizar un filtro sobre los elementos de la colección.



```
filter(función a aplicar, objeto iterable)
```

Función reduce

- Usaremos la función reduce cuando poseamos una colección de elementos y necesitemos generar un único resultado. reduce nos permitirá reducir los elementos de la colección. Podemos ver a esta función como un acumulador.

```
reduce(función a aplicar, objeto iterable)
```

Funciones lambda

- Una función lambda es una función anónima, una función que no posee un nombre. En Python la estructura de una función lambda es la siguiente.

```
lambda argumentos : cuerpo de la función
```


APIS

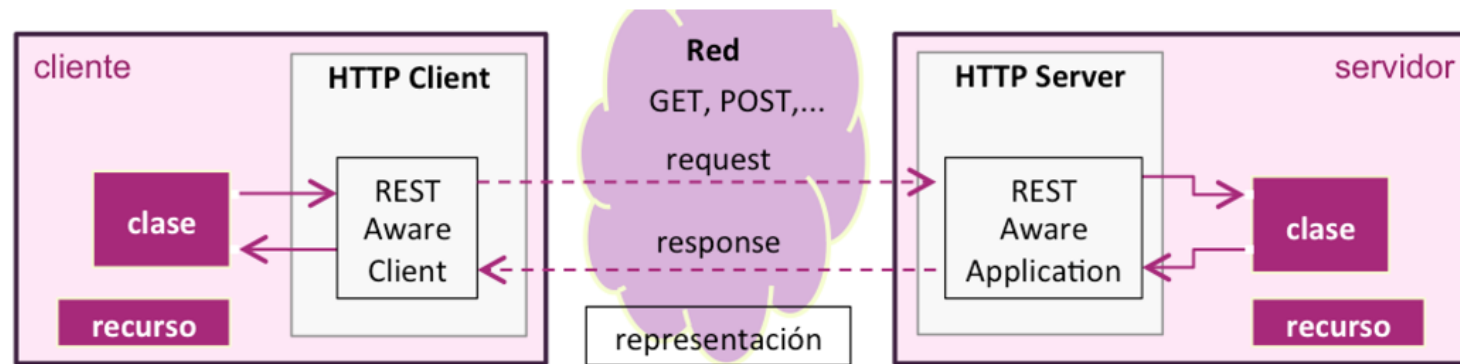
Tema 5

API

- Una API del tipo RESTful, o RESTful Web Service, es una API web implementada con HTTP y los principios REST, con los siguientes aspectos:
 - Una URI base del servicio.
 - Un formato de mensajes, por ejemplo JSON o XML.
 - Un conjunto de operaciones, que utilizan los métodos HTTP (GET, PUT, POST o DELETE).

Servicios REST

- REST (Representational State Transfer) es un estilo de arquitectura para sistemas distribuidos, desarrollada por la W3C, junto con el protocolo HTTP.
- Las arquitecturas REST tienen clientes y servidores.
- El cliente realiza un envío (request) al servidor, el cual lo procesa y retorna una respuesta al cliente.
- Las peticiones y respuestas son construidas alrededor de representaciones de recursos. Recurso es una entidad, y representación es cómo se formatea.



Métodos HTTP

- Con REST, los métodos HTTP se asocian a tipos de operaciones sobre recursos. El uso comúnmente aceptado es el siguiente:
 - GET: Para recuperar la representación de un recurso. Es idempotente, es decir, si se invoca múltiples veces, retorna el mismo resultado.
 - POST: Para crear un recurso. También, por las características del método, se utiliza para envíos grandes, o para evitar limitaciones de los otros métodos.
 - PUT: Para actualizar un recurso.
 - DELETE: Para eliminar un recurso.

JSON / PYTHON



GRACIAS

*logi*RAIL

Completa nuestra encuesta
de satisfacción a través del QR



<https://forms.office.com/e/sYgkUFwdfs>