



1

A circular portrait of a man, Francisco de Borja Cabeza Rozas, wearing a dark suit and tie, set against a dark background.

Francisco de Borja Cabeza Rozas

- Arquitecto de Software
- Microsoft Certified Trainer (MCT)
- Microsoft Certified Solution Expert (MCSE)
Cloud Platform and Infrastructure
- Microsoft Certified Solutions Associate (MCSA)
Cloud Platform and Web Applications
- Microsoft Certified Solution Developer (MCSD)
App Builder, Web Applications y Windows Store Apps

Microsoft CERTIFIED
Trainer

Four Microsoft Certified Professional badges are shown in a row. From left to right:

- MCSE Cloud Platform and Infrastructure, Earned 2018
- MCSA Cloud Platform, Earned 2018
- MCSA Web Applications, Earned 2016
- MCSD App Builder, Earned 2018

2

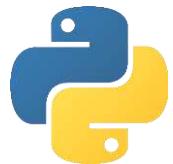
Agenda

1	Entorno de Trabajo	2	Conceptos Básicos
3	Módulos y Paquetes	4	Programación Orientada a Objetos
5	Patrones de Diseño	6	Archivos CSV
7	Acceso a Base de Datos	8	Biblioteca NumPy
9	Servicios con Requests y Zeep	10	Networking
11	Debug y Testing		



3

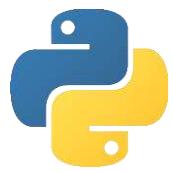
El lema de Python



- 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.
- Escaso es mejor que denso.
- Cuenta la legibilidad.
- Aunque practicidad late pureza.
- Los casos especiales no son lo suficientemente especial como para romper las reglas.
- Los errores nunca debe pasar en silencio.
- A menos que explícitamente silenciados.

4

El lema de Python II



- Ante la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una – y preferiblemente sólo una – manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que seas holandés.
- Ahora es mejor que nunca.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede ser una buena idea.
- Los espacios de nombres son una gran idea
- Vamos a hacer más de espacios de nombres

5



6

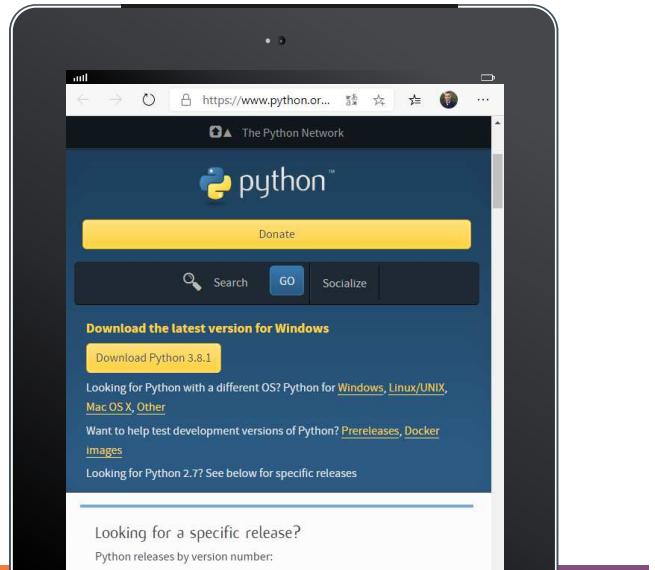
Entorno de Trabajo

Instalación de Python

Tenemos dos opciones para descargar Python:

- <https://www.python.org/downloads/>
- <https://www.anaconda.com/distribution/>

Anaconda es una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la *Data Science* con Python.



7

Entorno de Trabajo

Jupyter Notebook

Jupyter Notebook está basado en la web y nos permite crear documentos Python.

Ofrece todos los componentes básicos del clásico Jupyter Notebook (notebook, terminal, editor de texto, explorador de archivos, texto enriquecido, etc.) en una interfaz de usuario flexible y potente.

Se instala conjuntamente a la distribución de Python, Anaconda.



8

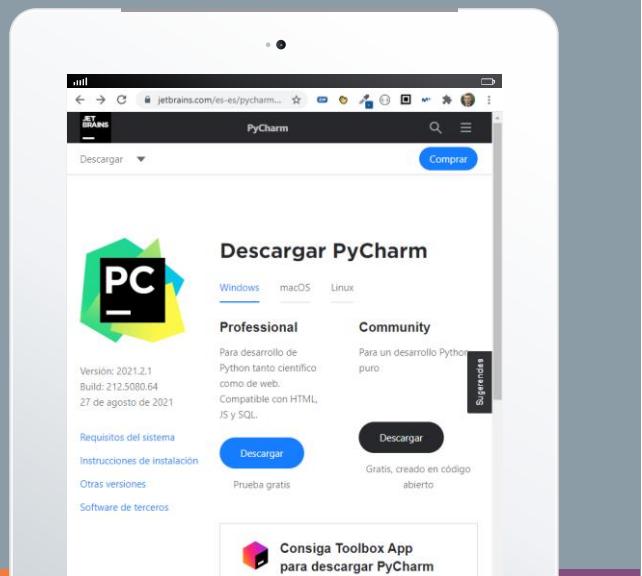
Entorno de Trabajo

Instalación de PyCharm

PyCharm es una IDE creada por JetBrains para desarrolladores profesionales de Python.

Cuenta con una versión Professional de Pago y otra Community Gratis.

- <https://www.jetbrains.com/es-es/pycharm/>



9

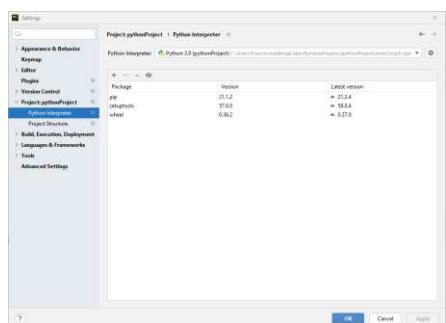
Entorno de Trabajo

Entornos de Trabajo en PyCharm

PyCharm permite utilizar la herramienta `virtualenv` para crear un entorno virtual específico para un proyecto. El entorno virtual es creado en el momento que se crea un nuevo proyecto.

Para realizar cambios en el entorno del proyecto, usamos las teclas (Ctrl+Alt+S).

Desplegamos `Project <project name>` y accedemos la opción `Python Interpreter`.



10

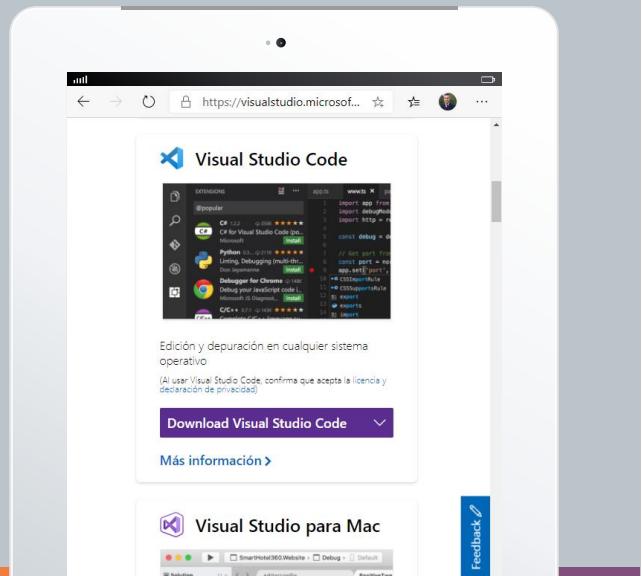
Entorno de Trabajo

Instalación de Visual Studio

Diferentes IDE para trabajar con Python, una buena elección puede ser Visual Studio o Visual Studio Code.

- <https://visualstudio.microsoft.com/>
- <https://code.visualstudio.com/>

Si decides utilizar Visual Studio Code también debes instalar *Python extension for Visual Studio Code* desarrollada por Microsoft.

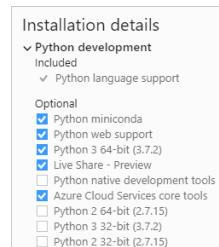


11

Entorno de Trabajo

Instalación de Visual Studio II

Para instalar la compatibilidad de Python para Visual Studio seleccionamos la carga de trabajo Python, que incluye la útil extensión **Cookiecutter** que proporciona una interfaz gráfica de usuario para detectar plantillas, opciones de plantilla y crear proyectos y archivos.



12

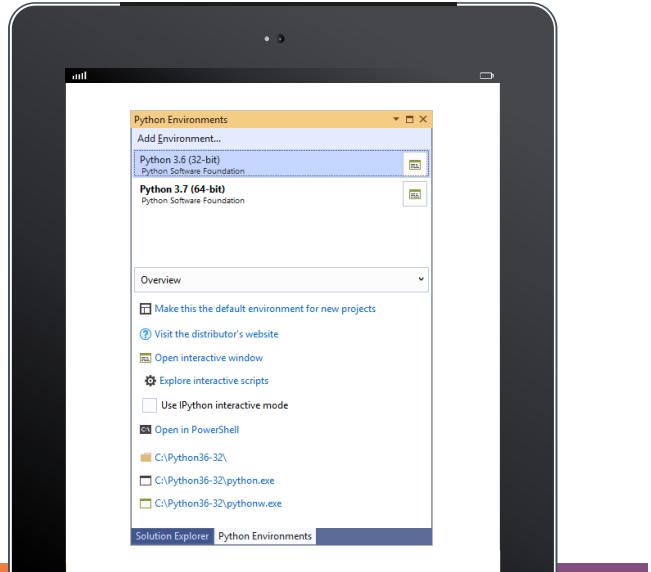
Entorno de Trabajo

Entorno de Trabajo en Visual Studio

Un entorno de Python es un contexto en el que se ejecuta el código de Python.

Un entorno consta de un intérprete, una biblioteca (normalmente la biblioteca estándar de Python) y un conjunto de paquetes instalados.

En Visual Studio, se usa la ventana Entornos de Python, para administrar entornos y seleccionar uno como valor predeterminado para los proyectos nuevos.



13

Entorno de Trabajo

Entornos de Trabajo en Visual Studio

Entornos globales, cada instalación de Python (por ejemplo, Python 2.7, Python 3.9, Python 3.10, Anaconda 2022.10, etc., mantiene su propio *entorno global*.

Cada entorno está formado por el intérprete de Python correspondiente, su biblioteca estándar, un conjunto de paquetes preinstalados y cualquier paquete adicional que instale mientras ese entorno está activado.

Al instalar un paquete en un entorno global, este pasa a estar disponible para todos los proyectos que usan ese entorno.

14

Entorno de Trabajo

Entornos de Trabajo en Visual Studio II

Entornos virtuales, es una subcarpeta en el proyecto que contiene una copia de un intérprete determinado.

Los desarrolladores suelen crear un entorno virtual para cada proyecto

En cualquier momento, en el *Explorador de Soluciones*, expanda el nodo del proyecto, haga clic con el botón derecho en *Entornos de Python* y seleccione "Agregar entorno virtual".

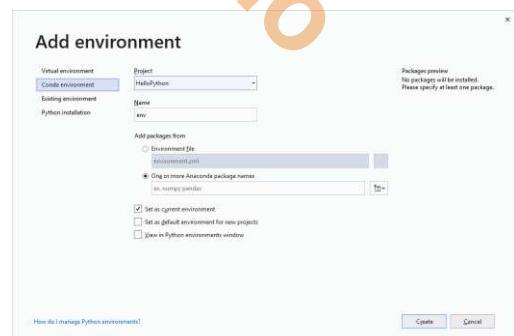
15

Entorno de Trabajo

Entornos de Trabajo en Visual Studio III

Entornos de coda, un entorno de *Conda* es el creado mediante la herramienta *Conda* o con la administración de *Conda* integrada en Visual Studio.

Cuando cree un entorno de *Conda*, asegúrese de especificar al menos una versión de Python o un paquete de Python con *environments.yml*, lo que garantiza que el entorno contiene un runtime de Python.



16

Entorno de Trabajo

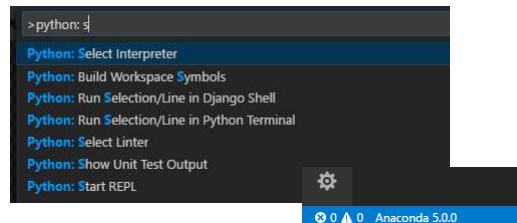
Entornos de Trabajo en Visual Studio Code

En Visual Studio Code debemos configurar su entorno de desarrollo de Python. Específicamente se requiere, Python 3.x y extensión VS Code Python.

Para crear un entorno virtual, usamos el comando: `python -m venv .venv`

Para seleccionar un entorno específico, usamos las teclas (Ctrl+Shift+P).

La barra de estado siempre muestra el intérprete actual.



17

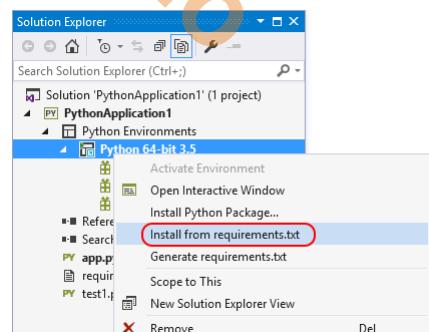
Entorno de Trabajo

Administración de los paquetes *requirements.txt*

Si comparte un proyecto con otros usuarios, usa un sistema de compilación o pretende copiar el proyecto en cualquier otra ubicación donde necesita restaurar un entorno, necesita especificar los paquetes externos que dicho proyecto requiere.

```
pip freeze > requirements.txt
```

registra la lista de paquetes actual del entorno en *requirements.txt*



18



19

Conceptos Básicos

Lenguajes de Programación Interpretados

Actualmente, empresas o grupos de trabajo que desarrollan lenguajes de programación están apostando por los lenguajes interpretados, como la plataforma .NET Framework, Java, Ruby, PHP, JavaScript, Python, ...

Un lenguaje de programación interpretado es aquel que el código fuente se ejecuta directamente, instrucción a instrucción.

20

Conceptos Básicos

Lenguajes de Programación Interpretados

Ventajas de los lenguajes interpretados:

- Multiplataforma, puede ser interpretado en varios sistemas operativos
- Portabilidad, el mismo programa puede llevarse a diferentes plataformas

La principales desventaja son:

- Velocidad en la ejecución del programa, porque cuando compilamos lo transformamos a código máquina que es en el que funciona el procesador.
- Es necesario que en la máquina donde va a funcionar tenga el interprete, ya sea como framework o como máquina virtual.

21

Conceptos Básicos

Intérprete de Python

El intérprete de Python habitualmente se instala en:

- Unix o Linux, /usr/local/bin/python3.9
- Windows, C:\Users\UserName\AppData\Local\Programs\Python\Python39\

El intérprete funciona de manera similar al Shell de Unix/Windows:

- Llamada estándar conectada a un terminal, lee y ejecuta comandos de manera interactiva
- Llamada con el nombre de un archivo como argumento, lee y ejecuta un script desde ese archivo
- Llamada estándar al nombre de un archivo, lee y ejecuta un script desde ese archivo

22

Conceptos Básicos

Intérprete de Python II

De forma predeterminada, los archivos fuente de Python se tratan como codificados en UTF-8.

Para declarar una codificación que no sea la predeterminada, se debe agregar una línea de comentario especial como la primera línea del archivo. La sintaxis es la siguiente:

```
# -*- coding: cp1252 -*-
```

23

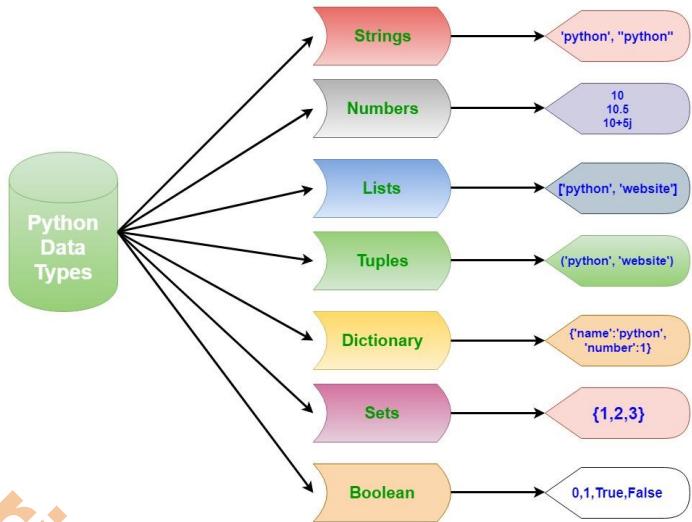
Conceptos Básicos

Introducción a Tipo de Datos

En Python tiene varios tipos de datos estándar disponibles por defecto en el interprete, como los tipos numéricos, secuencias, mapeos y conjuntos usados para agrupar otros valores.

Para el caso de las estructuras de datos se usan variables y constantes las cuales usan operadores para tratar los tipos de datos estándar.

24



01-Sintaxis-y-Estructuras-de-Control
-> 01.01-Variables-Tipo-de-Datos.py

25

Conceptos Básicos

Introducción a Tipo de Datos II

None

Se utiliza para indicar la ausencia de un valor en muchas situaciones, por ejemplo, se retorna desde funciones que no retornan nada explícitamente.

NotImplemented

Los métodos numéricos y los métodos de comparación enriquecidos deben devolver este valor si no implementan la operación para los operandos proporcionados.

No debe evaluarse en un contexto booleano.

26

Conceptos Básicos



COMENTARIOS

- # para comentar líneas de código
- """ """ para bloque de valores alfanuméricos que puede tener varias líneas

27

Conceptos Básicos



Operadores

Literales booleanos	Operadores numéricos	Operadores de comparación
Verdadero True	+ Suma	== igual
Falso False	- Diferencia o negación	!= distinto
Operadores lógicos	* Producto	> mayor
and Y lógico	/ División con decimales	>= mayor o igual
or O lógico	% Resto	< menor
not negación de verdad	// División entera (piso)	<= menor o igual
	** Potencia	is mismo objeto
		is not distinto objeto
Operadores a nivel de bits	Operadores de asignación	
& and	= asignación	
or	+= incremento	
~ not	-= decremento	
^ xor	Estos otros operadores pueden combinarse con asignación: /&///* & ^ >> <<	
>> desplazamiento a la derecha	No hay +++ - - ⊕	
<< desplazamiento a la izquierda		

28

Conceptos Básicos

Declaración de Variables

Python tiene **tipado dinámico**, asignando un tipo a los objetos en tiempo de ejecución en función de su valor, a diferencia del tipado estático donde las variables tienen un tipo.

Igualmente es un lenguaje **fueramente tipado**, donde el tipo de los objetos no cambia repentinamente.

Un string que contiene solo dígitos no se convierte mágicamente en un número. Cada cambio de tipo requiere una conversión explícita.

29

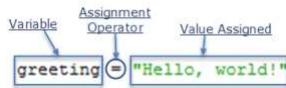
Conceptos Básicos

```
numero = 10
Numero = 20
saludo = "Hola Mundo"
print(numero)
print(Numero)
print(numero + Numero)
print("Saludo: " + saludo)
print(type(numero))
print(type(saludo))
```

10
20
30
Saludo: Hola Mundo
<class 'int'>
<class 'str'>

DECLARACIÓN DE VARIABLES

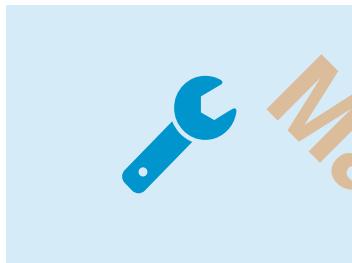
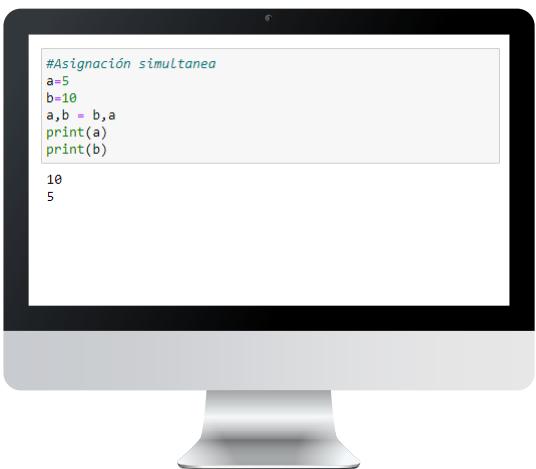
- No especificamos tipo
- Un variable puede ser cualquier palabra a la que asignamos un valor
- El nombre de la variable siempre comienza por guion bajo o letra, nunca por número
- Diferencia mayúsculas de minúsculas
- **print** para mostrar el contenido de la variable en la pantalla
- **type** para mostrar el tipo de la variable
- **del** para eliminar una variable



30

Asignación Simultánea

Descubre la potencia de la asignación simultánea entre variables

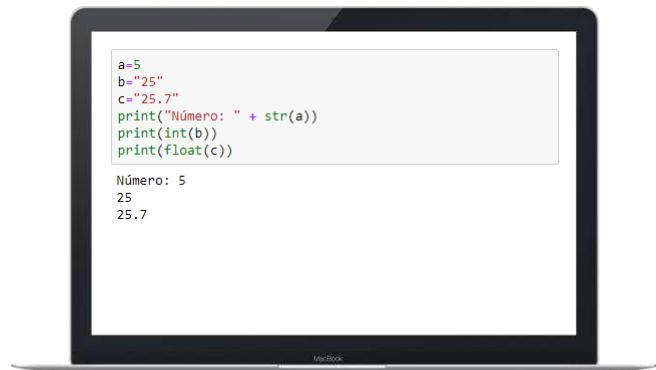


Intercambia los valores de las variables **a** y **b** para que **a** contenga 10 y **b** contenga 5.

01-Sintaxis-y-Estructuras-de-Control
-> 01.02-Variables-Asignación-Simultánea.py

31

Conceptos Básicos



CONVERSIONES

- **str(n)** convierte variables numéricas en texto
- **int(s)** convierte un texto en un valor numérico entero
- **float(s)** convierte un texto en un valor numérico con parte decimal

32

Conceptos Básicos



CADENA DE CARACTERES

- Las cadenas de caracteres son inmutables, no las podemos cambiar
- Si podemos cambiar el valor de una variables de tipo *String*
- Si podemos leer el contenido total o parcial de una cadena de caracteres
- **len(s)** retorna el número de caracteres de una cadena de texto
- [Descubre las funciones de las cadenas pulsando punto después del nombre de la variable](#)

33

Formateando Cadenas

Descubre las opciones de **format** para mostrar el contenido de las variables



Utiliza *Input()* para preguntar el nombre usuario al cual debes saludar de forma personalizada utilizando *format*.

Conceptos Básicos

```

from datetime import datetime

datenow1 = datetime.now().date()
print('Fecha:', datenow1)
datenow2 = datetime.now()
print('Fecha:', datenow2)
print('Año: ', datenow2.year)
print('Mes: ', datenow2.month)
print('Dia: ', datenow2.day)
print(f'Hora: {datenow2.hour}:{datenow2.minute}')

Fecha: 2020-02-23
Fecha: 2020-02-23 19:31:59.127833
Año: 2020
Mes: 2
Dia: 23
Hora: 19:31

```

FECHAS Y HORAS

- Para trabajar con variables que almacenan fechas y horas tenemos que importar el módulo `datetime`
- `now()` retorna la fecha actual incluyendo la hora
- `now().date()` retorna la fecha actual

36

Conceptos Básicos

```

from datetime import datetime

fecha = '10-11-2018'
obj = datetime.strptime(fecha, '%m-%d-%Y').date()
print(obj)
print(f'{obj.day}-{obj.month}-{obj.year}')

```

2018-10-11
11-10-2018

PARSE FECHAS

- `strptime()` convierte valores de texto en fecha

Python
strptime()
Convert string to datetime



37

Conceptos Básicos



FORMAT FECHAS

- **strftime()** formatea la representación de fechas

Python
strftime() Function
String Format to timestamp



38

Comodines para formatear y convertir Fechas

Utiliza los diferentes comodines para transformar texto en fechas y formatear la representación de fechas



01-Sintaxis-y-Estructuras-de-Control
-> 01.04-Variables-Fechas.py

%a	Short name of weekday	Fri
%A	Full name of Weekday	Friday
%w	Weekday as a number 0-6	0 is Sunday
%d	Day of month 01-31	01
%b	Month name, short version	Oct
%B	Month name, full version	October
%m	Month as a number 01-12	10
%y	Year, short version	18
%Y	Year, full version	2018
%H	Hour 00-23	00
%I	Hour 00-12	12
%p	AM/PM	AM
%M	Minute 00-59	38
%S	Second 00-59	01
%f	Microsecond 000000-999999	844628
%z	UTC offset	+0100
%Z	Timezone	UTC+01:00
%j	Day number of year 001-366	283
%U	Week number of year, 00-53	40 Sunday as the first day of the week
%W	Week number of year, 00-53	41 Monday as the first day of the week
%c	Local version of date and time	Wed Oct 10 03:38:01 2018
%x	Local version of date	10/10/18
%X	Local version of time	03:38:01
%%	A % character	%

39

Conceptos Básicos

```

import time

print("Time : ", time.time())
print(time.localtime(time.time()))
print("Año: ", time.localtime(time.time()).tm_year)
print("Minutos: ", time.localtime(time.time()).tm_min)
print('Milliseconds ', int(time.time() * 1000.0))
print(time.asctime(time.localtime(time.time())))

```

Time : 1582483728.6359816
 time.struct_time(tm_year=2020, tm_mon=2, tm_mday=23, tm_hour=19, tm_min=48, tm_sec=48, tm_wday=6, tm_yday=54, tm_isdst=0)
 Año: 2020
 Minutos: 48
 Milliseconds 1582483728636
 Sun Feb 23 19:48:48 2020

TIEMPO

- Para trabajar con variables que almacenan tiempo tenemos que importar el módulo `time`
- `time()` retorna el tiempo actual
- `localtime()` retorna una representación local del tiempo
- `asctime()` retorna una representación del tiempo alfanumérica, con fecha, hora y día de la semana

40

Zonas Horarias

Puede trabajar con fechas y horas de diferentes zonas horarias



```

from datetime import datetime, timedelta
from pytz import timezone
import pytz

print(pytz.all_timezones)
print(datetime.now(pytz.timezone('Asia/Tokyo')))
print(datetime.now(pytz.timezone('Europe/Madrid')))

```

2020-02-25 04:52:54.762609+09:00
 2020-02-24 20:52:54.764611+01:00

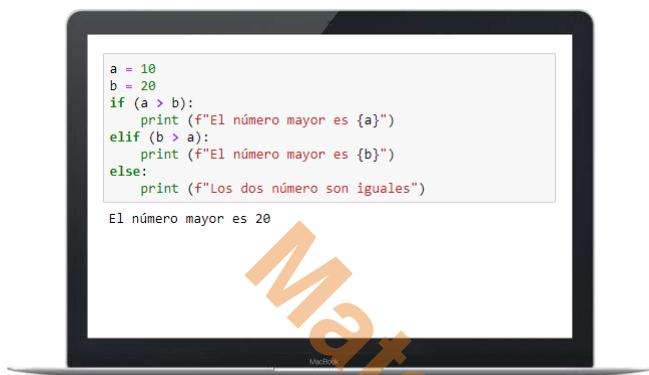
Instalamos el módulo que nos permite trabajar con zonas horarias en Python:

`pip install pytz`

01-Sintaxis-y-Estructuras-de-Control
-> 01.05-Variables-Zonas-Horarias.py

41

Conceptos Básicos

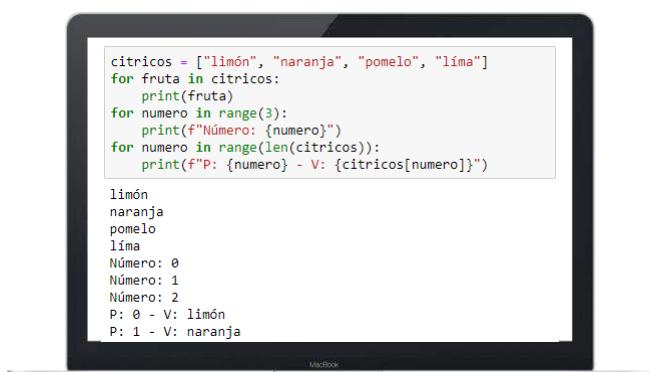


SENTENCIA DE DECISIÓN IF/ELSE

- Las sentencias de decisión determinan el flujo del programa tras evaluar una expresión de comparación
- if** determina la condición y el bloque de sentencias que cumple la condición
- else** marca el bloque de sentencias que no cumple la condición
- elif** simplifica un *else if* (entonces si)

42

Conceptos Básicos



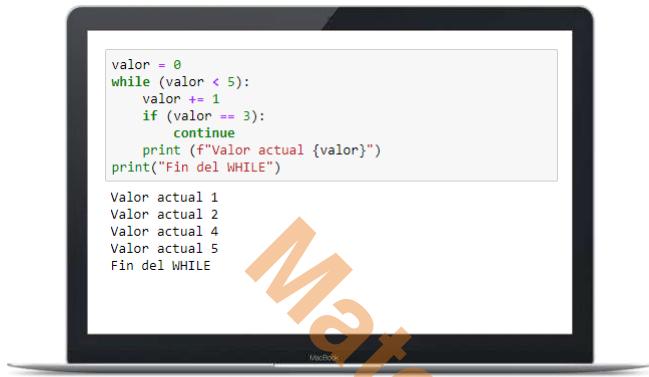
SENTENCIA DE REPETICIÓN FOR

- Las sentencias de repetición *for* ejecutan bloques de código de forma repetida
- La sentencia de repetición *for* se emplean para recorrer y trabajar con las colecciones
- range(n)** nos permite codificar contadores
- continue** continua con la siguiente operación de incremento
- break** finaliza el *for*



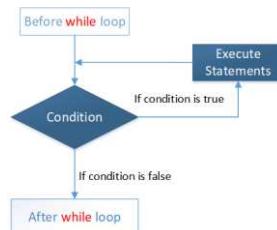
43

Conceptos Básicos



SENTENCIA DE REPETICIÓN WHILE

- Las sentencia de repetición *while* ejecutan bloques de código de forma repetida mientras se cumpla una condición
- continue** continua con la siguiente iteración
- break** finaliza el *while*



44

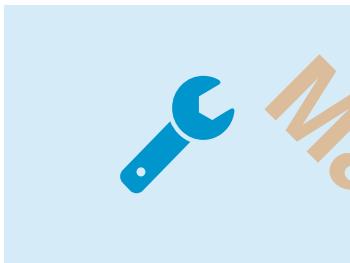
Conceptos Básicos

Sentencia de Repetición while

En Python tiene una palabra reservada llamada **while** que nos permite ejecutar ciclos, o bien secuencias periódicas que nos permiten ejecutar código múltiples veces.

Al igual que la sentencia **if**, la estructura **while** también puede combinarse con una sentencia **else**.

45

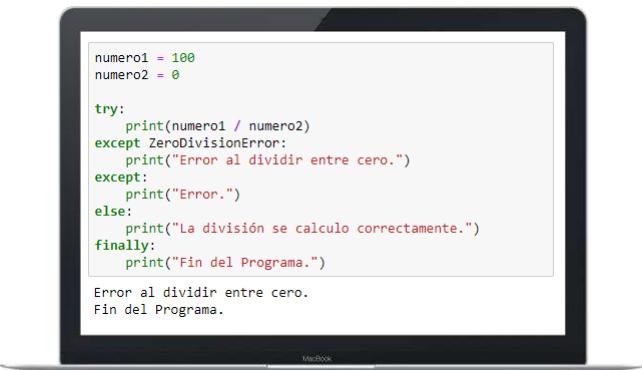


Al igual que la sentencia **if**, la estructura **while** también puede combinarse con una sentencia **else**.

01-Sintaxis-y-Estructuras-de-Control
-> 01.05-Variables-Zonas-Horarias.py

46

Conceptos Básicos



CONTROL DE EXCEPCIONES

- **try** permite controlar las excepciones producidas en un bloque de código
- **except** bloque de instrucciones que se ejecutan cuando se produce una excepción
- **else** bloque de instrucciones que se ejecutan al finalizar el **try** si no se produce un excepción
- **finally** bloque de instrucciones que se ejecutan siempre que finaliza el **try**, **except** o **else**

47

Conceptos Básicos

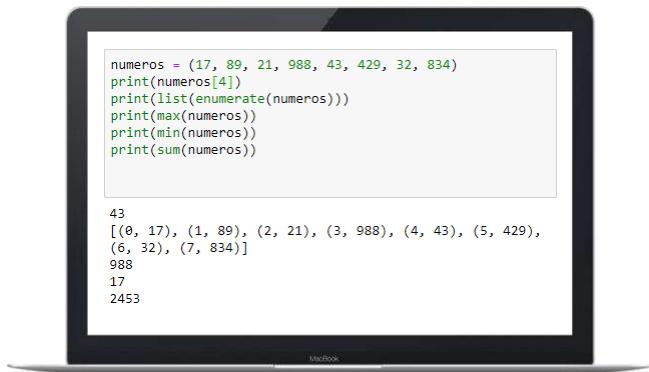


COLECCIONES, LISTAS

- Una *Lista* es una colección de elementos ordenados con un índice de base 0
- En una *Lista* se pueden añadir, eliminar y modificar elementos
- **extend** une dos Listas en una
- **sort** ordenar los elementos en base al valor
- **reverse** invierte el orden de los elementos en base a su índice
- **pop(i)** elimina el elemento de la posición i

48

Conceptos Básicos

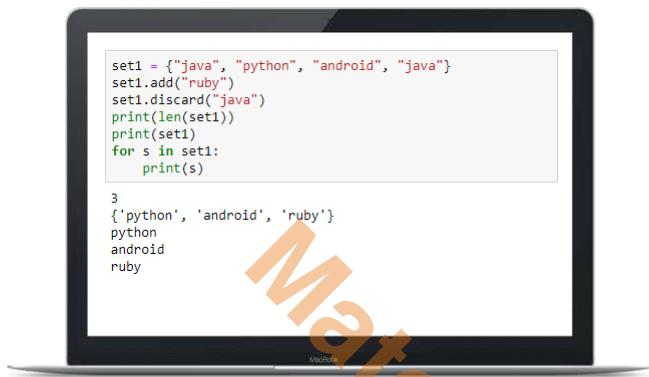


COLECCIONES, TUPLAS

- Una *Tupla* es una colección de elementos ordenados con un índice de base 0
- En una *Tupla* NO se pueden añadir, eliminar y modificar elementos

49

Conceptos Básicos

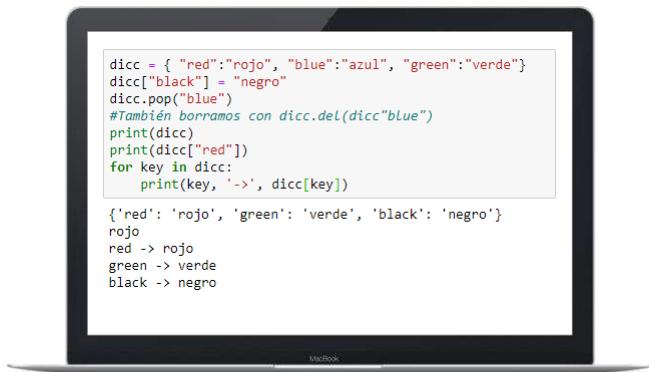


COLECCIONES, CONJUNTOS

- Un *Conjunto* es una colección de elementos sin índice, se dice que esta desordenado
- En un *Conjunto* se pueden añadir y eliminar elementos.
- Para acceder a los valores tenemos que recorrer la colección mediante un *for*

50

Conceptos Básicos



COLECCIONES, DICCIONARIOS

- Una *Diccionario* es una colección de elementos indexados por su clave
- Cada elemento del *Diccionario* se compone de clave y valor
- Cuando recorremos un *Diccionario* mediante *for* los valores que obtenemos son las claves
- **get(k, "")** muestra el valor de una clave o un valor alternativo si la clave no existe

51



{json}

JSON (JavaScript Object Notation) es un formato de texto sencillo para el intercambio y almacenamiento de datos.

Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.



```
import json

citos = ["limón", "naranja", "pomelo", "líma"]
citosJSON = json.dumps(citos)

print(citosJSON)

lista = json.loads(citosJSON)
print(lista[2])

["lim\u00f3n", "naranja", "pomelo", "l\u00edma"]
pomelo
```

52

Conceptos Básicos

GUI

Tk/Tcl ha sido durante mucho tiempo una parte integral de Python. Proporciona un conjunto de herramientas robusto e independiente de la plataforma para administrar ventanas.

Disponible para desarrolladores a través del paquete **tkinter** y sus extensiones, los módulos **tkinter.tix** y **tkinter.ttk**.

El **tkinter** paquete es una fina capa orientada a objetos encima de **Tk/Tcl**. Para usarlo **tkinter**, no necesita escribir código Tcl, pero deberá consultar la documentación de **Tk/Tcl**.

tkinter también es famoso por tener una apariencia obsoleta.

53

Conceptos Básicos

GUI, alternativas a tkinter

Kivy, acelerado de OpenGL ES 2 para la creación GUI. Es compatible con Windows, Mac OS, Linux, Android iOS y Raspberry Pi.

WxPython biblioteca de código abierto multiplataforma implementado como un módulo de extensión de Python. Con **WxPython**, puedes crear aplicaciones nativas para Windows, Mac OS y Linux.

PyGUI, framework multiplataforma para Windows, Mac OS y Linux. En comparación con algunos otros framework de GUI, **PyGUI** es, con mucho, el más simple y liviano de todos, ya que la API está puramente sincronizada con Python.

54

Conceptos Básicos

Django

Framework de código abierto para el desarrollo web, escrito en Python, que respeta el patrón de diseño conocido como *modelo–vista–controlador (MVC)*.

Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano *Django Reinhardt*.

En junio de 2008 fue anunciado que la recién formada Django Software Foundation se haría cargo de *Django* en el futuro.

Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés Don't Repeat Yourself).

55



56

Módulos y Paquetes

Módulos

En Python las diversas aplicaciones o funcionalidad se encuentran dentro de módulos y paquetes en forma de ficheros.

La sentencia **import** se utiliza para importar un módulo. Usted puede usar cualquier archivo de código Python como un módulo ejecutando esta sentencia en otro archivo de código Python.

La sentencia **import** tiene la siguiente sintaxis:

```
import os  
import re, datetime
```

57

Módulos y Paquetes



MÓDULOS

- Un *módulo* es un conjunto de funciones y clases definidas en un fichero que puedes reutilizar en diferentes aplicaciones de Python
- **import** es la palabra reservada de Python para importar y reutilizar las definiciones de otros ficheros (*módulos*)
- **from/import** para importar una definición específica de un módulo
- **from/import/as** para importar una definición específica de un módulo y crearla un alias

58

Módulos y Paquetes

Módulos II

El proceso de importación rellena estos atributos en cada objeto de módulo durante la carga:

- **__name__** se establece con el nombre completo del módulo
- **__path__** y **__file__** representa la ruta completa del módulo

59

Módulos y Paquetes

Módulos III

Podemos ejecutar un módulo como un script, pero con `__name__` con el valor de `"__main__"`.

```
python fibo.py <arguments>
$ python fibo.py 50

if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

Si el módulo se importa, ese código no se ejecuta.

60

Módulos y Paquetes

Módulos Estándar

La biblioteca estándar de Python es muy amplia, y ofrece una gran cantidad de módulos (escritos en C) que brindan acceso a las funcionalidades del sistema.

<https://docs.python.org/es/3/library/>

Además de la biblioteca estándar, existe un colección creciente de varios miles de componentes (abarcando módulos o programas individuales, paquetes o frameworks completos de desarrollo de aplicaciones), disponibles en el Python Package Index.

61

Módulos y Paquetes

Módulos VS Paquetes

Cualquier archivo de Python es un módulo, su nombre es el nombre base del archivo sin la extensión `.py`.

Un paquete es una colección de módulos de Python. Mientras que un módulo es un solo archivo de Python, un paquete es un directorio de módulos de Python que contiene un archivo adicional `__init__.py`, para distinguir un paquete de un directorio.

62

Módulos y Paquetes

Módulos VS Paquetes II

La distinción entre módulo y paquete parece mantenerse solo en el nivel del sistema de archivos.

Cuando importamos un módulo o un paquete, el objeto correspondiente creado por Python siempre es de tipo `module`.

```
└─ foo
    ├─ a.py
    ├─ b.py
    └─ c.py
```

```
import foo.a
from foo import b
import foo
```

Suponiendo que la carpeta `foo` está dentro de otra carpeta que forma parte de `PYTHONPATH`

```
└─ foo
    ├─ __init__.py
    ├─ a.py
    ├─ b.py
    └─ c.py
```

```
# Contenido de foo/__init__.py
from . import a, b, c
```

Entonces `import foo` ejecutaría `__init__.py`. Si `__init__.py` estuviera vacío, lo cual es válido, sería exactamente igual que si no estuviera.

63



pip

Gestor de paquetes que podemos incluir en el proyecto



pip (Python Index Package) es comando que nos permite gestionar los paquetes (módulos y librerías) que podemos incluir nuestros proyectos.

- Repositorio: <https://pypi.org/>

- Ubicaciones:

`/usr/local/bin/python3.9/Scripts
C:\Users\UserName\AppData\Local\Programs\Python\Python39\Scripts`

- Algunos comandos:

`pip --version
pip list
pip install to-camel-case`



64

Módulos y Paquetes

TensorFlow

Biblioteca de código abierto para cálculo numérico, usando como forma de programación grafos de flujo de datos.

Los nodos en el grafo representan operaciones matemáticas, mientras que las conexiones o links del grafo representan los conjuntos de datos multidimensionales (tensores).



Módulos y Paquetes

TensorFlow

TensorFlow se puede instalar en los siguientes sistemas de 64 bits con Python 3.6 a 3.9:

- Ubuntu 16.04 o versiones posteriores
- Windows 7 o versiones posteriores (con C++ redistribuible)
- macOS 10.12.6 (Sierra) o versiones posteriores (no tiene compatibilidad con GPU)

```
# Requires the latest pip
pip install --upgrade pip

# Current stable release for CPU and GPU
pip install tensorflow

# Or try the preview build (unstable)
pip install tf-nightly
```

The screenshot shows the TensorFlow homepage. The main heading reads "Plataforma de extremo a extremo de código abierto para el aprendizaje automático". Below it, there's a diagram illustrating various applications of machine learning, including a smartphone, a laptop, a server, and a car. At the bottom, there's a navigation bar with links for "TensorFlow", "Para JavaScript", "Para dispositivos móviles y de IoT", and "Para prod.". A footer note states: "La principal biblioteca de código abierto para enseñarte a desarrollar y entrenar modelos de AA. Comienza enseñándote y ejecuta notebooks de Colab directamente en tu navegador."

66

Módulos y Paquetes

scikit-learn

Biblioteca de software libre para aprendizaje automático para Python.

Incluye varios algoritmos de clasificación, regresión y análisis de grupos, diseñada para interoperar con las bibliotecas numéricas NumPy y científicas como SciPy.



67

Módulos y Paquetes

scikit-learn

El uso de un entorno tan aislado hace posible instalar una versión específica de scikit-learn con pip o conda y sus dependencias independientemente de cualquier paquete de Python previamente instalado.

```
# Requires the latest pip
pip install --upgrade pip

# Install scikit-learn
pip install -U scikit-learn

# To see which version and where scikit-learn is installed
python -m pip show scikit-learn

# To see all packages installed in the active virtualenv
python -m pip freeze
python -c "import sklearn; sklearn.show_versions()"
```

scikit-learn
Machine Learning in Python

Getting Started | Release Highlights for 0.24 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification
Identifying which category an object belongs to.
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experimental outcomes.
Algorithms: k-Means, spectral clustering, mean-shift, and more...

Dimensionality | **Model selection** | **Preprocessing**

68

Módulos y Paquetes

Keras

Biblioteca de Redes Neuronales de Código Abierto escrita en Python. Es capaz de ejecutarse sobre *TensorFlow*, *Microsoft Cognitive Toolkit* o *Theano*.

Está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de Aprendizaje Profundo. Sus fuertes se centran en ser amigable para el usuario, modular y extensible.



69

Módulos y Paquetes

La forma más sencilla de instalar Keras es con pip:

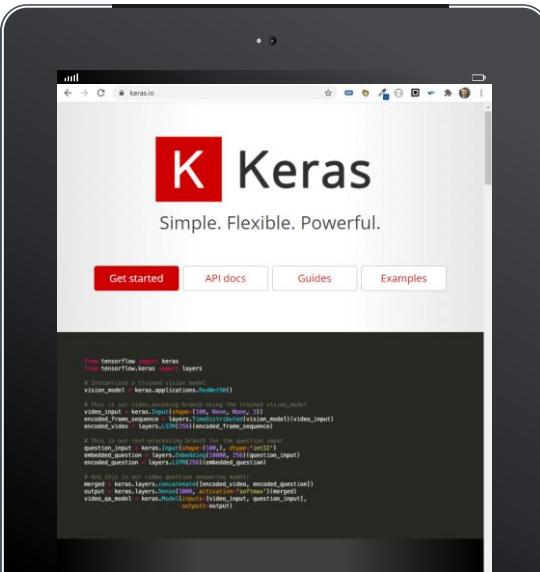
```
# Requires the latest pip
pip install --upgrade pip

# Install Keras is to via pip
pip install keras

# The Keras codebase is also available on GitHub
git clone https://github.com/keras-team/keras.git
```



Keras



70

Módulos y Paquetes

PyTorch

Es un paquete de Python diseñado para realizar cálculos numéricos haciendo uso de la programación de tensores. Además permite su ejecución en GPU para acelerar los cálculos.



71

Módulos y Paquetes

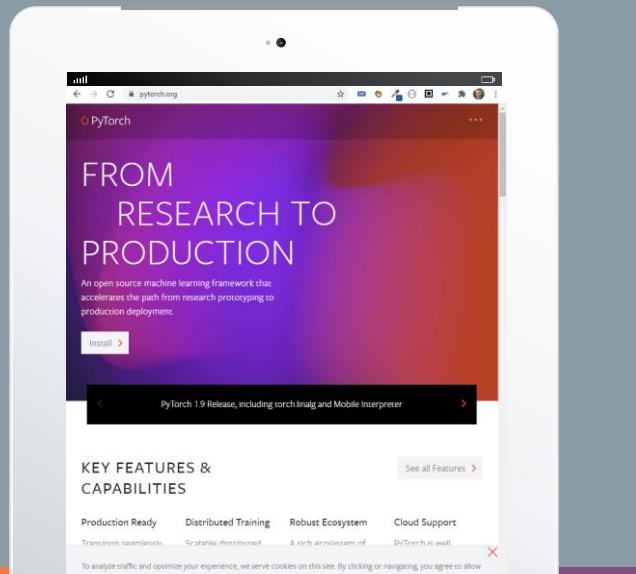
PyTorch

PyTorch se puede instalar en los siguientes sistemas:

- Ubuntu 13.03 o versiones posteriores
- Windows 7 o versiones posteriores
(se recomienda Windows 10)
- macOS 10.10 (Yosemite) o versiones posteriores

```
# Requires the latest pip
pip install --upgrade pip

# Install PyTorch is via pip
pip3 install torch==1.9.0+cu102
torchvision==0.10.0+cu102
torchaudio==0.9.0
-f https://download.pytorch.org/whl/torch_stable.html
```



72

Módulos y Paquetes

Numpy

Biblioteca para realizar cálculo numérico en Python. La usaremos principalmente porque nos permite crear y modificar matrices, y hacer operaciones sobre ellas con facilidad.



73

Módulos y Paquetes

pandas

Proporciona estructuras de datos. Pandas depende de Numpy, la librería que añade un potente tipo matricial a Python.

Los principales tipos de datos que pueden representarse con pandas son:

- Series: Estructura de una dimensión.
- DataFrame: Estructura de dos dimensiones (tablas).
- Panel: Estructura de tres dimensiones (cubos).



74

Módulos y Paquetes

pandas

Pandas proporciona herramientas que permiten:

- leer y escribir datos en diferentes formatos:
 - CSV
 - Microsoft Excel
 - Bases SQL
 - datos en formato HDF5
- seleccionar y filtrar tablas de datos en función de posición, valor o etiquetas
- fusionar y unir datos
- transformar datos aplicando funciones
- manipulación de series temporales
- hacer gráficas

75

Programación Orientada a Objetos



76

Programación Orientada a Objetos

Funciones

La definición de una función es una sentencia ejecutable que utiliza **def**, donde se vincula el nombre de la función con el espacio de nombres local.

Cuando la marcamos con **pass**, es nula, cuando es ejecutada no sucede nada.

```
def NOMBRE(LISTA_DE_PARAMETROS):
    """DOCSTRING_DE_FUNCION"""
    SENTENCIAS
    RETURN [EXPRESION]

def Saludo(arg):
    """El docstring de la función"""
    print("Hola", arg, "!")
    Saludo("Borja")

Hola Borja !
```

```
#Una función que no hace nada (aun)
def funcion(a): pass

#Una clase sin ningún método (aun)
class Persona: pass

type(funcion)
type(Persona)
funcion("M")
```

77

38

Programación Orientada a Objetos



FUNCIONES

- Una *Función* es un bloque de código que se ejecuta cuando es llamado
- **def** es la palabra reservada de Python para crear una funciones
- **return** es la palabra reservada de Python para retornar un resultado desde una *Función*
- Las colecciones y otros objetos son parámetros por referencia

78

Parámetros

Aprende a utilizar los parámetros de las funciones de Python



01-Sintaxis-y-Estructuras-de-Control
-> 01.01-Variables-Tipo-de-Datos.py

79

```
#Parámetros por posición
def resta(a, b):
    return a - b
```

20

```
#Parámetros por nombre
def resta(a, b):
    return a - b
```

resta(b=30, a=10)

-20

```
#Parámetros por defecto
def resta(a=None, b=None):
    if(a == None or b == None):
        print("Error, debes enviar dos números a la función")
        return
    return a - b
```

resta(30, 10)

resta()

Error, debes enviar dos números a la función



```
#Parámetros por indeterminados, args es una Tupla
def indeterminados(*args):
    for arg in args:
        print(arg)

indeterminados(5, "Hola Plone", [1,2,3,4,5])
5
Hola Plone
[1, 2, 3, 4, 5]

#Parámetros por indeterminados, kwargs es una Diccionario
def indeterminados(**kwargs):
    print(kwargs)

indeterminados(n=5, c="Hola Plone", l=[1,2,3,4,5])
{'n': 5, 'c': 'Hola Plone', 'l': [1, 2, 3, 4, 5]}

#Parámetros por indeterminados, kwargs es una Diccionario
def indeterminados(**kwargs):
    for kwarg in kwargs:
        print(kwarg, "=>", kwargs[kwarg])

indeterminados(n=5, c="Hola Plone", l=[1,2,3,4,5])
n => 5
c => Hola Plone
l => [1, 2, 3, 4, 5]
```

01-Sintaxis-y-Estructuras-de-Control
-> 01.01-Variables-Tipo-de-Datos.py

80

Programación Orientada a Objetos

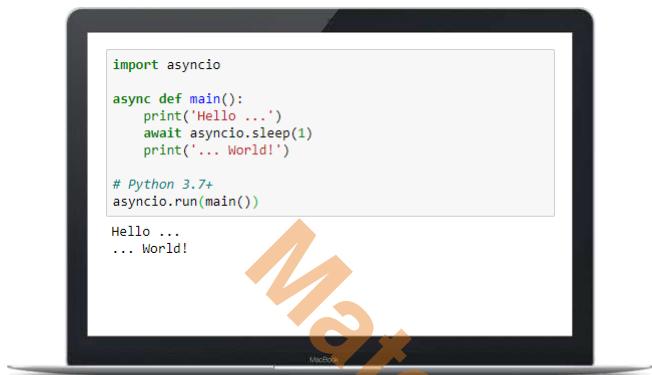
FUNCIONES LAMBDA

- Una *Función Lambda* es una función pequeña y anónima
- **lambda** es la palabra reservada de Python para crear una funciones lambda



81

Programación Orientada a Objetos



FUNCIONES ASÍNCRONAS

- **asyncio** es una biblioteca para escribir código utilizando la sintaxis `async / await`.
- Se utiliza como base para múltiples marcos asíncronos de Python que proporcionan redes y servidores web de alto rendimiento, bibliotecas de conexión de bases de datos, colas de tareas distribuidas, etc.

82

Programación Orientada a Objetos

Clases

Una definición de clase es una sentencia ejecutable que utiliza `class`.

La lista de herencia generalmente proporciona una lista de clases base, las clases sin una lista de herencia heredan, por defecto, de la clase base `object`.

Las clases en Python cuentan con múltiples métodos especiales, los cuales se encuentran entre dobles guiones bajos `__<metodo>__()`.

Los métodos especiales más utilizados son `__init__()`, `__str__()` y `__del__()`.

83

Programación Orientada a Objetos

Clases

```
class Persona:
    """Clase que representa una Persona"""
    cedula = "V-13458796"
    nombre = "Leonardo"
    apellido = "Caballero"
    sexo = "M"

    persona = Persona()
    print(type(persona))
    print(Persona.__name__)
    print(persona.__doc__)

<class '__main__.Persona'>
Persona
Clase que representa una Persona
```

```
class Persona:
    """Clase que representa una Persona"""
    cedula = "V-13458796"
    nombre = "Leonardo"
    apellido = "Caballero"
    sexo = "M"

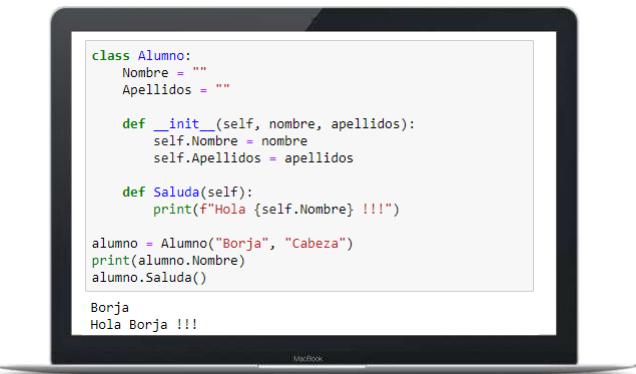
    def hablar(self, mensaje):
        """Mostrar mensaje de saludo de Persona"""
        return mensaje

    persona = Persona()
    print(type(persona.hablar))
    print(Persona.hablar.__name__)
    print(persona.hablar.__doc__)
    print(persona.hablar("Hola Borja."))

<class 'method'>
hablar
Mostrar mensaje de saludo de Persona
Hola Borja.
```

84

Programación Orientada a Objetos



```
class Alumno:
    Nombre = ""
    Apellidos = ""

    def __init__(self, nombre, apellidos):
        self.Nombre = nombre
        self.Apellidos = apellidos

    def Saluda(self):
        print(f"Hello {self.Nombre} !!!")

alumno = Alumno("Borja", "Cabeza")
print(alumno.Nombre)
alumno.Saluda()

Borja
Hello Borja !!!
```

CLASES

- Una *clase* es un constructor de objetos
- **class** es la palabra reservada de Python para crear una *clase*
- Las clases pueden contener variables, funciones y constructores
- Las funciones y los constructores pueden estar sobrecargados
- **__init__** es el nombre especial para la función constructor
- **self** es un parámetro especial que permite acceder al objeto mismo

85

Programación Orientada a Objetos

Herencia de Clases

La herencia simple se apoya en el uso de clases base para compartir sus atributos y comportamientos con otros clases derivadas.

Python permite la herencia múltiple, es decir, se puede heredar de múltiples clases.

Esto conlleva un problema, y es que si varias súper clases tienen los mismos atributos o métodos, la subclase sólo podrá heredar de una de ellas.

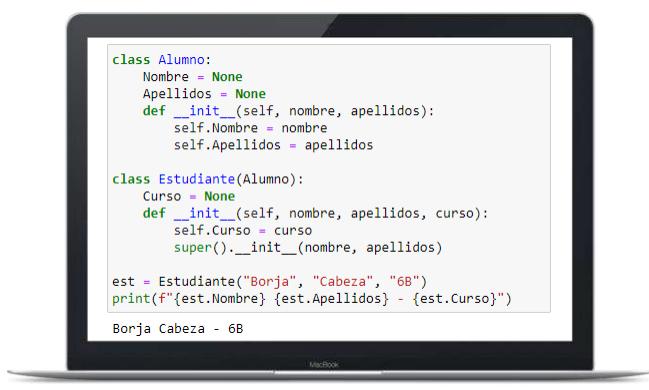
En estos casos Python dará prioridad a las clases más a la izquierda en el momento de la declaración de la subclase.

86

Programación Orientada a Objetos

HERENCIA DE CLASES

- La **herencia** nos permite definir una clase que hereda todos los métodos y propiedades de otra clase.
- La clase principal es la clase de la que se hereda, también llamada **clase base**.
- La clase hija es la clase que hereda de otra clase, también llamada **clase derivada**.
- La función **super()** nos permite acceder desde la clase derivada a funciones y propiedades de la clase base.



87

Programación Orientada a Objetos

Variables y Métodos Privados

En Python, por defecto, ***todos los elementos dentro de una clase serán públicos***, es decir, podrán ser consultados y/o modificados fuera de la clase.

Y, ¿Qué pasa si queremos definir un elemento como privado? En esos casos, por convención, haremos uso del guión bajo (_).

De esta forma le indicamos a cualquier desarrollador que, ese elemento en particular, debe tratarse como privado, y no debe ser expuesto ni modificado externamente.

Ojo, esto es una convención, ***el intérprete en ningún momento negará el acceso a dicho elemento***.

88

Programación Orientada a Objetos

Variables y Métodos Privados II

Al utilizar un doble guion bajo, ya sea para una atributo o algún método, el intérprete de Python re nombra al elemento para evitar colisiones con las subclases.

Aunque la idea original era esa, evitar colisiones, muchos desarrolladores utilizan el doble guion bajo (_) para prevenir accesos no autorizados.

Utilizando **dir** podemos listar todos los atributos de la clase y comprobar que los atributos con doble guion bajo fueron renombrados.

89

Programación Orientada a Objetos

```

class Demo:
    def __secret(self):
        print('Nadie puede saber!')

    def public(self):
        return self.__secret()

class Child(Demo):
    def __secret(self):
        print('No puedo contarte!')

demo = Demo()
print(demo.public())
print(demo.__secret())

child = Child()
print(child.public())
print(child.__secret())

print(dir(demo))
print(demo.__dict__)

```

Nadie puede saber!
None
Nadie puede saber!
None
['__Demo__secret', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'public']
Nadie puede saber!
None

90

Programación Orientada a Objetos

Generadores

Los generadores son clases o funciones que no sólo tiene una sentencia **return**, sino que devuelve más un valor haciendo uso de **yield**.

```

def funcion():
    return 5

def generador():
    yield 5

print(funcion())
print(generador())

print("Next:", next(generador()))

for i in generador():
    print("For:", i)

5
<generator object generador at 0x00000253373778C8>
Next: 5
For: 5

```

91

Programación Orientada a Objetos



GENERADORES

- Son clases o funciones que retorna más de un valor utilizando la palabra reservada **yield**.
- Cuando invocamos la función nos retorna un objeto de tipo **generador**.
- Podemos recorrer los valores del generador utilizando el método **next()** o una sentencia **for**.

92

Programación Orientada a Objetos

Generadores II

Los generadores también pueden ser creados de una forma mucho más sencilla y en una sola línea de código. Su sintaxis es similar a las *list comprehension*, pero cambiando el corchete [] por paréntesis () .

```

lista = [2, 4, 6, 8, 10]
al_cuadrado = [x**2 for x in lista]
al_cuadrado_generador = (x**2 for x in lista)

def generador(numeros):
    for x in numeros:
        yield x**2

print(type(al_cuadrado), al_cuadrado)
print(type(al_cuadrado_generador), list(al_cuadrado_generador))
print(type(generador(lista)), list(generador(lista)))

<class 'list'> [4, 16, 36, 64, 100]
<class 'generator'> [4, 16, 36, 64, 100]
<class 'generator'> [4, 16, 36, 64, 100]

```

93



94

Patrones de Diseño

Definición

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución ya probada y documentada para problemas en el desarrollo de software en contextos similares.

Elementos de un patrón:

- su nombre y el problema (cuando aplicar un patrón)
- la solución (descripción abstracta del problema y la solución)
- las consecuencias (costos y beneficios)



95

Patrones de Diseño

Clasificación

Clasificación de Patrones de Diseño:

- **Creacionales:** solucionan problemas de creación de instancias ayudando a encapsular y abstraer dicha creación.
- **Estructurales:** solucionan problemas de composición cómo las clases se agrupan.
- **Comportamiento:** ofrecen soluciones respecto a la interacción entre clases, así como los algoritmos que encapsulan.



Creacionales



Estructurales



Comportamiento

96

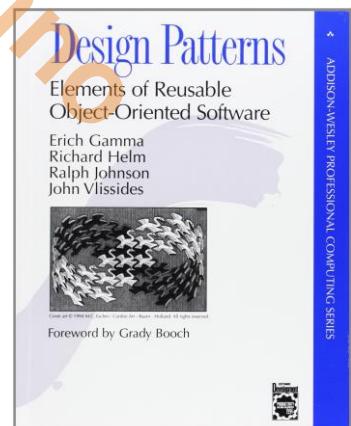
Patrones de Diseño

Clasificación II

En 1994 el grupo Gang of Four (GoF) compuesto por los programadores de IBM:

- Erich Gamma
- Richard Helm
- Ralph Johnson
- John Vlissides

publicaron el libro ***Design Patterns*** donde se recogen 23 patrones de diseño más comunes.



97

Patrones de Diseño

Modelo-Vista-Controlador

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución ya probada y documentada para problemas en el desarrollo de software en contextos similares.

Elementos de un patrón:

- su nombre y el problema (cuando aplicar un patrón)
- la solución (descripción abstracta del problema y la solución)
- las consecuencias (costos y beneficios)

98

Patrones de Diseño

Command

Convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud.

Esta transformación te permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.

Es muy común en el código Python. La mayoría de las veces se utiliza como alternativa a las retrolllamadas (*callbacks*) para parametrizar elementos UI con acciones. También se utiliza para poner tareas en cola, realizar el seguimiento del historial de operaciones, etc.

99

Patrones de Diseño

Command



Terminología utilizada en el diseño de un *Command*.

- **Receptor**, el objeto que recibirá y ejecutará el comando. Las clases de receptor contienen una lógica empresarial. Saben realizar todo tipo de operaciones, asociadas a la realización de una solicitud. De hecho, cualquier clase puede servir como Receptor.
- **Invocador**, el objeto que envía el comando al receptor. El invocador está asociado con uno o varios comandos. Envía una solicitud al comando, por ejemplo, un botón.
- **Objeto de Comando**, en sí mismo, un objeto, que implementa un método de ejecución o acción, y contiene toda la información necesaria para ejecutarlo.
- **Cliente**, la aplicación o componente que conoce el receptor, el invocador y los comandos.

100

Patrones de Diseño

Observador



El patrón *Observer* proporciona una forma de suscribirse y cancelar la suscripción a estos eventos para cualquier objeto que implementa una interfaz suscriptora.

El patrón *Observer* es bastante habitual en el código Python, sobre todo en los componentes GUI. Proporciona una forma de reaccionar a los eventos que suceden en otros objetos, sin acoplarse a sus clases.

101

Patrones de Diseño

Fachada o Facade



Patrón de diseño estructural que proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.

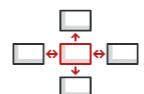
Una fachada es una clase que proporciona una interfaz simple a un subsistema complejo que contiene muchas partes móviles.

Una fachada puede proporcionar una funcionalidad limitada en comparación con trabajar directamente con el subsistema. Sin embargo, tan solo incluye las funciones realmente importantes para los clientes.

102

Patrones de Diseño

Mediador



Patrón de diseño de comportamiento que te permite reducir las dependencias caóticas entre objetos. El patrón restringe las comunicaciones directas entre los objetos, forzándolos a colaborar únicamente a través de un objeto mediador.

El patrón *Mediator* sugiere que detengas toda comunicación directa entre los componentes que quieras hacer independientes entre sí.

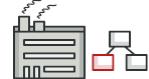
En lugar de ello, estos componentes deberán colaborar indirectamente, invocando un objeto mediador especial que redireccione las llamadas a los componentes adecuados.

Como resultado, los componentes dependen únicamente de una sola clase mediadora, en lugar de estar acoplados a decenas de sus colegas.

103

Patrones de Diseño

Factory



Patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

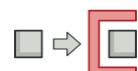
El patrón *Factory Method* sugiere que, en lugar de llamar al operador new para construir objetos directamente, se invoque a un método fábrica especial.

No te preocupes: los objetos se siguen creando a través del operador new, pero se invocan desde el método fábrica. Los objetos devueltos por el método fábrica a menudo se denominan productos.

104

Patrones de Diseño

Proxy



Permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.

El patrón Proxy sugiere que crees una nueva clase proxy con la misma interfaz que un objeto de servicio original. Si necesitas ejecutar algo antes o después de la lógica primaria de la clase, el proxy te permite hacerlo sin cambiar esa clase.

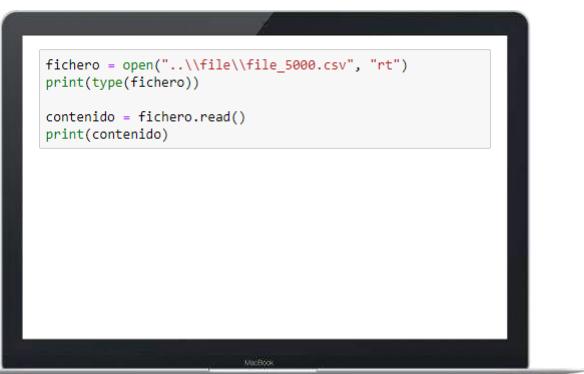
Ya que el proxy implementa la misma interfaz que la clase original, puede pasarse a cualquier cliente que espere un objeto de servicio real.

105



106

Archivos CSV



FICHEROS DE TEXTO

- **open** es la palabra reservada de Python para trabajar con ficheros
- Requiere dos parámetros, el ruta o del fichero y el modo de apertura
- Devuelve un objeto que representa el fichero con métodos como *read*, *write*, *readline* que nos permiten trabajar con el fichero para leer, escribir y modificar su contenido

107

Archivos CSV

```

bytes1 = "En un lugar de la Mancha ...".encode()
bytes2 = b"En un lugar de la Mancha ..."
bytes3 = bytes("En un lugar de la Mancha ...", "utf-8")
print(bytes1)
print(bytes2)
print(type(bytes3))
for b in bytes3:
    print(b, end=' ')

```

b'En un lugar de la Mancha ...'
b'En un lugar de la Mancha ...'
b'En un lugar de la Mancha ...'
<class 'bytes'>
69 110 32 117 110 32 188 117 103 97 114 32 100 101 32 108
97 32 77 97 110 99 104 97 32 46 46 46

FICHEROS DE BYTES

- **b** delante del limitador de una cadena de caracteres la convierte en un array de bytes
- **bytes()** transforma una cadena de caracteres en un array de bytes
- **encode()** transforma una cadena de caracteres en un array de bytes
- **decode()** transforma un array de bytes en una cadena de caracteres

108

Archivos CSV

```

import pickle

texto = "Nuevo contenido para el fichero."

fichero = open("demo.bin", "wb")
pickle.dump(texto, fichero)
fichero.close()

fichero = open("demo.bin", "rb")
texto = pickle.load(fichero)
fichero.close()

print(texto)

```

Nuevo contenido para el fichero.

FICHEROS BINARIOS

- Los **pickle** de Python son muy útiles ya que representan un objeto Python como una cadena de bytes
- Se pueden hacer multitud de cosas con los bytes, como por ejemplo, almacenarlos en un archivo, almacenarlos en una base de datos, transferirlos a través de una red, ...
- Para utilizar los **pickle** en Python, utilizaremos el módulo Pickle

109

Convertiendo a Fechas

Utiliza los diferentes comodines para transformar texto en fechas



'r' Este es el modo predeterminado. Abre archivo para leer.

'w' Este modo Abre el archivo para escribir.
Si el archivo no existe, crea un nuevo archivo.
Si el archivo existe, trunca el archivo.

'x' Crea un nuevo archivo.
Si el archivo ya existe, la operación falla.

'a' Abre el archivo en modo agregar.
Si el archivo no existe, crea un nuevo archivo.

't' Este es el modo predeterminado. Se abre en modo texto.

'b' Esto se abre en modo binario.

'+' Esto abrirá un archivo para leer y escribir (actualizar)

Crear ficheros, leer datos y añadir datos, finalmente eliminar el fichero.

Trabajaremos con ficheros de texto, bytes y binarios.

05-Trabajando-con-Ficheros
>> 05.01-Ficheros-Texto.py
>> 05.02-Ficheros-Bytes.py
>> 05.03-Ficheros-Binario.py

110

Archivos CSV

Archivos CSV

Un archivo CSV (valores separados por comas) permite que los datos sean guardados en una estructura tabular con una extensión .csv.

Los archivos CSV han sido usados de manera extensiva en aplicaciones porque son considerados muy fáciles de procesar.

El módulo CSV tiene varias funciones y clases disponibles para leer y escribir CSVs, y estas incluyen:

- función csv.reader
- función csv.writer
- clase csv.Dictwriter
- clase csv.DictReader

111

Archivos CSV

```

import csv

with open('sample.csv', newline='') as File:
    reader = csv.reader(File)
    for row in reader:
        print(row)

with open('sample.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row['Product'], row['Country'])
        print(row)

results = []
with open('sample.csv') as File:
    reader = csv.DictReader(File)
    for row in reader:
        results.append(row)
print(results)

with open('names.csv', 'w', newline='') as csvfile:
    fieldnames = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'first_name': 'Baked', 'last_name': 'Beans'})
    writer.writerow({'first_name': 'Lovely', 'last_name': 'Spam'})
    writer.writerow({'first_name': 'Wonderful', 'last_name': 'Spam'})

```

112

Archivos CSV

Ficheros CSV con Panda

La función `read_csv()` permite leer un CSV como y terminar con un *dataframe* de pandas, también se puede utilizar la función `read_table()` para leer CSV al declarar el delimitador como coma.

```

import pandas as pd

df = pd.read_csv("names.csv")
print(df)

import pandas as pd

df2 = pd.read_csv("name.csv", header=None,
names=["Nombre", "Apellidos"])
print(df2)

```

113



114

Acceso a Base de Datos

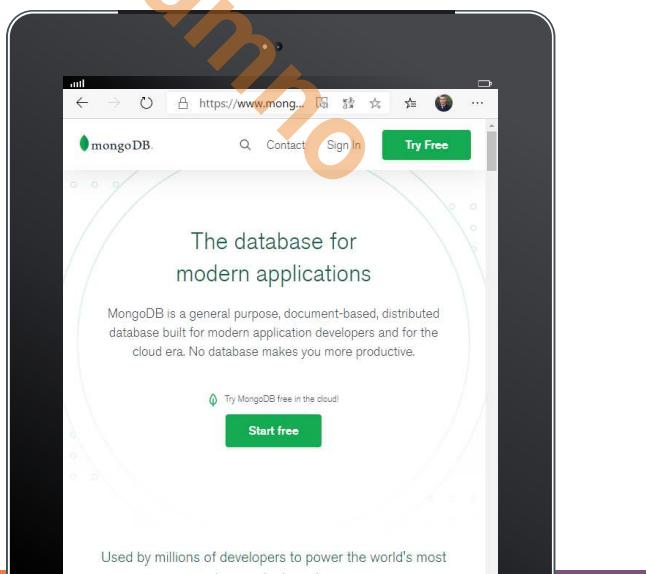
Instalación de  mongoDB

Descargamos e instalamos la base de datos mongoDB

<https://www.mongodb.com/>

Descargamos e instalamos la herramienta Robot 3T

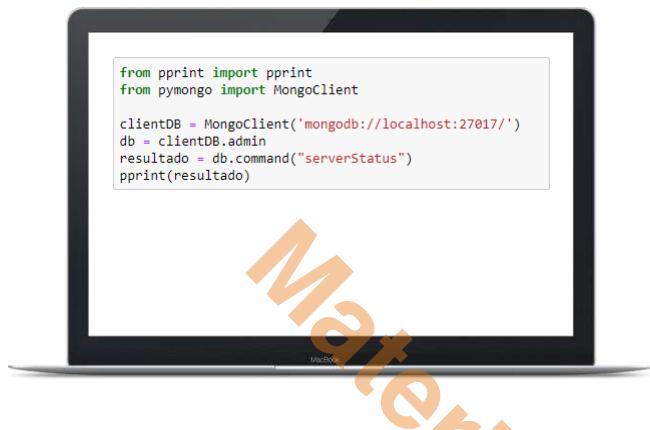
<https://robomongo.org/>



Used by millions of developers to power the world's most innovative products and services

115

Acceso a Base de Datos



CONEXIÓN MONGODB

- **MongoClient** es el objeto que representa la conexión a la base de datos
- Desde el objeto cliente podemos acceder a todas las bases de datos
- Desde la base de datos a todas las colecciones
- Desde las colecciones podemos buscar, crear, modificar y eliminar documentos



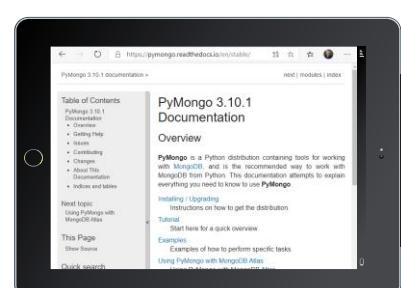
116



pymongo (Python Mongo) es el paquete que me permite conectarme con mongoDB y trabajar con sus base de datos, colecciones y documentos.

Instalación:

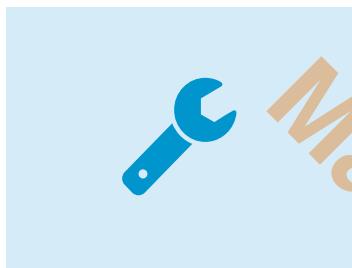
```
pip install pymongo
```



Documentación:

<https://api.mongodb.com/>

117



06-Acceso-Base-de-Datos
-> 06_01-MongoDB.py

118

Sigue las instrucciones para probar los siguientes objetos y métodos de pymongo:

- **cliente.db.command**
Ejecuta comando sobre la base de datos.
- **cliente.db.collection.find**
Realiza búsqueda de documentos con o sin filtros
- **cliente.db.collection.find_one**
Realiza búsqueda de documentos con o sin filtros
- **cliente.db.collection.count_documents**
Cuenta documentos coincidentes con los valores del filtros
- **cliente.db.collection.insert_one**
Inserta un documento en la colección
- **cliente.db.collection.update_one**
Modifica el primer documentos coincidente con los valores del filtros
- **cliente.db.collection.delete_one**
Elimina el primer documentos coincidente con los valores del filtros

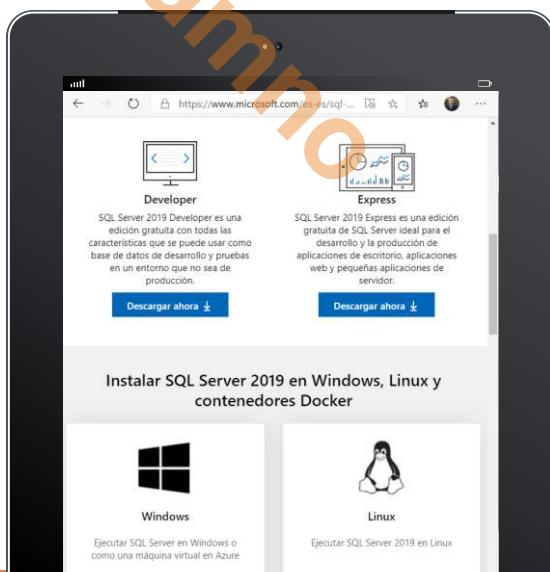
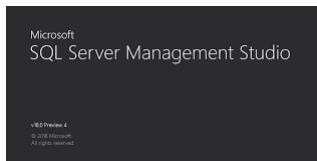
Acceso a Base de Datos

Instalación de Microsoft SQL Server

Descargamos e instalamos Microsoft SQL Server

<https://www.microsoft.com/es-es/sql-server/sql-server-downloads>

Descargamos Instalamos la última versión de Microsoft Management Studio



119

Acceso a Base de Datos



CONEXIÓN SQL SERVER

- **pymssql** es el objeto que representa la conexión a la base de datos
- Desde el objeto conexión creamos un cursor que nos permite ejecutar comando de Transat-SQL
- Si la ejecución del comando retorna filas podemos recorrerlas mediante un *while*
- Las posiciones de los campos en la fila nos permiten acceder a la información del registro
- **fetchone** nos posiciona en el siguiente registro

120



pymssql es el paquete que me permite conectarme con Microsoft SQL Server y trabajar con sus base de datos, tablas y registros



Instalación:

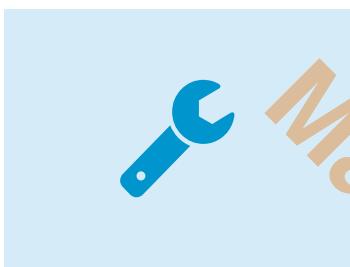
`pip install pymssql`



Documentación:

<https://pythonhosted.org/>

121



06-Acceso-Base-de-Datos
>> 06_02-SQLServer.py

122

Sigue las instrucciones realizar consultas, modificaciones, inserciones y borrado de registros en SQL Server:

```
import pymssql

conexion = pymssql.connect(server='localhost',
                            user='developer',
                            password='Pa$$w0rd',
                            database='Northwind')

cursor = conexion.cursor()
Para retornar diccionarios en Lugar de Tupla usamos conexion.cursor(as_dict=True)

cursor.execute("SELECT * FROM Customers")

row = cursor.fetchone()
while row:
    print (f'{row[0]}# {row[1]} - {row[6]} ({row[9]})')
    row = cursor.fetchone()

cursor.close()
conn.close()
```

Acceso a Base de Datos



SQLite es una biblioteca C que proporciona una base de datos liviana basada en disco que no requiere un proceso de servidor separado y permite acceder a la base de datos utilizando una variante no estándar del lenguaje de consulta SQL.

Algunas aplicaciones pueden usar SQLite para el almacenamiento interno de datos.

También es posible crear un prototipo de una aplicación utilizando SQLite y luego transferir el código a una base de datos más grande como PostgreSQL u Oracle.

The screenshot shows a web browser displaying the Python Standard Library documentation for the sqlite3 module. The page title is "11.13. sqlite3 – DB-API 2.0 interface for SQLite databases". It includes a table of contents with sections like "11.13.1. Module", "11.13.2. Connection Objects", and "11.13.4. Row Objects". The main content area describes SQLite as a C library for disk-based databases and its use in Python applications. It also mentions the sqlite3 module's connection object and its use for internal data storage.

123



```

import sqlite3
conn = sqlite3.connect('demo.db')

c = conn.cursor()

# Crear table
c.execute("""CREATE TABLE alumnos
            (id, nombre, apellidos, curso, notas)""")

# Insert una fila de datos
c.execute("INSERT INTO alumnos VALUES ('A00', 'Borja', 'Cabeza', '1A', '')")

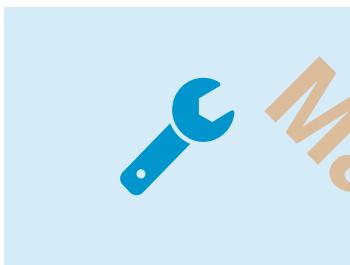
# Insertar varias filas de datos
listaAlumnos = [('AR0', 'Ana', 'Trujillo', '1A', ''),
                 ('Z02', 'Jimena', 'Sanz', '2A', '')]

c.executemany('INSERT INTO alumnos VALUES (?,?,?,?,?)', listaAlumnos)

# Guardar y confirmar los cambios (commit)
conn.commit()

# También podemos cerrar la conexión si hemos terminado con ella.
# Solo asegúrese de que se hayan confirmado los cambios o se perderán.
conn.close()

```



06-Acceso-Base-de-Datos
-> 06.03-SQLite.py

124



```

import sqlite3

conn = sqlite3.connect('demo.db')
c = conn.cursor()

# Recuperar registros
id = ('A00',)
c.execute('SELECT * FROM alumnos WHERE id=?', id)
r = c.fetchone()

print(r)
print(type(r))

# Mostar todos los campos
for dato in r:
    print (dato)

# Mostar varios registros
c.execute('SELECT * FROM alumnos')
r = c.fetchone()
while (r):
    print(r)
    r = c.fetchone()

conn.close()

```



06-Acceso-Base-de-Datos
-> 06.03-SQLite-2.py

125



126

Biblioteca NumPy

Numpy

Biblioteca para realizar cálculo numérico en Python. La usaremos principalmente porque nos permite crear y modificar matrices, y hacer operaciones sobre ellas con facilidad.

Fue creado en 2005 por Travis Oliphant. Es un proyecto de código abierto y puedes usarlo libremente. NumPy son las siglas de Numerical Python.



127

Biblioteca NumPy

¿Por qué NumPy?

Las matrices de NumPy se almacenan en un lugar continuo en la memoria a diferencia de las listas de Python, por lo que los procesos pueden acceder a ellos y manipularlos de manera muy eficiente.

Este comportamiento se denomina localidad de referencia en informática.

También está optimizado para trabajar con las últimas arquitecturas de CPU. NumPy es una biblioteca de Python y está escrito parcialmente en Python, pero la mayoría de las partes que requieren un cálculo rápido están escritas en C o C++.

128

Biblioteca NumPy

¿Por qué NumPy?

Las matrices de NumPy se almacenan en un lugar continuo en la memoria a diferencia de las listas de Python, por lo que los procesos pueden acceder a ellos y manipularlos de manera muy eficiente.

Este comportamiento se denomina localidad de referencia en informática.

También está optimizado para trabajar con las últimas arquitecturas de CPU. NumPy es una biblioteca de Python y está escrito parcialmente en Python, pero la mayoría de las partes que requieren un cálculo rápido están escritas en C o C++.

129

Biblioteca NumPy

Creación de Matrices

Las matrices de NumPy se almacenan en un lugar continuo en la memoria a diferencia de las listas de Python, por lo que los procesos pueden acceder a ellos y manipularlos de manera muy eficiente.

Este comportamiento se denomina localidad de referencia en informática.

También está optimizado para trabajar con las últimas arquitecturas de CPU. NumPy es una biblioteca de Python y está escrito parcialmente en Python, pero la mayoría de las partes que requieren un cálculo rápido están escritas en C o C++.

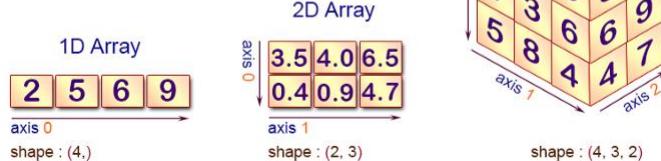
130

Biblioteca NumPy

Dimensiones de las Matrices

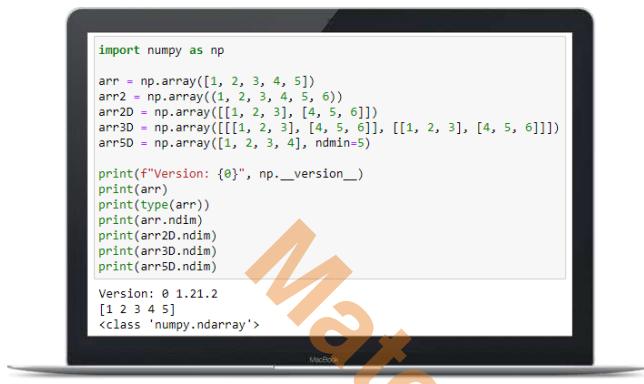
Un array es una estructura de datos de un mismo tipo organizada en forma de tabla o cuadrícula de distintas dimensiones.

Las dimensiones de un array también se conocen como ejes.



131

Biblioteca NumPy



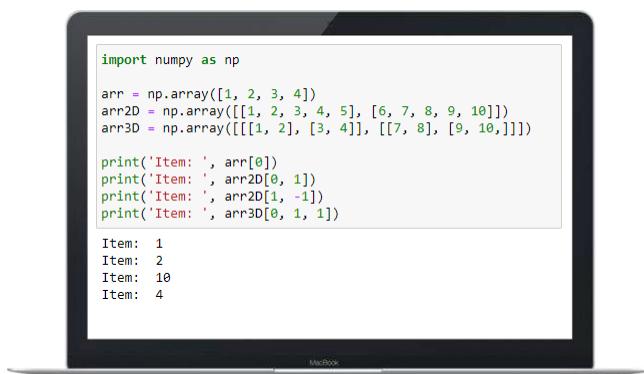
CREACIÓN DE ARRAYS

- **array()** es el método que utilizamos para crear matrices, pudiendo indicar las dimensiones mediante el atributo **ndmin**
- **ndim** es la propiedad de la matriz que nos indica las dimensiones de la misma
- El atributo **dtype** es la propiedad que nos permite especificar el tipo de los datos contenidos en la matriz
- La propiedad **itemsize** devuelve el número de bytes que ocupa cada dato de la matriz

```
x = np.array([1,2,3,4,5], dtype = np.float32)
print x.itemsize
```

132

Biblioteca NumPy

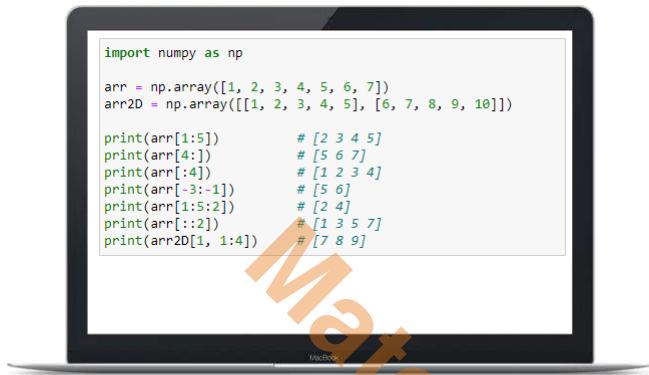


INDEXACIÓN

- Puede acceder a un elemento de matriz consultando su número de índice.
- Los índices en las matrices comienzan con 0
- Utilice la indexación negativa para acceder a una matriz desde el final.

133

Biblioteca NumPy

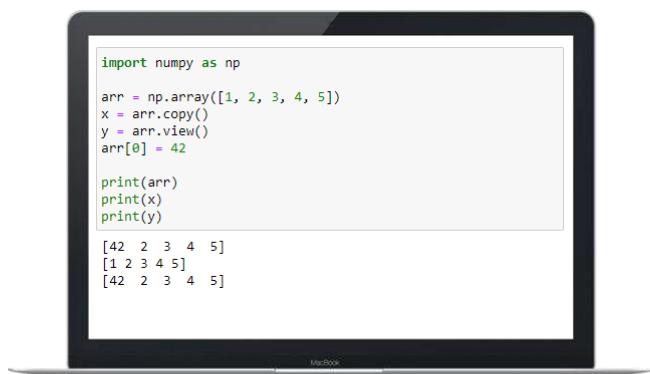


SLICING

- Cortar en Python significa tomar elementos de un índice a otro índice.
- Para cortar indicamos `.[start:end]`
- También podemos indicar: `.[start:end:step]`
- Si no pasamos el `start` se considera 0
- Si no pasamos `end` se considera la longitud de la matriz en esa dimensión
- Si no pasamos el `step` se considera 1

134

Biblioteca NumPy

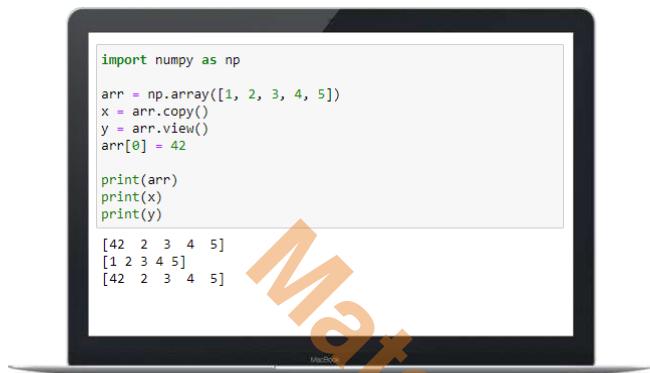


COPIANDO

- La copia es propietaria de los datos y cualquier cambio realizado en la copia no afectará la matriz original, y cualquier cambio realizado en la matriz original no afectará la copia
- La vista no es propietaria de los datos y cualquier cambio realizado en la vista afectará a la matriz original, y cualquier cambio realizado en la matriz original afectará la vista.

135

Biblioteca NumPy

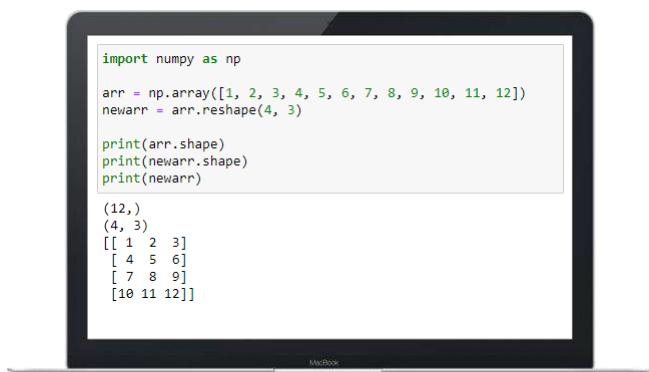


COPIANDO

- La copia es propietaria de los datos y cualquier cambio realizado en la copia no afectará la matriz original, y cualquier cambio realizado en la matriz original no afectará la copia
- La vista no es propietaria de los datos y cualquier cambio realizado en la vista afectará a la matriz original, y cualquier cambio realizado en la matriz original afectará la vista

136

Biblioteca NumPy

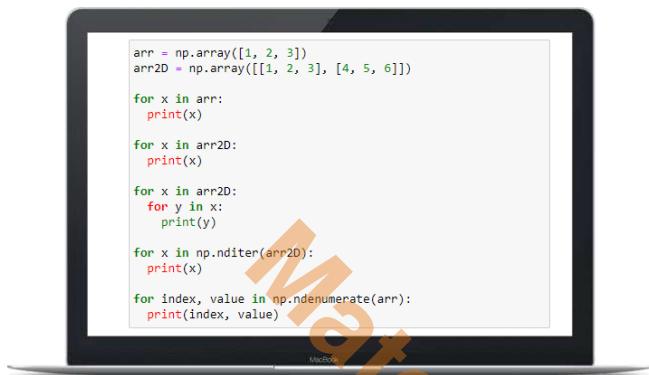


SHAPE Y RESHAPE

- **shape** es la propiedad que nos devuelve el número de elementos en cada dimensión
- **reshape()** lo utilizamos para cambiar la forma de una matriz, indicando el número de elementos en cada dimensión

137

Biblioteca NumPy

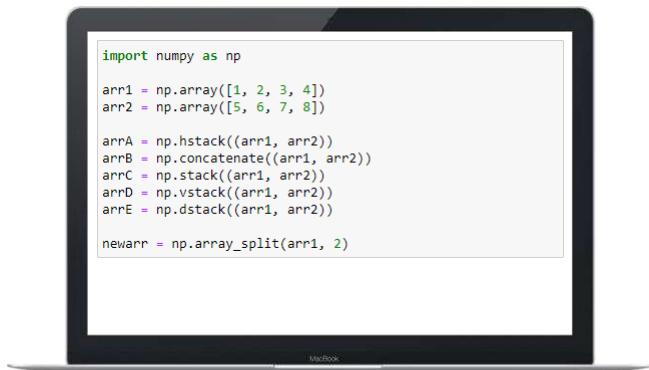


ITERACIÓN DE LAS MATRICES

- **nditer()** resuelve diferentes problemas de las iteraciones, especialmente con las matrices multidimensionales
- **ndenumerate()** es el método que utilizamos cuando necesitas el índice y el valor de cada uno de los elementos de la matriz

138

Biblioteca NumPy

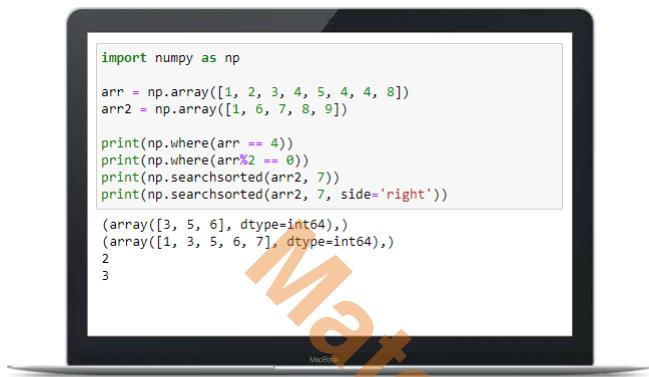


UNIÓN Y DIVISIÓN

- **concatenate()** une dos matrices pudiendo ser de distinto tamaño
- **array_split()** es el método que utilizamos para dividir matrices indicando el número de dimensiones

139

Biblioteca NumPy



BUSCAR

- **where()** lo usamos para buscar en una matriz un valor determinado y devuelve los índices que coinciden
- **searchsorted()** realiza una búsqueda en la matriz y devuelve el índice donde se insertaría el valor especificado para mantener el orden

140

Biblioteca NumPy

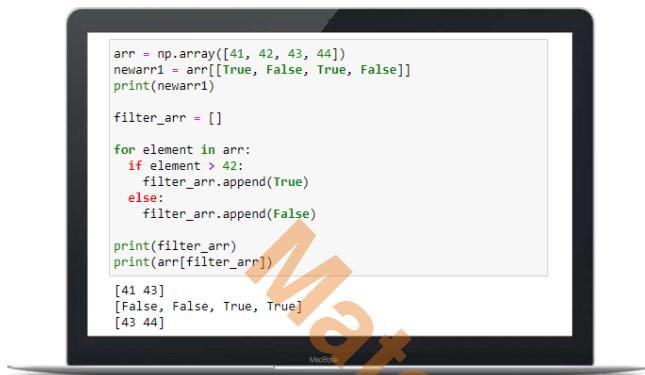


ORDENAR

- **sort()** nos permite poner los elementos de una matriz en una secuencia ordenada

141

Biblioteca NumPy



FILTRAR

- Si el valor de un índice es **True** que el elemento está contenido en la matriz filtrada, si el valor de ese índice es, **False** ese elemento se excluye de la matriz filtrada
- Podemos sustituir directamente la matriz de **True/False** por una variable iterable con nuestra condición

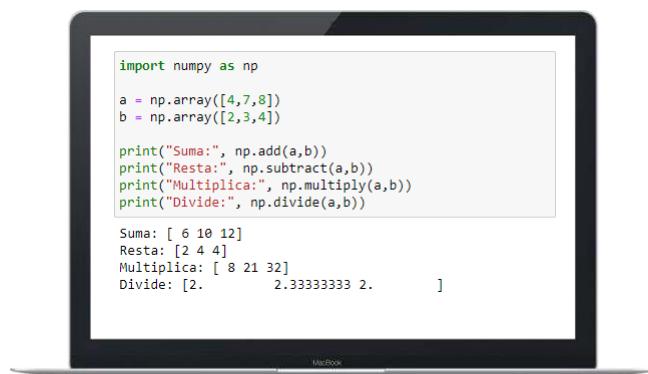
```

arr = np.array([41, 42, 43, 44])
filter_arr = arr > 42
newarr = arr[filter_arr]

```

142

Biblioteca NumPy

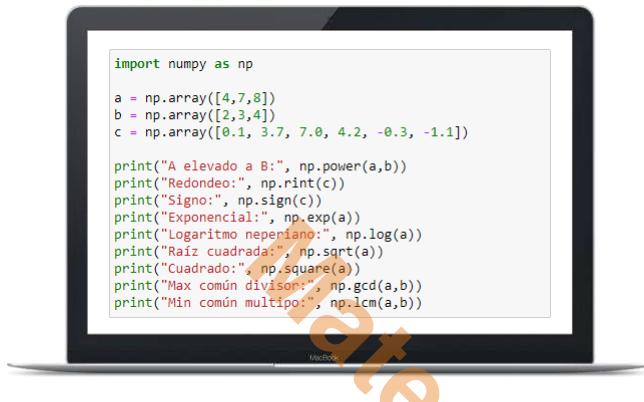


FUNCIONES MATEMÁTICAS I

- add()**, suma los elementos de dos estructuras
- subtract()**, resta los elementos de las dos estructuras
- multiply()**, multiplica los elementos de las dos estructuras
- divide()**, divide los elementos de una de las estructuras por los elementos de la otra

143

Biblioteca NumPy



FUNCIONES MATEMÁTICAS II

- **power()**, eleva cada elemento A al valor de B
- **rint()**, redondea al entero más próximo
- **sign()**, devuelve un array con valores que indican el signo de los elementos (1, 0 y -1)
- **exp()**, calcula la exponencial de los elementos
- **log()**, devuelve el logaritmo neperiano
- **sqrt()**, devuelve la raíz cuadrada
- **square()**, devuelve el cuadrado de los valores
- **gcd()**, devuelve el máximo común divisor
- **lcm()**, devuelve el mínimo común múltiplo

144

Biblioteca NumPy

Funciones de Trigonométricas

sin(), devuelve el seno de los elementos de la estructura de entrada
cos(), devuelve el coseno de los elementos de la estructura de entrada
tan(), devuelve la tangente de los elementos de la estructura de entrada
arcsin(), devuelve el arcoseno de los elementos de la estructura de entrada
arccos(), devuelve el arcocoseno de los elementos de la estructura de entrada
arctan(), devuelve la arcotangente de los elementos de la estructura de entrada
deg2rad(), convierte ángulos de grados sexagesimales a radianes
rad2deg(), convierte ángulos de radianes a grados sexagesimales

145

Biblioteca NumPy

Funciones de Comparación y Flotantes

Las funciones de comparación son:

`greater()`, `greater_equal()`, `less()`, `less_equal()`, `not_equal()`, `equal()`, `logical_or()`, `logical_xor()`, `logical_not()`, `maximum()` y `minimum()`.

Las funciones flotantes son:

`isnan()`, `floor()`, `ceil()`, `trunc()`, y `round()`.

146

Biblioteca NumPy

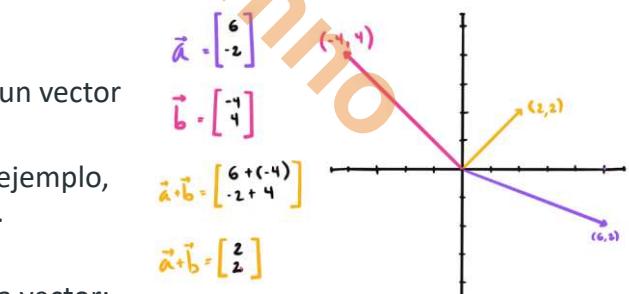
Vectores

Lo primero que debemos saber es que un vector tiene tanto magnitud como dirección.

Utilizamos vectores para describir, por ejemplo, la velocidad de objetos en movimiento.

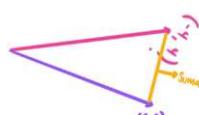
Para sumar los vectores sumamos los componentes correspondientes de cada vector:

```
import numpy as np
#Suma de Vectores
a = np.array([6, -2])
b = np.array([-4, 4])
print(a + b)
[2 2]
```



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \end{bmatrix}$$

147



Biblioteca NumPy

Multiplicar un vector por un escalar

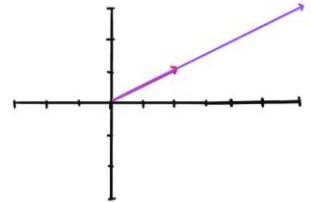
Si solo tienes magnitud, pero no dirección, entonces estás hablando de escalares.

En la multiplicación la dirección se mantiene y cambia la magnitud. Lo que estamos haciendo es escalar al vector 3 veces.

$$\vec{a} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$3\vec{a} = 3 \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$3\vec{a} = \begin{bmatrix} 6 \\ 3 \end{bmatrix}$$



```
import numpy as np
a = np.array([2, 1])
print(3 * a)
[6 3]
```

$$n \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} n \times x_1 \\ n \times x_2 \end{bmatrix}$$

148

Biblioteca NumPy

Multiplicar un vector por un escalar negativo

Pero ¿Sucede lo mismo si el número es negativo?

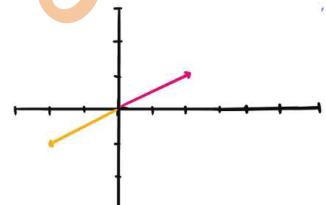
En este caso observamos que cambia la dirección pero no la magnitud.

```
import numpy as np
a = np.array([2, 1])
print(-1 * a)
[-2 -1]
```

$$n \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} n \times x_1 \\ n \times x_2 \end{bmatrix}$$

$$-1 \vec{a} = -1 \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\vec{a} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$



149

Biblioteca NumPy

Producto Escalar

Multiplicación de vectores cuyo resultado es un escalar.

Para calcular el producto escalar de dos vectores, primero debemos multiplicar los elementos correspondientes y luego sumar los términos del producto.

```
import numpy as np
a = np.array([3, 2])
b = np.array([2, 2])
print(a.dot(b))
```

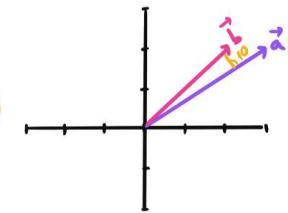
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = (x_1 \times y_1) + (x_2 \times y_2)$$

$$\vec{a} \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad \vec{b} \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$\vec{a} \cdot \vec{b} = (3 \times 2) + (2 \times 2)$$

$$\vec{a} \cdot \vec{b} = 6 + 4$$

$$\vec{a} \cdot \vec{b} = 10$$



150

Biblioteca NumPy

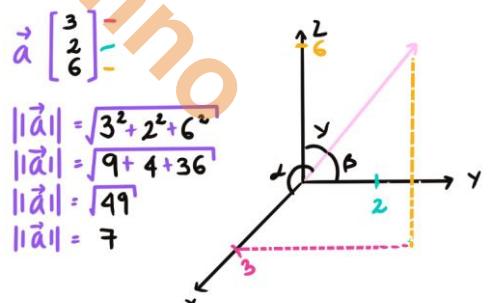
Norma de un vector

La norma es otro término para la magnitud de un vector y se denota con dos líneas dobles (||) en cada lado.

Se define como una raíz cuadrada de la suma de cuadrados para cada componente de un vector

- Elevar al cuadrado cada componente
- Suma todos los cuadrados
- Toma la raíz cuadrada

$$\|\vec{x}\| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$



```
import numpy as np
a = np.array([3, 2])
print(np.linalg.norm(a))
```

7.0

151

Biblioteca NumPy

Ángulos entre vectores

El cálculo del ángulo entre vectores tiene muchas aplicaciones prácticas.

Es ampliamente utilizado en PNL (procesamiento del lenguaje natural) cuando se buscan cadenas que son muy similares.

```
import numpy as np

def angle_between(v1, v2):
    dot_pr = v1.dot(v2)
    norms = np.linalg.norm(v1) * np.linalg.norm(v2)

    return np.rad2deg(np.arccos(dot_pr / norms))

a = np.array([1, 4, 5])
b = np.array([2, 1, 5])

print(angle_between(a, b))
29.152519407030084
```

$$\cos(\theta) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

$$\vec{a} = \begin{bmatrix} 1 \\ 4 \\ 5 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} 2 \\ 1 \\ 5 \end{bmatrix}$$

$$\cos(\theta) = \frac{[1 \ 4 \ 5] \cdot [2 \ 1 \ 5]}{(\sqrt{1^2 + 4^2 + 5^2})(\sqrt{2^2 + 1^2 + 5^2})}$$

$$\cos(\theta) = \frac{(1 \times 2) + (4 \times 1) + (5 \times 5)}{(\sqrt{1+16+25})(\sqrt{4+1+25})}$$

$$\cos(\theta) = \frac{31}{(\sqrt{42})(\sqrt{30})}$$

$$\cos(\theta) = \frac{31}{\sqrt{35}}$$

$$\theta = \arccos\left(\frac{31}{\sqrt{35}}\right)$$

$$\theta = 29.15^\circ$$

152



153

Servicios con Requests



Requests es una librería para HTTP, escrita en Python, para seres humanos.

El módulo urllib2 que se encuentra en el estándar de Python, ofrece la mayoría de las funcionalidades necesarias para HTTP, pero sin coherencia.

Fue construida para otra época, - y una web diferente-. Requiere una gran cantidad de trabajo (incluso reimplementar métodos) para ejecutar las tareas más sencillas.

Las cosas no deberían ser así. No en Python.

154

Servicios con Requests



Para instalar Requests, simplemente ejecute este simple comando en su terminal de elección:

```
python -m pip install requests
```

Obtenga el código fuente en GitHub, donde el código siempre está disponible . Puede clonar el repositorio público:

```
git clone git://github.com/psf/requests.git
```

155

Requests: HTTP para Humanos

Versión v1.1.0. (installation)

Requests es una librería para HTTP, refilencenciada bajo Apache2 <Apache2>, escrita en Python, para seres humanos.

El módulo urllib2 que se encuentra en el estándar de Python, ofrece la mayoría de las funcionalidades necesarias para HTTP, pero su api está completamente rota. Fue construida para otra época, - y una web diferente-. Requiere una gran cantidad de trabajo (incluso reimplementar métodos) para ejecutar las tareas más sencillas.

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"id": 1, "name": "User", ...'
>>> r.json()
[{"private_gists": 419, "total_private_repos": 77, ...]
```

[Ver el mismo código, sin Requests.](#)

Requests quita las complicaciones de trabajar HTTP/1.1 en Python - haciendo que la integración con servicios web sea transparente. No hay necesidad de agregar queries a tus URLs manualmente, o convertir tu información a formularios para hacer una petición POST. La reutilización de keep-alive y conexión HTTP se hace automáticamente, todo gracias a

Requests: HTTP para Humanos

Versión v1.1.0. (installation)

Requests es una librería para HTTP, refilencenciada bajo Apache2 <Apache2>, escrita en Python, para seres humanos.

El módulo urllib2 que se encuentra en el estándar de Python, ofrece la mayoría de las funcionalidades necesarias para HTTP, pero su api está completamente rota. Fue construida para otra época, - y una web diferente-. Requiere una gran cantidad de trabajo (incluso reimplementar métodos) para ejecutar las tareas más sencillas.

Las cosas no deberían ser así. No en Python.

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"id": 1, "name": "User", ...'
>>> r.json()
[{"private_gists": 419, "total_private_repos": 77, ...}]
```

[Ver el mismo código, sin Requests.](#)

Requests quita las complicaciones de trabajar HTTP/1.1 en Python - haciendo que la integración con servicios web sea transparente. No hay necesidad de agregar queries a tus URLs manualmente, o convertir tu información a formularios para hacer una petición POST. La reutilización de keep-alive y conexión HTTP se hace automáticamente, todo gracias a

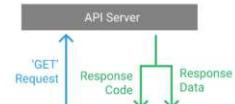
Servicios con Requests

```
import requests
r = requests.get('http://api.open-notify.org/iss-now.json')
print(type(r))
print(r.headers)
print(r.headers['content-type'])
print(f'{r.status_code} / {r.encoding}')
print(r.text)
print(r.json()['message'])

<class 'requests.models.Response'>
application/json
200 / None
{"message": "success", "timestamp": 1582760314, "iss_position": {"latitude": "-50.7640", "longitude": "-76.3681"}}
success
```

REQUESTS.GET

- **requests** es el objeto que representa la conexión http
- Las funciones de *requests* genera un objeto respuesta desde donde podemos acceder a los datos de respuesta
- **text** contiene el texto de la respuesta
- **json()** retorna el objeto deserializado de una respuesta JSON



156

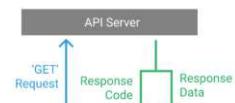
Servicios con Requests

```
import requests
u = {'Nombre': 'Borja', 'Apellidos': 'Cabeza'}
r = requests.post("https://postman-echo.com/post", data=u)
print(r.text)

{"args":{}, "data": "", "files": {}, "form": {"Nombre": "Borja", "Apellidos": "Cabeza"}, "headers": {"x-forwarded-proto": "https", "host": "postman-echo.com", "content-length": "29", "accept": "*/*", "accept-encoding": "gzip, deflate", "content-type": "application/x-www-form-urlencoded", "user-agent": "python-requests/2.22.0", "x-forwarded-port": "443"}, "json": {"Nombre": "Borja", "Apellidos": "Cabeza"}, "url": "https://postman-echo.com/post"}
```

REQUESTS.POST

- **data** es el parámetro utilizado para enviar los datos en la petición
- Los datos se representan mediante diccionarios



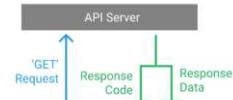
157

Servicios con Requests



HEADERS PERSONALIZADOS

- **headers** es el parámetro utilizado para enviar la información de la cabecera en la petición
- La información de las cabeceras se representan mediante diccionarios



158

Servicios con Zeep



Zeep inspecciona el documento WSDL y genera el código correspondiente para usar los servicios y tipos en el documento.

Esto proporciona una interfaz programática fácil de usar para un servidor SOAP 1.1 y SOAP 1.2, también ofrece soporte para enlaces HTTP Get y Post.

El análisis de los documentos XML se realiza mediante la biblioteca lxml . Esta es la biblioteca XML de Python más eficaz y compatible actualmente disponible. Esto da como resultado importantes beneficios de velocidad al procesar grandes respuestas SOAP.

```

from zeep import Client
client = Client('http://www.webservices.net/ConvertSpeed.asmx?wsdl')
result = client.service.ConvertSpeed(
    100, 'kilometersPerHour', 'milesPerHour')
assert result == 62.137
  
```

159

Servicios con Zeep

Para instalar archivos de rueda, necesita un cliente pip reciente, ejecute:

```
pip install zeep
```

Tenga en cuenta que la última versión compatible con Python 2.7, 3.3, 3.4 y 3.5 es Zeep 3.4, instale a través de:

```
pip install zeep == 3.4.0
```



160

Servicios con Zeep

OPERACIONES DISPONIBLES

- Una de las primeras cosas que hará si comienza a desarrollar una interfaz para un servicio web wsdl es **obtener una descripción general de todas las operaciones disponibles y sus firmas** de llamadas.
- Zeep ofrece una interfaz de línea de comandos para facilitar esto.

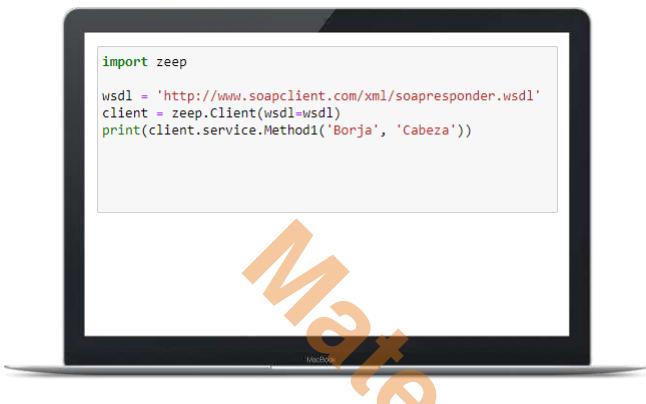
```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="SoapResponder" targetNamespace="http://www_soapclient_com_xml_soapresponder_wsdl" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:types>
    <xs:schema targetNamespace="http://www_soapclient_com_xml_soapresponder_wsdl">
      <xs:import namespace="http://www_soapclient_com_xml_soapresponder_wsdl"/>
      <xs:element name="MethodId" type="xs:string"/>
      <xs:element name="MethodParam1" type="xs:string"/>
      <xs:element name="MethodParam2" type="xs:string"/>
      <xs:element name="MethodReturn" type="xs:string"/>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="MethodIdResponse">
    <wsdl:part name="MethodId" type="xs:string"/>
  </wsdl:message>
  <wsdl:message name="MethodResponse">
    <wsdl:part name="MethodParam1" type="xs:string"/>
    <wsdl:part name="MethodParam2" type="xs:string"/>
    <wsdl:part name="MethodReturn" type="xs:string"/>
  </wsdl:message>
  <wsdl:portType name="SoapResponderPortType">
    <wsdl:operation name="Method">
      <wsdl:input message="MethodIdResponse"/>
      <wsdl:output message="MethodResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SoapBinding" type="soap:Binding">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Method">
      <soap:operation name="Method" style="document"/>
      <wsdl:input>
        <soap:body use="encoded" namespace="http://www_soapclient_com_xml_soapresponder_wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        <soap:header name="MethodId" message="MethodIdResponse" part="MethodId" use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="encoded" namespace="http://www_soapclient_com_xml_soapresponder_wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        <soap:header name="MethodParam1" message="MethodResponse" part="MethodParam1" use="literal"/>
        <soap:header name="MethodParam2" message="MethodResponse" part="MethodParam2" use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="SoapResponderService">
    <wsdl:port name="SoapResponderPort" binding="SoapBinding">
      <wsdl:address>http://www_soapclient_com_xml_soapresponder_wsdl</wsdl:address>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
  
```

```
python -mzeep http://www.soapclient.com/xml/soapresponder.wsdl
```

161

Servicios con Zeep

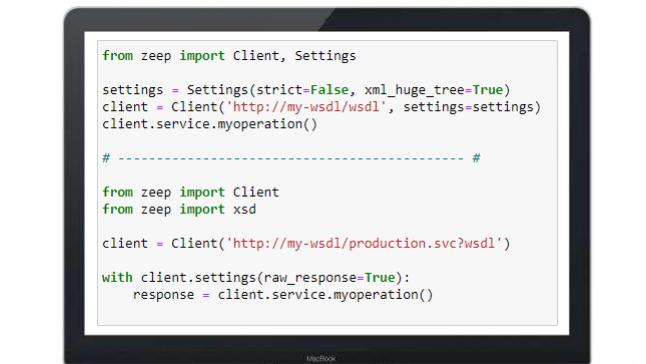


OBJETO CLIENT

- **Client** la interfaz principal para interactuar con un servidor SOAP.
- El enlace predeterminado se puede especificar al iniciar el cliente pasando el **service_name** y **port_name**.
- Almacenamiento en caché de archivos WSDL y XSD
- Cuando se inicializa el cliente, recuperará automáticamente el archivo WSDL que no almacena en caché, pero se recomienda habilitarlo por rendimiento.

162

Servicios con Zeep

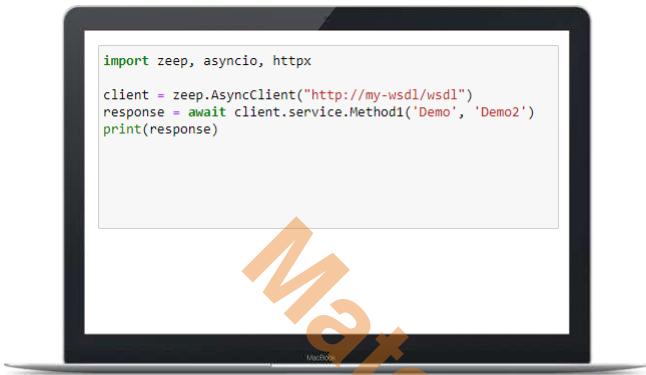


CONFIGURANDO CLIENT

- La clase **Client** acepta un argumento de configuración para configurar el cliente. Puede inicializar el objeto usando el siguiente código:
- El objeto de configuración siempre es accesible a través del cliente usando **client.settings**.

163

Servicios con Requests



ASYNCCLIENT

- **AsyncClient** le permite ejecutar operaciones de forma asíncrona.
- Sin embargo, los documentos wsdl se cargan utilizando métodos síncronos.
- El código originalmente no fue escrita para uso asíncrono.
- Para usar operaciones asíncronas, debe usar **AsyncClient()** y el **AsyncTransport()**.

164



165

Networking

Niveles de Acceso a Red

Python proporciona dos niveles de acceso a los servicios de red.

En un nivel bajo, puede acceder mediante socket del sistema operativo subyacente, lo que le permite implementar clientes y servidores para protocolos orientados a conexión y sin conexión.

Python también tiene bibliotecas que brindan acceso de nivel superior a protocolos de red de nivel de aplicación específicos, como FTP, SMTP, HTTP, etc.

166

Networking

Sockets

Los sockets son los puntos finales de un canal de comunicaciones bidireccional.

Los sockets pueden comunicarse dentro de un proceso, entre procesos en la misma máquina o entre procesos en diferentes continentes.

Los sockets se pueden implementar en varios tipos de canales diferentes: sockets de dominio Unix, TCP, UDP, etc. La biblioteca de sockets proporciona clases específicas para manejar los transportes comunes, así como una interfaz genérica para manejar el resto.

167

Networking

Sockets II

Dominio o protocolos que se utiliza como mecanismo de transporte. Estos valores son constantes como AF_INET, PF_INET, PF_UNIX, PF_X25, etc.

Tipo, puede ser SOCK_STREAM para protocolos orientados a conexión y SOCK_DGRAM para protocolos sin conexión.

Protocolo, normalmente cero, para identificar una variante de un protocolo dentro de un dominio y tipo.

Nombre del Host o identificador de una interfaz de red. Puede ser una cadena, una dirección IPv4 o IPv6, o direcciones INADDR_BROADCAST o INADDR_ANY.

Puerto, número de puerto por donde el servidor escucha a los clientes o el nombre de un servicio.

168

Networking

Métodos Generales de los Sockets

`socket.recv()`, recibe un mensaje TCP

`socket.send()`, transmite un mensaje TCP

`socket.recvfrom()`, recibe un mensaje UDP

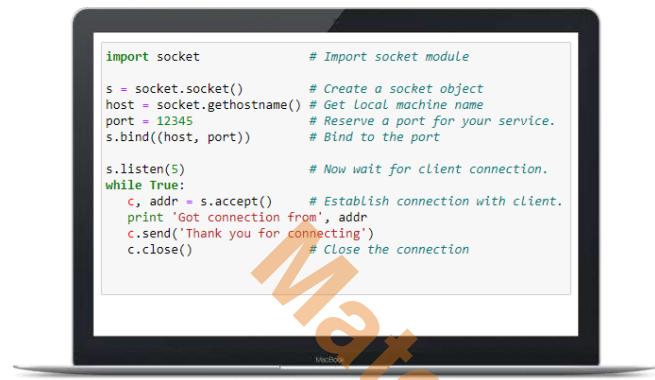
`socket.sendto()`, transmite un mensaje UDP

`socket.close()`, cierra el zócalo

`socket.gethostname()`, devuelve el nombre de host

169

Networking

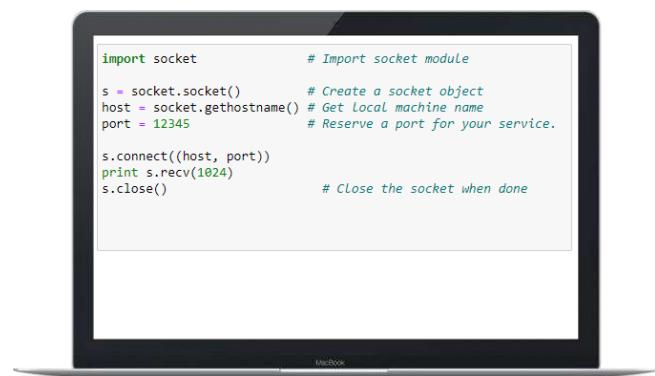


SOCKETS SERVER

- **socket()** es el método que crear el objeto Socket
- **gethostname()** es el método que devuelve en el nombre de host donde se ejecuta el programa
- **bind()** es el método que vincula la dirección y el puerto al socket
- **listen()** es el método que configura e inicia la escucha TCP
- **accept()** es el método que acepta la conexión de un cliente TCP
- **send()** es el método que transmite un mensaje TCP

170

Networking



SOCKETS CLIENT

- **socket()** es el método que crear el objeto Socket
- **gethostname()** es el método que devuelve en el nombre de host donde se ejecuta el programa
- **connect()** es el método que inicia activamente la conexión del servidor TCP
- **recv()** es el método que recibe un mensaje TCP

171

Networking

Módulos de Internet

Lista de los módulos más importantes en la programación de Python para Internet.

Protocol	Common function	Port No	Python module
HTTP	Web pages	80	httpplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib

172

Networking

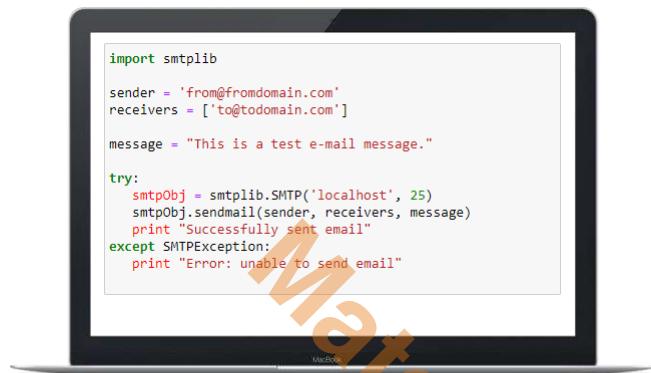
```
from ftplib import FTP
ftp = FTP('ftp.debian.org')
ftp.login()
ftp.cwd('debian')
ftp.retrlines('LIST')
ftp.retrbinary("RETR README", open('README', 'wb').write)
ftp.quit()
```

FTP CLIENT

- **login()** es el método que nos autentica en el servidor FTP, permite usuario y contraseña
- **cwd()** es el método que nos permite cambiar de carpertos
- **retrline()** es el método que recupera una lista de archivos o directorios
- **retrbinary()** es el método que recupera un archivo en el modo de transferencia binaria
- **close()** o **quit()** cierra la conexión con el servidor FTP

173

Networking

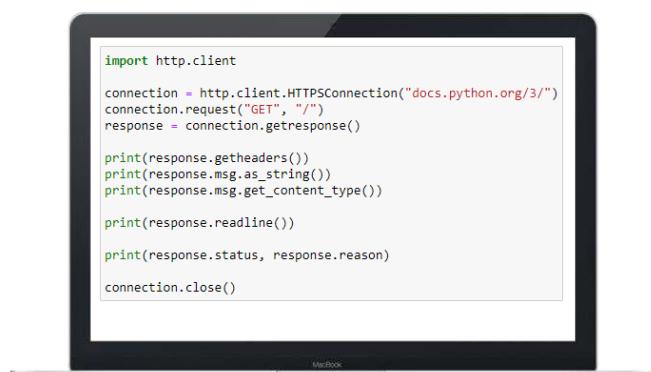


SMTP CLIENT

- **SMTP()** es el método que crear el objeto cliente de correo electrónico
- **sendmail()** envía una copia del mensaje a todos los destinatarios de la lista *receivers*

174

Networking



HTTP CLIENT

- **HTTPSConnection()** establece una conexión con un sitio Web
- **request()** permite lanzar mensajes de petición al sitio Web
- **getresponse()** permite capturar el mensaje de respuesta que nos envía el sitio Web
- Diferentes método nos devuelven información tanto del *Header* como del *Body* del mensaje.

175



176

Debug y Testing

Depuración de Código

En ocasiones el resultado obtenido al programar no es el esperado, por algún error cometido, lo cual hace que el programa funcione pero no lo haga correctamente. En estas ocasiones **es muy importante depurar** el programa.

En la depuración lo que hacemos es ejecutar el programa paso a paso, así podemos ver las instrucciones que se están ejecutando, además de poder ver los valores que van tomando las variables.

Esto nos puede servir de ayuda para ver las variables, los valores que van tomando, las instrucciones que se están ejecutando y así poder ver dónde hemos cometido un error.

177

Debug y Testing

Depuración de Código II

El módulo **pdb** define un depurador de código fuente interactivo para programas Python. El uso típico para ejecutar un programa bajo el control del depurador es:

```
>>> import pdb  
>>> import mymodule  
>>> pdb.run('mymodule.test()')
```

pdb.py también puede ser invocado como un script para depurar otros scripts:

```
python3 -m pdb myscript.py
```

Es habitual que cada IDE incluya las herramientas para depurar código.

178

Debug y Testing

Pruebas Unitarias

Las pruebas unitarias es un método para determinar si un módulo o un conjunto de módulos de código funciona correctamente, y son imprescindibles para el sólido desarrollo de software.

Al escribir pruebas unitarias se automatiza el control de calidad, al mismo tiempo que se crea una documentación viva de las funcionalidades y APIs implementadas.

Existen diversos frameworks para implementar pruebas unitarias. La librería estándar incluye: **unittest** y **doctest**.

179

Debug y Testing

Pruebas Unitarias con unittest

Para la implementación de la prueba unitaria utilizando el módulo **unittest**, crearemos una clase como unidad de prueba que comprobará el comportamiento de nuestro módulo.

Dentro de ésta creamos tantos métodos (que retorna True o False) como funciones del módulo que queramos probar. Todos los métodos que comiencen con el nombre **test_** serán ejecutados.

unittest admite métodos **assert** proporcionados por la clase base **TestCase** que utilizamos para probar algo.

180

Debug y Testing

Pruebas Unitarias con unittest II

Las pruebas pueden ser muchas y su preparación puede ser repetitiva. El método **setUp()** se utiliza con este fin y es llamado por la prueba cuando se ejecute.

Si el método **setUp()** lanza una excepción mientras se ejecuta la prueba, se considerará que la prueba ha sufrido un error y no se ejecutará el método de prueba propiamente dicho.

Análogamente, se puede proporcionar un método **tearDown()** que haga limpieza después de que se ejecute el método de prueba.

181

Debug y Testing

Pruebas Unitarias con unittest III

```
class Calculadora:
    Numero1 = 0
    Numero2 = 0

    def __init__(self) -> None:
        pass

    def Suma(self) -> int:
        return (self.Numero1 + self.Numero2)

    def Resta(self) -> int:
        return (self.Numero1 - self.Numero2)

    def Multiplica(self) -> int:
        return (self.Numero1 * self.Numero2)

    def Divide(self) -> int:
        return (self.Numero1 / self.Numero2)
```

182

Debug y Testing

Pruebas Unitarias con unittest IV

```
import unittest
from calculadora import *

class CalculadoraTesting(unittest.TestCase):
    def setUp(self):
        self.calculadora = Calculadora()
        self.calculadora.Numero1 = 5
        self.calculadora.Numero2 = 10

    def test_suma(self):
        self.assertEqual(self.calculadora.Suma(), 15)

    def test_resta(self):
        self.assertEqual(self.calculadora.Resta(), -5)

if __name__ == '__main__':
    unittest.main()
```

183

Debug y Testing

Pruebas Unitarias con unittest V

El bloque final muestra un modo sencillo de ejecutar las pruebas. `unittest.main()` proporciona una interfaz de línea de órdenes para el script de prueba.

Si se le pasa una opción `-v` al script de prueba, se establecerá un nivel mayor de detalle del proceso de `unittest.main()`.

También se puede usar el módulo `unittest` desde la línea de comandos para ejecutar pruebas de módulos, clases o hasta métodos de prueba individuales:

```
python -m unittest test_module1 test_module2
python -m unittest test_module.TestClass
python -m unittest test_module.TestClass.test_method
```

184

Debug y Testing

Pruebas Unitarias con unittest VI

La clase `TestCase` proporciona varios métodos de afirmación para comprobar y reportar fallos.

En la siguiente tabla se enumeran los métodos más utilizados.

Puede consultar todos los métodos en:

<https://docs.python.org/3.9/es/library/unittest.html>

Método	Comprueba que	Nuevo en
<code>assertEqual(a, b)</code>	<code>a == b</code>	
<code>assertNotEqual(a, b)</code>	<code>a != b</code>	
<code>assertTrue(x)</code>	<code>bool(x) is True</code>	
<code>assertFalse(x)</code>	<code>bool(x) is False</code>	
<code>assertIs(a, b)</code>	<code>a is b</code>	3.1
<code>assert IsNot(a, b)</code>	<code>a is not b</code>	3.1
<code>assertIsNone(x)</code>	<code>x is None</code>	3.1
<code>assertIsNotNone(x)</code>	<code>x is not None</code>	3.1
<code>assertIn(a, b)</code>	<code>a in b</code>	3.1
<code>assertNotIn(a, b)</code>	<code>a not in b</code>	3.1
<code>assertIsInstance(a, b)</code>	<code>isinstance(a, b)</code>	3.2
<code>assertNotIsInstance(a, b)</code>	<code>not isinstance(a, b)</code>	3.2

185



186



187



188



189



190

Material del Alumno