

# Microsoft® Technology Associate Training

# Programación en Python



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO

**EOI** Escuela de  
organización  
Industrial

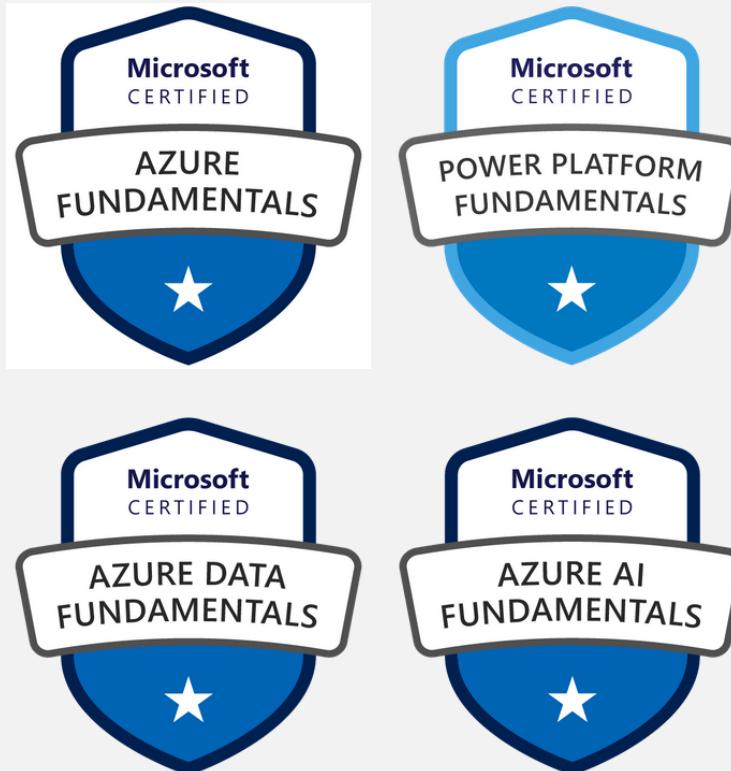
**Generation**  
SPAIN

Unión Europea  
Fondo Social Europeo  
Iniciativa de Empleo Juvenil  
El FSE invierte en tu futuro



# python





# Billy M. Vanegas B.

Ingeniero Informático

[billy@billyclasstime.com](mailto:billy@billyclasstime.com)

<https://billyclasstime.com>



# Hola!

## Presentación de los alumnos

Vamos a familiaricémonos

Su nombre

Formación

Conocimientos sobre programación

Espectativas de este Modulo 2

HELLO

# Agenda

1 - Introducción

6 – Trabajando con Ficheros

2 – Entorno de trabajo

7 – Acceso a base de datos

3 – Variables y sentencias de control

8 – Patrones de diseño y arquitectónico

4 – Colecciones y JSON

9 – Aplicaciones Web

5 – Funciones clases y módulos

10 – Consumiendo ApiRESTful

# 1 - Introducción

[Artículo](#)[Discusión](#)[Leer](#)[Editar](#)[Ver historial](#)

# Python

Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, se utiliza para desarrollar aplicaciones de todo tipo, ejemplos: Instagram, Netflix, Panda 3D, entre otros.<sup>2</sup> Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Administrado por la Python Software Foundation, posee una licencia de código abierto, denominada Python Software Foundation License.<sup>3</sup> Python se clasifica constantemente como uno de los lenguajes de programación más populares.

Python	
	Desarrollador(es)
Python Software Foundation	<a href="#">Sitio web oficial</a>
Información general	
Extensões comunes	.py , .pyc , .pyd , .pyo , .pyw , .pyz , .pyi
Paradigma	Multiparadigma: orientado a objetos, imperativo, funcional, reflexivo
Apareció en	1991
Diseñado por	Guido van Rossum
Última versión estable	3.10.2 <sup>1</sup> (14 de enero de 2022 (3 meses y 8 días))
Sistema de tipos	Fuertemente tipado, dinámico
Implementaciones	Cython, IronPython, Jython, Python for S60, PyPy, ActivePython, Unladen Swallow
Dialectos	Stackless Python, RPython
Influido por	ABC, ALGOL 68, C, Haskell, Icon, Lisp, Modula-3, Perl, Smalltalk, Java
Ha influido a	Boo, Cobra, D, Falcon, Genie, Groovy, Ruby, JavaScript, Cython, Go Latino
Sistema operativo	Multiplataforma
Licencia	Python Software Foundation License
<a href="#">[editar datos en Wikidata]</a>	

# Filosofía de Python

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.

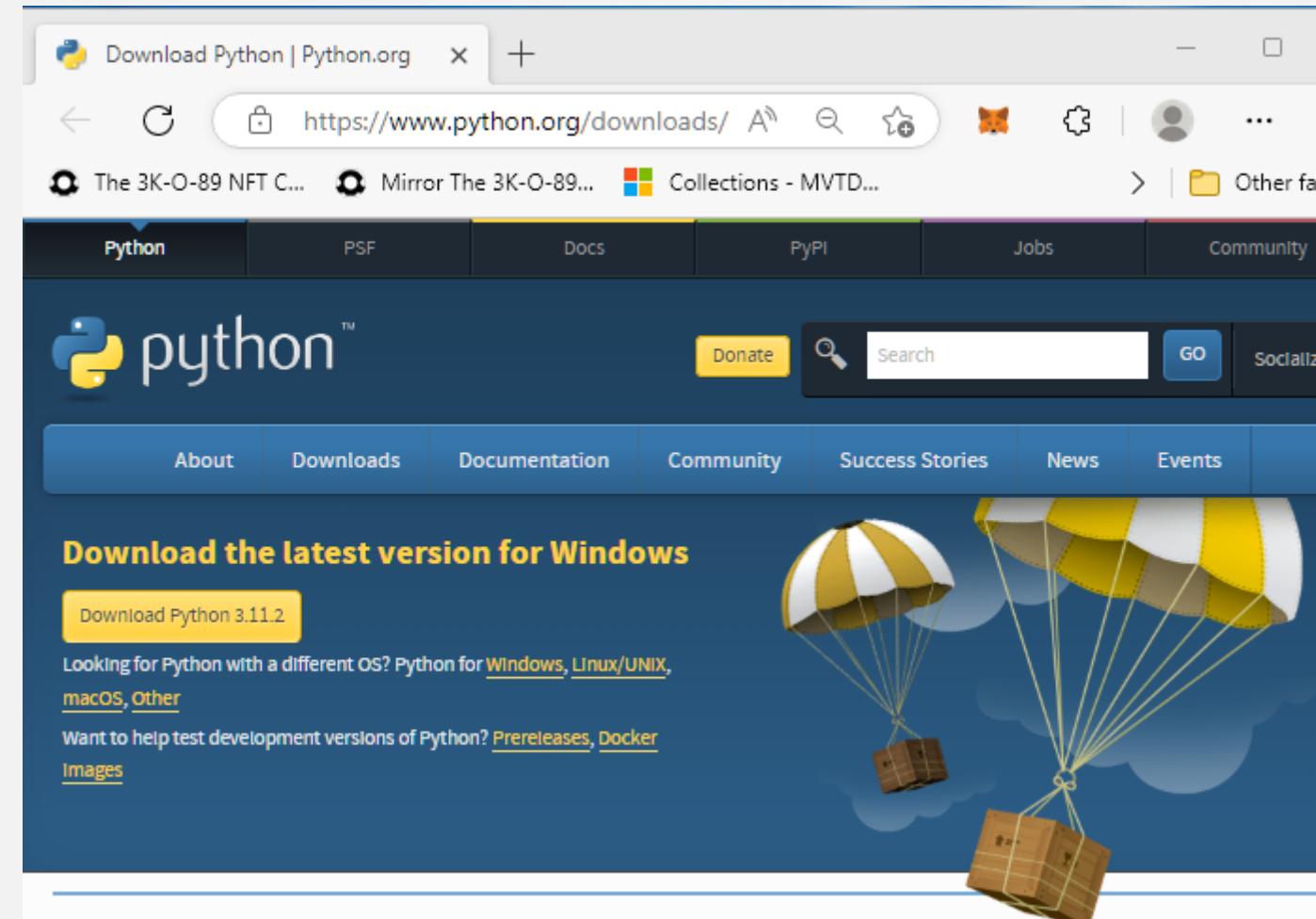
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una —y preferiblemente solo una— manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que seas holandés
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

## 2 – 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.

# Actividad Instalar Python y otras herramientas de trabajo

- Introducción al control de versiones con [Git](#)
- Practica [Introduction to Git - Training | Microsoft Learn](#)
- Mas información [Introduction to version control with Git - Training | Microsoft Learn](#)

# 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/>

The image shows two side-by-side screenshots. On the left is the official Visual Studio Code website at <https://code.visualstudio.com>. It features a dark header with the Microsoft logo and navigation links for Docs, Updates, Blog, API, Extensions, FAQ, and Learn. A banner at the top right announces "Version 1.66 is now available! Read about the new features and fixes from March." Below this is a large white section with the text "Code editing. Redefined." and "Built on open source. Runs everywhere." followed by download buttons for "Download for Windows Stable Build" and "Other platforms and Insiders Edition". At the bottom, a note states "By using VS Code, you agree to its license and privacy statement." On the right is a screenshot of the Visual Studio Code application window. The title bar says "Visual Studio Code - Code Editin...". The interface shows a sidebar with file icons and a list of extensions: Python, GitLens — Git supercharged, C/C++ (highlighted), ESLint, Debugger for Chrome, and Language Support for Java. The main editor area displays a portion of a JavaScript file with code related to service workers and a service worker registration function. The status bar at the bottom of the window shows "serviceWorker.js - create react app - Visual Studio Code - In...".

*Visual Studio Code*  
instalar **Python extension for Visual Studio Code** desarrollada por Microsoft.

# Practica – Como comenzar con Python y VS Code

- [Get started with Python in Visual Studio Code - Training | Microsoft Learn](#)

# Extensiones en Visual Studio Code



instalar **Python extension for Visual Studio Code** desarrollada por Microsoft.

[Python - Visual Studio Marketplace](https://marketplace.visualstudio.com/items?itemName=ms-python.python)

<https://marketplace.visualstudio.com/items?itemName=ms-python.python>



instalar **Jupyter** desarrollada por Microsoft. (Opcional)

[Jupyter - Visual Studio Marketplace](https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter)

<https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter>

# Entorno virtual

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

- Windows PowerShell

```
py -m venv env  
env\Scripts\activate.ps1
```

- Unix (Linux o Mac OS)

```
python3 -m venv env  
source env/bin/activate
```

# Entorno de trabajo

**Entornos globales**, cada instalación de Python (por ejemplo, Python 2.7, Python 3.6, Python 3.7, Python 3.9 Anaconda 4.4.0, 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.

# 3 – Variables y Sentencias de Control

# Sintaxis y Estructuras de Control

## COMENTARIOS

```
#Comentarios  
text = """Escribamos valores alfanumericos  
en distintas lineas. """
```

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

# Sintaxis y Estructuras de Control

## DECLARACION DE VARIABLES

```
numero = 10
Numero = 20
saludo = "Hola mundo"
print (numero)
print (Numero)
print (Numero + numero)
print ("saludos" + saludo)
print (type(numero))
print (type(saludo))
```

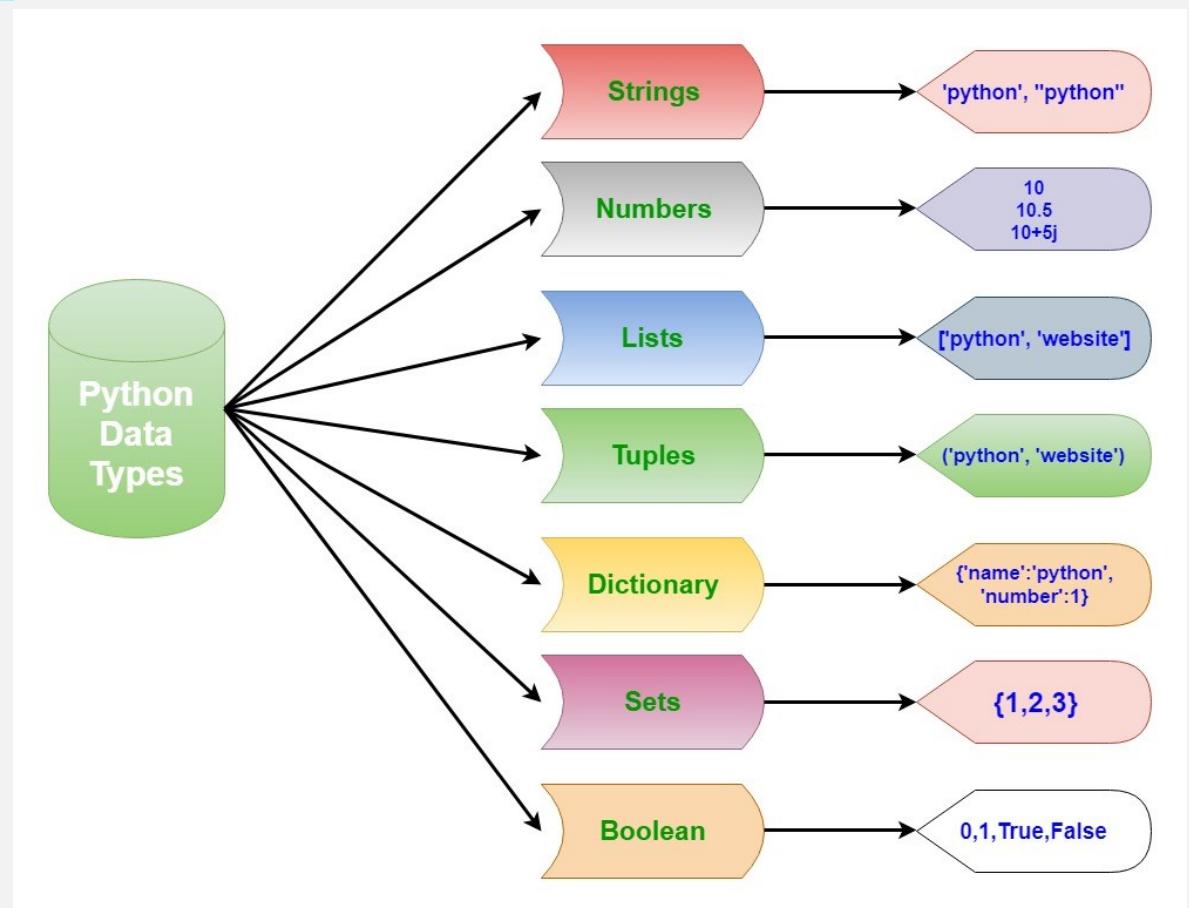
```
10
20
30
saludos:Hola mundo
<class 'int'>
<class 'str'>
```

- 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

# Sintaxis y Estructuras de Control

## Tipos de Datos

Algunos de los datos más communes en Python



# Sintaxis y Estructuras de Control

## Tipos de Datos (Cont.)

Resumen tipos de datos en Python

Tipo	Clase	Notas	Ejemplo
<code>str</code>	Cadena	Inmutable	<code>'Cadena'</code>
<code>unicode</code>	Cadena	Versión Unicode de <code>str</code>	<code>u'Cadena'</code>
<code>list</code>	Secuencia	Mutable, puede contener objetos de diversos tipos	<code>[4.0, 'Cadena', True]</code>
<code>tuple</code>	Secuencia	Inmutable, puede contener objetos de diversos tipos	<code>(4.0, 'Cadena', True)</code>
<code>set</code>	Conjunto	Mutable, sin orden, no contiene duplicados	<code>{4.0, 'Cadena', True}</code>
<code>frozenset</code>	Conjunto	Inmutable, sin orden, no contiene duplicados	<code>frozenset([4.0, 'Cadena', True])</code>
<code>dict</code>	Mapping	Grupo de pares clave:valor	<code>{'key1': 1.0, 'key2': False}</code>
<code>int</code>	Número entero	Precisión fija, convertido en <code>long</code> en caso de overflow.	<code>42</code>
<code>long</code>	Número entero	Precisión arbitraria	<code>42L</code> o <code>456966786151987643L</code>
<code>float</code>	Número decimal	Coma flotante de doble precisión	<code>3.1415927</code>
<code>complex</code>	Número complejo	Parte real y parte imaginaria <code>j</code> .	<code>(4.5 + 3j)</code>
<code>bool</code>	Booleano	Valor booleano verdadero o falso	<code>True</code> o <code>False</code>

- Mutable: si su contenido (o dicho valor) puede cambiarse en tiempo de ejecución.
- Inmutable: si su contenido (o dicho valor) no puede cambiarse en tiempo de ejecución.

# Sintaxis y Estructuras de Control

## Asignación simultanea

```
#Asignación simultanea  
a=5  
b=10  
print("Paso1. valores iniciales")  
print(a)  
print(b)
```

Paso1. Valores iniciales

5

10

Descubre la potencia de la asignación simultanea entre variables

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

# Sintaxis y Estructuras de Control

## CONVERSIONES

```
a=5  
b="25"  
c="25.7"  
print("Número:" + str(a))  
print(int(b))  
print(float(c))
```

Número: 5

25

25.7

- **str(s)** 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

# Sintaxis y Estructuras de Control

## CADENA DE CARACTERES

```
cadena="Hola mundo"  
print(cadena)  
print(cadena[2])  
print(cadena[2:])  
print(cadena[:2])  
print(cadena[2:6])  
print(cadena[-2])  
print(len(cadena))
```

Hola mundo

|

la mundo

Ho

la m

d

10

- Una secuencia inmutable de caracteres Unicode
- Es un tipo secuencial e inmutable, un objeto de este tipo no se puede modificar después de haber sido creado.
- 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 (Diapositiva No. 38)

# Sintaxis y Estructuras de Control

## Formateando Cadenas

```
mensaje = "Mundo"
print ("Hola "+mensaje + " !!!")
print ("Hola {}".format(mensaje))
print ("Hola {} !!!".format(s=mensaje))
print (f"Hello {mensaje} !!!")
numero = 10/3
print (numero)
print ("Hello{:.2f}!!!".format(n=numero))
```

Hola Mundo !!!

Hola Mundo !!!

Hola Mundo !!!

Hola Mundo !!!

3.33333333335

Hola 3.33 !!!

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

# Sintaxis y Estructuras de Control

## Fechas y Horas

```
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: 2022-04-25

Fecha: 2022-04-25 16:26:03.567149

Año: 2022

Mes: 4

Dia: 25

Hora:16:26

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.

# Sintaxis y Estructuras de Control

## Parse Fechas

```
from datetime import datetime
fecha = "10-11-2022"
obj = datetime.strptime(fecha, '%m-%d-%Y').date()
print(obj)
print(f"{obj.day}-{obj.month}-{obj.year}")
```

2022-10-11

11-10-2022

Recibe el nombre de **Parse**, a la acción de analizar gramaticalmente un valor.

**strftime()** convierte los valores de texto en fecha.

# Python strftime( )

Convert string to datetime

'10/11/18 14:35:32'



'%m/%d/%y %H:%M:%S'

# Sintaxis y Estructuras de Control

## Formato de Fechas

```
from datetime import datetime  
fecha = datetime.now()  
print(fecha.strftime("%A %d %b %Y"))
```

Monday 25 Apr 2022

El formato de fechas es ampliamente utilizado para datos de fechas en diferentes culturas y países.

`strftime()` dar formato a un valor de fechas

Python  
`strftime()` Function  
String Format to timestamp



# Sintaxis y Estructuras de Control

## Comodines para formato y conversión de fechas

%a	Nombre corto para un día de la semana	vi.
%A	Nombre largo para un día de la semana	viernes
%w	Día de la semana en número	5 es viernes
%d	Día en el mes - desde el 1 hasta el 31 (Según el mes)	29
%b	Nombre del mes versión corta	abr.
%B	Nombre del mes versión larga	abril
%m	Numero del mes 01-12	4
%y	Año versión corta sin siglo	22
%Y	Año versión larga	2022
%H	Hora formato 00-23	10
%I	Hora formato 00-12	10
%p	AM/PM	AM
%M	Minutos 00-59	43
%S	Segundos 00-50	23
%f	Microsegundos	230203
%z	UTC offset	200
%Z	Zona horaria	UTC+02:00
%j	Número del día en el año	119
%U	Número de la semana del año (Iniciando semana en domingo)	17
%W	Número de la semana del año (Iniciando semana en lunes)	17
%c	versión local de fecha y hora	29/04/2022 0:00:00
%x	versión local de la fecha	29/04/2022
%X	Local versión of time	0:00:00
%%	Un carácter tanto porciento	%

Los comodines se utilizan para transformar texto en fechas y formatear su representación.

# Sintaxis y Estructuras de Control

## Fechas en español

```
from datetime import datetime
import locale
locale.setlocale(locale.LC_TIME, 'es_ES.UTF-8')

dt =datetime.strptime('29/04/2022','%d/%m/%Y') #datetime.now()

print(dt)
print('\nDirectivas\n-----')
print('Día de semana corto      ' + dt.strftime('%a'))
print('Día de la semana largo    ' + dt.strftime('%A'))
print('Número de dia de la semana' + dt.strftime('%w'))
print('Día del mes                ' + dt.strftime('%d'))
print('Nombre del mes corto       ' + dt.strftime('%b'))
```

### Directivas

---

Día de semana corto	:	vi.
Día de la semana largo	:	viernes

Para localizar (cambiar de cultura) una ejecución del interprete de Python importamos **locale**

# Sintaxis y Estructuras de Control

## Tiempo

```
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("Milisegundos:",int(time.time()*1000.0))
print(time.asctime(time.localtime(time.time())))
```

```
Time: 1650906051.0794837
time.struct_time(tm_year=2022, tm_mon=4, tm_mday=25,
tm_hour=19, tm_min=0, tm_sec=51, tm_wday=0,
tm_yday=115, tm_isdst=1)
Año: 2022
Minutos: 0
Milisegundos: 1650906051081
Mon Apr 25 19:00:51 2022
```

Para trabajar con variables que almacenan tiempo importamos 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.

# Sintaxis y Estructuras de Control

## 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')))
```

```
2022-04-26 02:16:38.391250+09:00
2022-04-25 19:16:38.394230+02:00
```

Para trabajar con diferentes zonas horarias hacemos uso del módulo `pytz`.

Este módulo debe ser instalado porque no viene por defecto.

`pip` utilidad para disponer y gestionar módulos externos de Python.

`pip install pytz` instala el módulo `pytz`

```
Python 3.10 (64-bit)

Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 2
Type "help", "copyright", "credits" or "license" fo
>>> help('modules')
```

# Sintaxis y Estructuras de Control

## Operadores

Literales booleanos	
Verdadero	True
Falso	False

Operadores lógicos	
and	Y Lógico
or	O Lógico
not	negación de verdad

Operadores a nivel de bits	
&	and
	Or
~	not
^	xor
>>	desplazamiento a la derecha
<<	desplazamiento a la izquierda

Los operadores son símbolos que le indican al intérprete que realice una operación específica, como aritmética, comparación, lógica, etc.

Operadores numéricos	
+	Suma
-	Diferencia o negación
*	Producto
/	División con decimales
%	Resto
//	División entera
**	Potencia

Operadores de asignación	
=	Igual
+=	Incremento
-=	decremento

Operadores de comparación	
==	igual
!=	distinto
>	mayor que
>=	mayor o igual que
<	menor que
<=	menor o igual que
is	mismo objeto
is not	distinto objeto

Operadores de pertenencia	
in	pertenece a la colección
not in	no pertenece a la colección

# Sintaxis y Estructuras de Control

## Sentencia de decisión If/Else

```
a = 10
b = 20
if (a>b):
    print (f"el número mayor es {a}")
else:
    if (b>a):
        print(f"El número mayor es {b}")
    else:
        print ("los dos números son iguales")
```

El número mayor es 20

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** simpplica un else if (SiNo entonces si)

Python no tiene la estructura **según caso**.

# Sintaxis y Estructuras de Control

## Sentencia de repetición FOR

```
lenguajes = ['python', 'c', 'c++', 'java']
for lenguaje in lenguajes:
    print(lenguaje)
for numero in range(3):
    print(numero)
for numero in range(len(lenguajes)):
    print(f"P:{numero} - V:{lenguajes[numero]}")
```

```
python
c
c++
java
0
1
2
P:0 - V:python
P:1 - V:c
P:2 - V:c++
P:3 - V:java
```

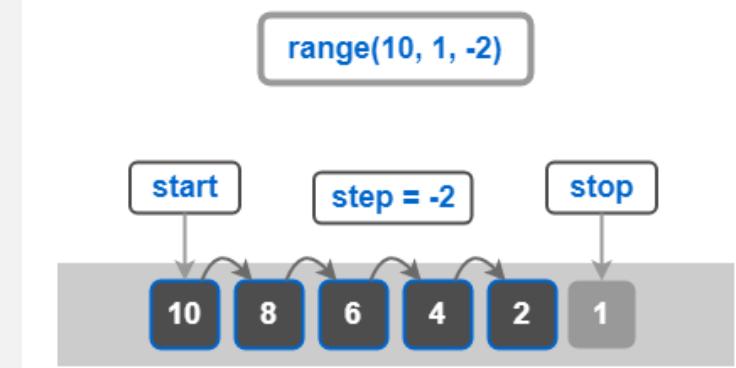
Las sentencia de repetición *for* ejecutan bloques de código de forma iterativa.

La sentencia de repetición *for* se usa para recorrer y trabajar con las colecciones.

*range(n)*, permite definir el inicio, el final y el paso.

*continue* con la siguiente operación de incremento

Break finaliza el *for*



# Sintaxis y Estructuras de Control

## Sentencia de repetición While

```
print("Instrucción antes del while")
valor = 0
while(valor < 5):
    valor += 1
    if (valor == 3):
        continue
    print(f"Valor actual:{valor}")
print("Instrucción después del while")
```

Instrucción antes del while

Valor actual:1

Valor actual:2

Valor actual:4

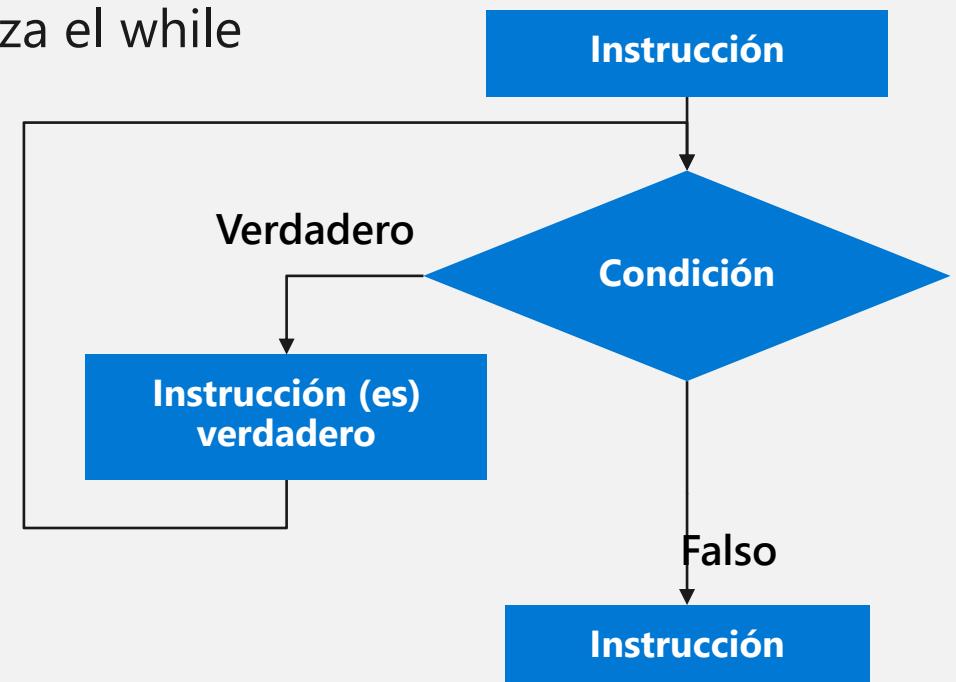
Valor actual:5

Instrucción después del while

La sentencia de repetición `while`, realiza iteraciones mientras se cumpla la una condición.

`continue`, continua con la siguiente iteración.

`Break` finaliza el `while`



Python no tiene Repetir Hasta

# Sintaxis y Estructuras de Control

## Control de excepciones

```
numero1 = 100
numero2 = 0

try:
    print(numero1/numero2)
except ZeroDivisionError:
    print("Error al dividir por cero.")
except:
    print("Error")
else:
    print("La división se calculó correctamente.")
finally:
    print("Fin del programa")
```

Error al dividir por cero.  
Fin del programa

**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 una excepción

**finally** bloque de instrucciones que se ejecutan siempre que finaliza el **try**, **except** o **else**

# Ejercicios Implementar el ejercicio de algoritmos de fundamentales de Programación

1. Calcular la letra del DNI Español
2. Calcular el salario de un empleado
3. Determinar la ruta para llegar a una ciudad por avión. [Con base de datos]
4. Calcula el área y perímetro de un círculo dado su radio.
5. Dada una lista de números enteros, determina cuál es el mayor y cuál es el menor.
6. Crea un programa en Python que convierta grados Celsius a Fahrenheit.
7. Dado un número entero, crea un programa en Python que determine si es par o impar.
8. Crea un programa en Python que determine si un año es bisiesto o no.
9. Crea un programa en Python que determine si una cadena de texto es un palíndromo o no.
10. Dada una lista de nombres, crea un programa en Python que ordene la lista alfabéticamente.

# Ejercicios Implementar el ejercicio de algoritmos de fundamentales de Programación (Cont.)

11. Crea un programa en Python que calcule el factorial de un número entero.
12. Dado un número entero, crea un programa en Python que determine si es primo o no.
13. Crea un programa en Python que calcule el área y volumen de un cubo dado su lado.
14. Dada una lista de números enteros, crea un programa en Python que calcule la suma de todos los números pares de la lista.
15. Crea un programa en Python que determine si un número es positivo, negativo o cero.
16. Dada una lista de números enteros, crea un programa en Python que calcule la media de la lista.
17. Crea un programa en Python que genere un número aleatorio entre 1 y 100, y le pida al usuario adivinarlo. El programa en Python deberá indicar si el número introducido es mayor o menor que el número aleatorio, hasta que el usuario adivine el número correcto.
18. Crea un programa en Python que determine si una cadena de texto es un anagrama de otra cadena de texto.
19. Dada una lista de números enteros, crea un programa en Python que elimine los números duplicados de la lista.
20. Crea un programa en Python que determine si un número es capicúa o no.

# 4 – Colecciones y JSON

# Colecciones y JSON

## Colecciones, str

```
ubicacion = 'Madrid capital de España'
print(ubicacion.upper())
print(ubicacion.lower())
aes = 'aaaaaaa'
nueva = aes.replace('a', 'b', 3)
print(nueva)
print(ubicacion.split(' '))
ubicacion = '\t\t Madrid capital de España'
print(ubicacion)
print(ubicacion.strip())
print(ubicacion.count('a'))
print(ubicacion.strip().endswith('paña'))
print(ubicacion.strip().upper().startswith('MADRID'))
datos="Pais:{}, capital:{}"
print(datos.format("España", 'Madrid'))
```

**upper()** convierte una cadena en mayúsculas  
**lower()** convierte una cadena en minúsculas  
**find()** Busca en la cadena un valor específico y devuelve la posición donde se encontró  
**replace()** Devuelve una cadena donde un valor especificado se reemplaza con un valor especificado  
**split()** Divide la cadena en el separador especificado y devuelve una lista  
**strip()** Devuelve una versión recortada de la cadena.  
**count()** Devuelve el número de veces que aparece un valor especificado en una cadena  
**endswith()** Devuelve verdadero si la cadena termina con el valor especificado  
**startswith()** Devuelve verdadero si la cadena comienza con el valor especificado  
**format()** Formatea valores especificados en una cadena

# Colecciones y JSON

## Colecciones, Listas

```
colores=["Red", "Yellow", "Green"]
colores.append("Black")
colores.insert(2,"Orange")
print("Lista de colores:",colores)
print("Posicion color Yellow:",colores.index("Yellow"))
colores.remove("Yellow")
print("Color en la posición 2:",colores[2])
print("Lista de colores:",colores)
print(colores*2)
```

```
List de colores: ['Red', 'Yellow', 'Orange',
'Green', 'Black']
Posicion color Yellow: 1
Color en la posición 2: Green
List de colores: ['Red', 'Orange', 'Green',
'Black']
['Red', 'Orange', 'Green', 'Black', 'Red',
'Orange', 'Green', 'Black']
```

- 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** ordena los elementos con base al valor

**reverse** invierte el orden de los elementos en base a su índice

**pop(i)** elimina el elemento de la posición i.

**remove(value)** elimina el primer elemento coincidente con el valor

Se puede duplicar la lista tantos elementos multiplicándola por un número

# Colecciones y JSON

## Colecciones, Tuplas

```
numeros = (17, 89, 21, 988, 42, 429, 32, 834)
print(numeros[4])
print(list(enumerate(numeros)))
print(max(numeros))
print(min(numeros))
print(sum(numeros))
print(numeros*2)
```

```
42
[(0, 17), (1, 89), (2, 21), (3, 988),
(4, 42), (5, 429), (6, 32), (7, 834)]
988
17
2452
(17, 89, 21, 988, 42, 429, 32, 834, 17,
89, 21, 988, 42, 429, 32, 834)
```

- Una tupla es una colección de elementos ordenados con un índice 0.
- En una tupla no se pueden añadir, eliminar y modificar elementos.
- Se puede duplicar la tupla tantos veces multiplicándola por un número

# Colecciones y JSON

## Colecciones, conjuntos

```
ciudades = {"madrid", "albacete", "sevilla"}  
ciudades.add("valencia")  
print("Conjunto de ciudades:", ciudades)  
ciudades.discard("albacete")  
print("Conjunto de ciudades:", ciudades)  
for ciudad in ciudades:  
    print(ciudad)  
print(len(ciudades))
```

```
Conjunto de ciudades: {'valencia',  
'sevilla', 'albacete', 'madrid'}  
Conjunto de ciudades: {'valencia',  
'sevilla', 'madrid'}
```

valencia

sevilla

madrid

3

- Un conjunto es una colección de elementos sin índice, se dice que esta desordenado
- En un conjunto se pueden añadir y eliminar elementos. (add y discard)
- Para acceder a los valores tenemos que recorrer la colección mediante un for.
- No se pueden duplicar los conjuntos al ser únicos en el conjunto. (No se admite el operador \*)

# Colecciones y JSON

## List Comprehension

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
    if "a" in x:
        newlist.append(x)
print('Nueva lista a partir de recorrer la primera:',newlist)

newlist = [x for x in fruits if "a" in x]
print('Nueva lista con LC',newlist)
```

- La comprensión de listas o **List Comprehension**, ofrece una sintaxis más corta cuando desea crear una nueva lista basada en los valores de una lista existente.

- Sintaxis:  
newlist = [expresión **for** el elemento en iterable **if condición == True**]

- La **condición Condición** es como un filtro que solo acepta los elementos que se valoran como Verdadero (**True**).

```
Nueva lista a partir de recorrer la primera: ['apple', 'banana', 'mango']
Nueva lista con LC ['apple', 'banana', 'mango']
```

# Colecciones y JSON

## Colecciones, diccionarios

```
dicc= {"red":"rojo", "blue":"blue", "green":"verde"}  
dicc["black"]="negro"  
print(dicc)  
dicc.pop("blue")  
print(dicc)  
print(dicc["red"])  
for key in dicc:  
    print(key,'→',dicc[key])
```

```
{'red': 'rojo', 'blue': 'blue', 'green': 'verde',  
'black': 'negro'}  
{'red': 'rojo', 'green': 'verde', 'black': 'negro'}  
rojo  
red → rojo  
green → verde  
black → negro
```

- Un diccionario es un conjunto 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 un valor de una clave o uno alternativo si la clave no existe.

No se puede duplicar un diccionario con \*

# Colecciones y JSON

## JavaScript Object Notation - JSON

```
import json
citricos = ["limon", "naranja", "pomelo", "lima"]
citricosJSON=json.dumps(citricos)
print("Json de citricos:",citricosJSON)

listacitricos = json.loads(citricosJSON)
print(listacitricos[2])
```

```
Json de citricos: ["limon", "naranja", "pomelo", "lima"]
pomelo
```



- JSON es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo.
- Está basado en un subconjunto del Lenguaje de Programación JavaScript, JSON es un formato de texto que es completamente independiente del lenguaje desde 2019.
- `dumps` serializamos un objeto a JSON
- `loads` deseabilizamos un JSON a objeto

# 5 – Funciones, clases y módulos

# Funciones, Clases y Objetos

## Funciones

```
def saludar(nombre):
    print(f'holá {nombre}!!!')
def sumar(a,b):
    return (a+b)
def addColores(colores,color):
    colores.append(color)
    return True

saludar('Billy')
print(sumar(4,3))
colores=["azul","rojo","verde"]
addColores(colores,'negro')
print(colores)
```

```
holá Billy!!!
7
['azul', 'rojo', 'verde', 'negro']
```

- Una Función es un bloque de código que se ejecuta cuando es llamado. Es la implementación de los subalgoritmos vistos en fundamentos
- **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
- Argumento es el valor enviado a la función
- Parámetro es el valor entre paréntesis de la función
- Las colecciones y otros objetos son parámetros por referencia

# Funciones, Clases y Objetos

## Funciones

```
def ProcesoInicial():
    print('Paso 1 desde aqui')

def SaludarATodos(*nombres):
    for i in nombres:
        print(f'Hello:{i}')

def SaludarATodosDict(**nombres):
    for i in nombres:
        print(f'Saludar a todos Dict:{i}, {nombres[i]}')

ProcesoInicial()

SaludarATodos('Franco', 'Mario')
SaludarATodos('Mariano')
SaludarATodos('Paris', 'Dakota', 'Helena', 'Rocio')
SaludarATodosDict(nombre="Billy", apellidos="Vanegas")
```

- Puede establecer \*param para definir N parámetros en la función o \*\*param para tratarlos como diccionarios.
- Puede definir valores por defecto en los parámetros param=<valor por defecto>

```
def ciudades(ciudad='Oslo'):
    print(ciudad)

ciudades('Bogotá')
ciudades()
```

```
Bogotá
Oslo
```

# Funciones, clases y objetos

## Funciones Lambda

```
def saludar(nombre):
    print(f'holá {nombre} !!!')

saludarLambda = lambda nombre:print(f'holá {nombre} !!!')

saludar('Billy')
saludarLambda('Carola')
```

```
holá Billy !!!  
holá Carola !!!
```

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

# Funciones, Clases y Objetos

## Funciones Asíncronas

```
import asyncio

async def saludo():
    print('Hola ... ')
    await asyncio.sleep(4)
    print('... world')

#python 3.7+
asyncio.run(saludo())
print('siguiente instrucción del programa')
```

```
Hola ...
...
siguiente instrucción del programa
```

- **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.

# Funciones, Clases y Objetos

## Modulos

```
#Modulo modulo.py
def saludar(nombre):
    print(f'Hello {nombre}!!')

#Programa principal App.py
import modulo
modulo.saludar("Billy")

#Programa principal App.py
from modulo import saludar
saludar('Billy -01')

#Programa Principal App.py
from modulo import saludar as saludo
saludo('Billy -02')
```

Hola Billy!!

Hola Billy -01!!

Hola Billy -02!!

- 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

# Funciones, Clases y Objetos

## Clases

```
class Persona:  
    Nombre = ""  
    Apellidos = ""  
  
    def __init__(self, nombre, apellidos):  
        self.Nombre = nombre  
        self.Apellidos = apellidos  
  
    def Saludar(self):  
        print(f'Hola {self.Nombre} !!!')  
  
profesor = Persona("Billy", "Vanegas")  
print(profesor.Nombre)  
profesor.Saludar()
```

Billy  
Hola Billy !!!

- 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

# Funciones, Clases y Objetos

## Atributos de Instancia y Atributos de Clase

```
class student:  
    school = "San Patric"      #Atributos de clase  
    totalStudents = 0          #Atributos de clase  
    def __init__(self, name, std, roll_no):  
        self.nm = name          #Atributos de instancia  
        self.std = std           #Atributos de instancia  
        student.totalStudents +=1  
    def getData(self):          #metodo getter  
        print("Student name: ", self.nm)  
        print("Standard: ", self.std)  
    def setData(self, name, std):#metodo setter  
        self.nm = name  
        self.std = std  
  
print("The school name is:", student.school)  
stud_1 = student("Om", "4th", 9)  
stud_1.getData()  
stud_2 = student("Hari", "5th", 14)  
stud_2.getData()  
print("The totals students are:", student.totalStudents)
```

- Python a diferencia de otros lenguajes utiliza el concepto de Atributo de clase y de instancia para diferenciar el uso de variables estáticas.
- Atributos de clase son variables dentro de la clase que comparten valores entre instancias (variables estáticas)
- Atributos de instancia son valores únicos para cada objeto creado a partir de la clase
- **getters** en Python son métodos utilizados para asignar valores a los atributos de instancia
- **setters** en Python son métodos utilizados para recuperar valores de los atributos de instancia

# Funciones, Clases y Objetos

## Convención para métodos privados

```
class Vehiculo():
    #Clase Vehiculo
    def __privado(self):
        #Metodo supuestamente privado"""
        print("Soy privado")
    def un_metodo(self):
        #Metodo supuestamente publico desde el que se
        # accede a __privado
        self.__privado()
g1 = Vehiculo()
#g1.__privado() Una excepción no se encuentra metodo
g1._Vehiculo__privado()
g1.un_metodo()
```

- Para cumplir con la encapsulación Python utiliza como convención el uso de las barras bajas en el nombramiento de las funciones de las clases.
- Convencionalmente un método (función) que sea nombrada con dos barras bajas serán marcadas como privadas.
- Esta misma convención se utiliza para los atributos de instancia y marcarlos como privados

# Funciones, Clases y Objetos

## Función `property()` para atributos de instancia

```
class partner:
    def __init__(self):
        self._age = 0
    def get_age(self):      # get value of _age
        print("getter method called")
        return self._age
    def set_age(self, a):   # set value of _age
        print("setter method called")
        if (a<18):
            | raise ValueError("age not valid")
        self._age = a
    def del_age(self):     # delete _age attribute
        del self._age
    age = property(get_age, set_age, del_age)
mark = partner()
mark.age = 18
mark._age=15
print(mark.age)
```

- En Python, `property()` es una función integrada que crea y devuelve un objeto de propiedad.
- Un objeto de propiedad tiene tres métodos, `getter()`, `setter()` y `delete()`.
- La sintaxis del uso de `property()` es `property(fget,fset,fdel,doc)` donde:
  - fget recupera el valor
  - gset establece el valor
  - fdel elimina el valor
  - doc es opcional y se usa para documentar
- Por convención, para establecer el enlace entre el atributo de instancia y los métodos de propiedad, este se nombra con `_`, ejemplo `_age`

# Funciones, Clases y Objetos

## Herencia de Clases

```
class Persona:  
    Nombre = None  
    Apellidos = None  
    def __init__(self, nombre, apellidos):  
        self.Nombre = nombre  
        self.Apellidos = apellidos  
  
class Alumno(Persona):  
    Curso = None  
    def __init__(self, nombre, apellidos, curso):  
        self.Curso = curso  
        super().__init__(nombre, apellidos)  
  
est = Alumno('Yolanda', 'Merizalde', 'Cloud computing')  
print(f'{est.Nombre} {est.Apellidos} - {est.Curso}')
```

Yolanda Merizalde - Cloud computing

- 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.

# Funciones clases y Objetos

## Polimorfismos con Classes

```
class Figura:
    def dibujar(self,nombre):
        print(f'Dibujando un:{nombre}')
class Triangulo(Figura):
    def __init__(self,nombre):
        self.nombre = nombre
class Rectangulo(Figura):
    def __init__(self,nombre):
        self.nombre = nombre
class Cuadrado(Figura):
    def __init__(self,nombre):
        self.nombre = nombre
def DibujarFigura(figura):
    figura.dibujar(figura.nombre)
triangulo = Triangulo('Triangulo verde')
cuadrado = Cuadrado('Cuadrado rojo')
rectangulo = Rectangulo('Rectangulo amarillo')
DibujarFigura(triangulo)
DibujarFigura(cuadrado)
DibujarFigura(rectangulo)
```

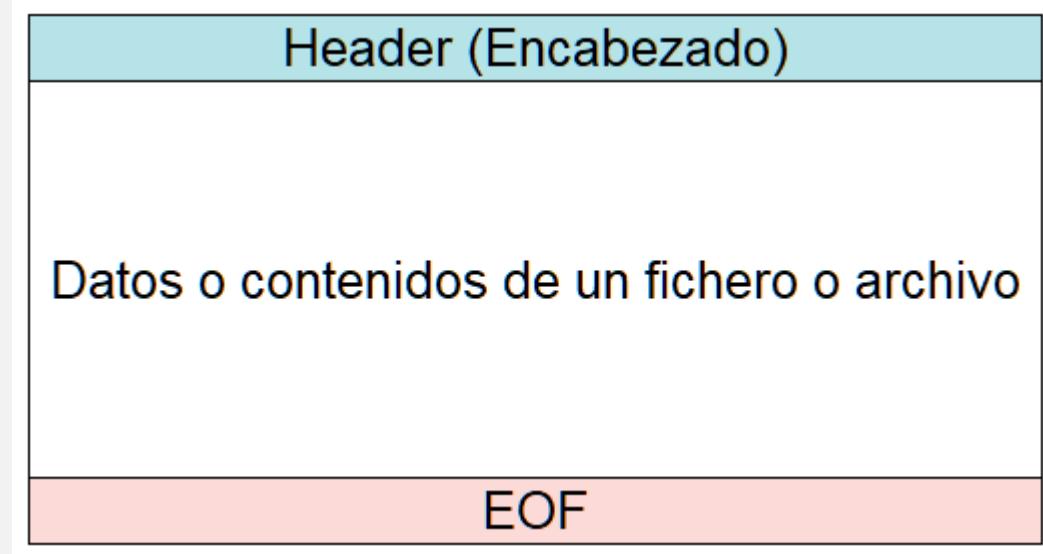
- Capacidad de un objeto para adoptar múltiples formas.
- Se utiliza como funcionalidad compartida de objetos de una misma clase

En el ejemplo se ha utilizado la función **DibujarFigura** con un parámetro que tiene el método común a todos.

# 6 – Trabajando con ficheros

# Ficheros

## Definición de fichero o archivo



- En esencia, un **archivo** o **fichero** es un conjunto contiguo de bytes que se utiliza para almacenar datos. Estos datos están organizados en un formato específico y pueden ser algo tan simple como un archivo o fichero de texto o tan complicado como un programa ejecutable.

Se constituyen de tres partes principales Encabezado, Datos y EOF

- **Encabezado:** metadatos sobre el contenido del fichero (nombre de fichero, tamaño, tipo, etc.)
- **Datos:** contenido del archivo escrito por el creador o editor
- **End of file (EOF):** carácter especial que indica el final del archivo

# Ficheros

## Rutas de ficheros

```
D:\00-CComputing\Python>Tree  
Folder PATH listing for volume Data2020  
Volume serial number is 8E2D-AB3E  
D:  
├── Documentacion  
├── Ejercicios  
│   ├── Basicos  
│   ├── Ficheros  
│   ├── Funciones  
│   └── ClasesModulos  
└── Medios
```

Cuando accede a un archivo en un sistema operativo, se requiere una ruta de archivo. La ruta del archivo es una cadena que representa la ubicación de un archivo. Se divide en tres partes principales:

- **Ruta de la carpeta:** la ubicación de la carpeta de archivos en el sistema de archivos donde las carpetas subsiguientes están separadas por una barra diagonal /(Unix) o una barra diagonal inversa \(Windows)
- **Nombre de archivo:** el nombre real del archivo
- **Extensión:** el final de la ruta del archivo precedido por un punto (.) que se usa para indicar el tipo de archivo

# Ficheros

## Open / Close

```
try:  
    fichero=open("file_500.csv", "rt")  
    #,encoding='UTF-8')  
    print(type(fichero))  
    contenido = fichero.read()  
    print(contenido)  
except FileNotFoundError as e:  
    print(f"{e}")  
except Exception as e:  
    print(f"E:{e}")  
finally:  
    fichero.close()  
    print('closing')
```

```
<class '_io.TextIOWrapper'>  
comma_separated_values,aÑ±o,corazÃ³n,Ã¡rbol  
closing
```

- `open` es la función preconstruida (built in) de Python para trabajar con ficheros
- Requiere dos parámetros, la ruta, 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
- `close()` es la función preconstruida de Python para finalizar el uso de un fichero
- `encoding=` controla la codificación de caracteres, para leer ficheros con contenidos como acentos, eñes, etc puede usar ‘UTF-8’

# Ficheros

## modos de operacion / Tipos ficheros

```
def fichero_existe(nombre_fichero):
    try:
        open(nombre_fichero, "rt")
    except FileNotFoundError:
        return False
    return True

try:
    file='file_300.txt'
    if (not fichero_existe(file)):
        fichero=open(file, "wt", encoding='UTF-8')
        contenido="árbol", 'limón', 'naranja'
        fichero.write(contenido)
        print('Escrito contenido en fichero')
    else:
        fichero=open(file, "rt", encoding='UTF-8')
        contenido = fichero.read()
        print(f'Leido conteido:{contenido}')
except Exception as e:
    print(f"E:{e}")
finally:
    fichero.close()
    print('closing')
```

- "r" - Lectura - Valor predeterminado. Abre un archivo para lectura, error si el archivo no existe
- "a" - Agregar - Abre un archivo para agregar, crea el archivo si no existe
- "w" - Escribir - Abre un archivo para escribir, crea el archivo si no existe
- "x" - Crear - Crea el archivo especificado, devuelve un error si el archivo existe.
- "t" - Texto - Valor por defecto. Modo de texto
- "b" - Binario - Modo binario (por ejemplo, imágenes)

# Ficheros

## Ejercicios

- Hacer un programa en Python que genere los datos de los 100 hombres y mujeres del ejercicio 1 de colecciones y almacenarlo en un fichero en disco.
- Hacer un programa que genere los datos de las edades de las 100 personas del ejercicio 2.
- Hacer un programa que genere los datos de las 100 personas del ejercicio 3 colecciones.
- Hacer un programa que genere los datos de las temperaturas mensuales de las 20 ciudades del ejercicio4 de colecciones.
- Crear sendos programas que lean los ficheros de los resultados anteriores y procesen la información solicitada por cada uno de los ejercicios de colecciones.
- El resultado de las soluciones de cada programa deberá ser guardado en sendos ficheros con el nombre de resultado de cada uno de los ejercicios-

# 7 – Acceso a base de datos

# Acceso a base de datos

## Entornos Virtuales en Python

### Create virtual enviroment

```
py -m venv env
```

### Activate virtual enviroment

```
env\Scripts\activate
```

### Bash

```
pip install python-dateutil
```

### Instaling the package of the enviroment

```
pip install -r requirements.txt
```

### Deactivating virutal enviroment

```
env\Scripts\deactivate.bat
```

- Un entorno virtual es una herramienta que ayuda a mantener separadas las dependencias requeridas por diferentes proyectos.

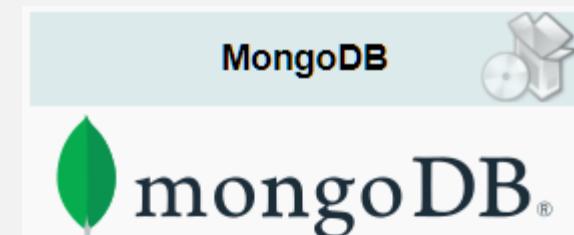
- Mediante la creación de entornos virtuales de Python se pueden aislar versiones del interprete, librerías módulos y dependencias.

- Esta es una de las herramientas más importantes que utilizan la mayoría de los desarrolladores de Python..

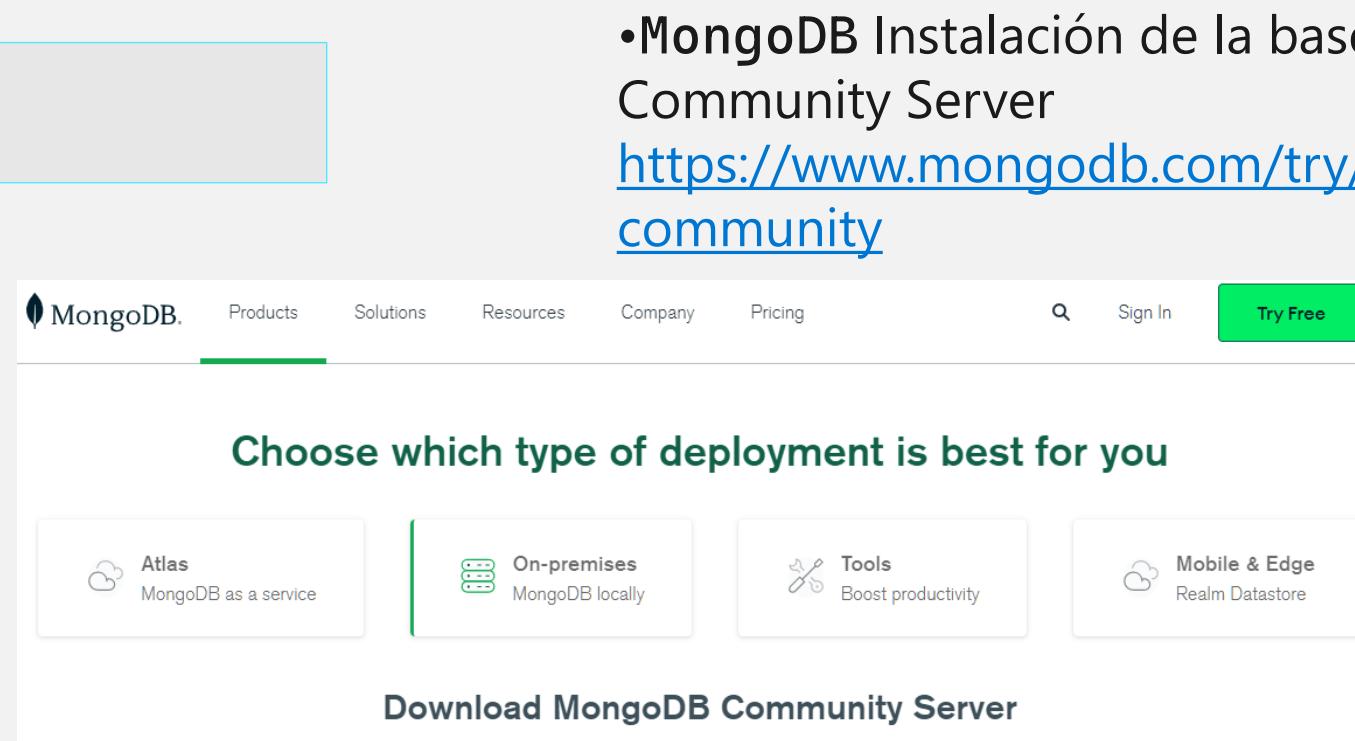
- Crear el entorno virtual
- Activar el entorno virtual
- Instalar paquetes en el entorno virtual
- Definir la migración de paquetes
- Desactivar el entorno virtual

# Acceso a bases de datos

## MongoDB

MongoDB	
	
Información general	
Tipo de programa	Base de datos, NoSQL
Desarrollador	MongoDB Inc.
Modelo de desarrollo	Software libre
Lanzamiento inicial	2009
Licencia	GNU AGPL v3.0 (drivers: licencia Apache)
Información técnica	
Programado en C++	
Versiones	
Última versión estable	4.0.8 (info) (29 de marzo de 2019 (3 años, 1 mes y 14 días))
Última versión en pruebas	4.1.9 (info) (13 de marzo de 2019 (3 años, 1 mes y 30 días))

• MongoDB Instalación de la base de datos Community Server  
<https://www.mongodb.com/try/download/community>

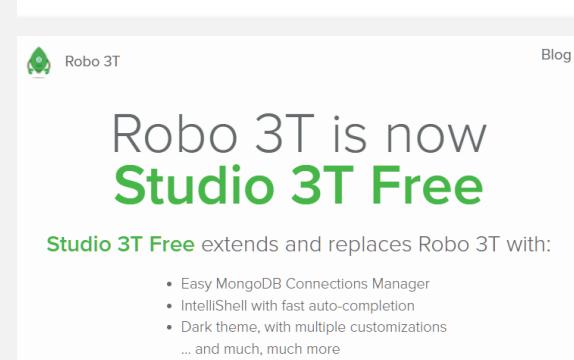


Choose which type of deployment is best for you

- Atlas MongoDB as a service
- On-premises MongoDB locally
- Tools Boost productivity
- Mobile & Edge Realm Datastore

Download MongoDB Community Server

• Robo 3T is now Studio 3T Free  
<https://robomongo.org/>



Robo 3T is now Studio 3T Free

Studio 3T Free extends and replaces Robo 3T with:

- Easy MongoDB Connections Manager
- IntelliShell with fast auto-completion
- Dark theme, with multiple customizations ... and much, much more

FREE FOREVER

# Acceso a bases de datos

## pymongo



### PyMongo

#### Introduction

Welcome to the documentation site for PyMongo, the official MongoDB driver for synchronous Python applications. Download it using [pip](#) or set up a runnable project by following our tutorial.



If you need to access MongoDB in a non-blocking manner or from co-routines, we recommend that you use the [Motor](#) driver instead.

- [Tutorial](#) on how to connect to MongoDB and run common operations.
- [API Reference](#)
- [Changelog](#)
- [Source Code](#)

- **pymongo** es el paquete que nos permite conectarnos a mongodb

- **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://www.mongodb.com/docs/drivers/pymongo/>

# Acceso a bases de datos

## MongoDb Conexión a la BBDD

```
from pymongo import MongoClient

clientDB = MongoClient('mongodb://localhost:27017/')
db = clientDB.admin
resultado = db.command('serverStatus')
print('Host:',resultado['host'])
print('Version:',resultado['version'])
print('Process:',resultado['process'])
```

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

# Acceso a bases de datos

## pymongo – trabajando con MongoDB

- `cliente.db.collection.find` búsqueda de documentos con o sin filtro
- `cliente.db.collection.find_one` búsqueda de documentos con o sin filtro
- `cliente.db.collection.insert_one` Inserta un documento en la colección
- `client.db.collection.insert_many` Inserta varios documentos en la colección
- `client.db.collection.update_one` Actualiza un documento en la colección
- `client.db.collection.update_many` Actualiza varios documentos en la colección
- `client.db.collection.delete_one` Borra un documento en la colección

# Acceso a bases de datos

## SQL Server

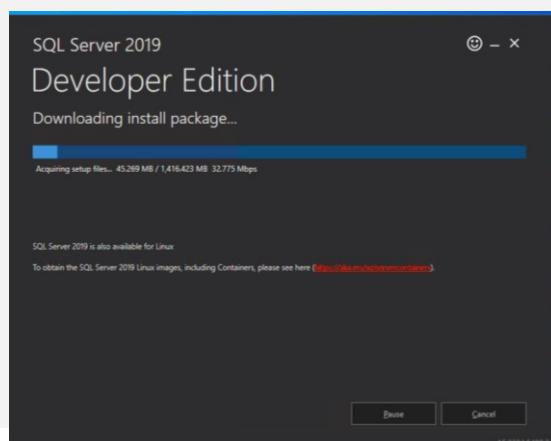


- Descargas de SQL Server | Microsoft

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

- Descargas de Microsoft Management Studio

<https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>



# Acceso a Base de datos

## pypdbc

- pyodbc es el paquete que me permite conectarme con Microsoft SQL Server y trabajar con sus base de datos, tablas y registros
- Instalación  
`pip install pyodbc`
- Documentación  
<https://pypi.org/project/pyodbc/#description>

[Controlador Python para SQL \(pyodbc\) –  
Python driver for SQL Server | Microsoft Docs](#)

pyodbc 4.0.32

`pip install pyodbc` 

DB API Module for ODBC

Navigation

Project description

 Project description

pyodbc

# Acceso a Base de datos

## SQL Server conexión a la BBDD

```
server = '(localdb)\ProjectsV13'
database = 'bikerentdb'
username = 'developer'
password = 'P4$$w0rd'

#Conecction String
driver='DRIVER={ODBC Driver 17 for SQL Server};'
others=f"SERVER={server};DATABASE={database};UID={username};PWD={password}"
connection_string='{}{}'.format(driver,others)
con = pyodbc.connect(connection_string)
cur = con.cursor()
res=cur.execute("SELECT @@VERSION AS 'SQL Server Version Details'")
for r in res:
    print(r[0])
```

- Existen varios controladores SQL de Python disponibles. Microsoft pone sus esfuerzos de prueba y su confianza en el controlador `pyodbc`.
- Si la ejecución del comando retorna filas podemos recorrerlas mediante un `while`
- Las posiciones de los campos en la fila nos permite acceder a la información del registro
- Otro driver disponible es `pymssql`

# Acceso a Base de datos

## Trabajando con MS-SQL

```
#create the Insert query
sql = f"""INSERT INTO customer (CustID,
CustName, CustAddress,CustPhone, CustEmail, CustCity)
VALUES ({cid}, '{cnm}', '{cad}', '{cph}', '{cem}', '{cct}')"""

#Execute SELECT statement
cur.execute("SELECT * FROM customer")
#Fetch all records from table
res = cur.fetchall()

#create update query
sql = f"""update customer set CustName='{cnm}' ,
CustAddress='{cad}',CustPhone='{cph}', CustEmail='{cem}' ,
CustCity='{cct}' where CustID={cid}"""

#Create Delete Query
sql = f"delete FROM customer where CustID = '{cid}'"
#execute delete query
cur.execute(sql)
#commit changes to DB
```

- TRANSACT-SQL es un lenguaje muy potente que nos permite definir casi cualquier tarea que queramos efectuar sobre la base de datos; incluye características propias de cualquier lenguaje de programación, características que nos permiten definir la lógica necesaria para el tratamiento de la información:
- Se realizan las operaciones CRUD (Crear, leer, actualizar y borrar (En inglés Create, Read, update and Delete).
- Las operaciones se confirman con **commit**
- Referencia a Transact-SQL : <https://bit.ly/3Nrktdn>

# 8 – Patrones de Diseño y Arquitectonico

# Patrones de diseño

## definciones

Son técnicas para resolver problemas recurrentes de diseño de software.



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)

# Patrones de diseño

## definciones



- **Creacionales** Soluciona la creación de objetos, hace un sistema independiente de como sus objetos son creados. Ejemplos: *Singleton, Factory*
- **Estructurales** Describe cómo los objetos se componen para formar estructuras complejas. Ejemplos: *Adapter, Decorator*
- **Comportamentales** Establece soluciones relacionadas con el comportamiento de la aplicación respecto a la interacción entre objetos y clases. Ejemplos: *Observer, State*

# Patrones Arquitectónico

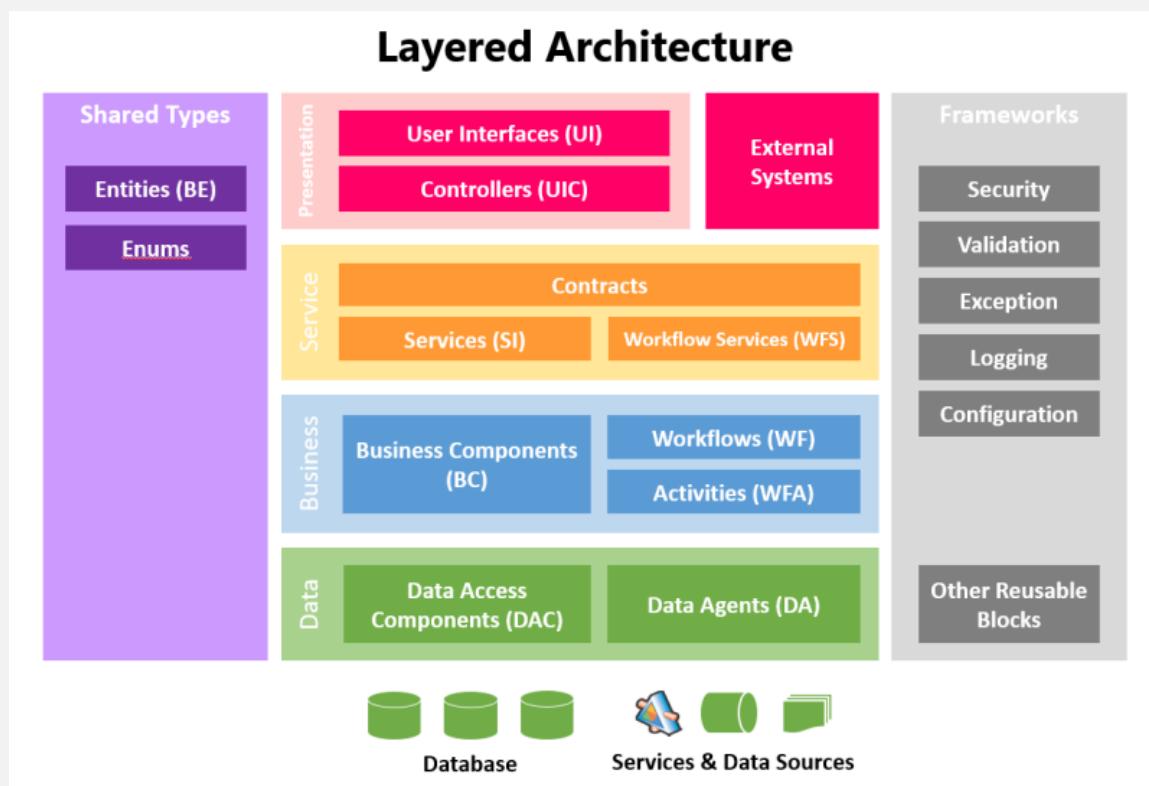
## definiciones



- Son patrones que ofrecen soluciones a problemas de arquitectura de software, dando:
  - descripción de los elementos
  - tipo de relaciones entre los diferentes elementos
  - restricciones sobre cómo pueden ser usados los elementos
- Un **patrón arquitectónico** expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones.
- En comparación con los patrones de diseño, los patrones arquitectónicos tienen una nivel de abstracción.

# Patrones Arquitectónico

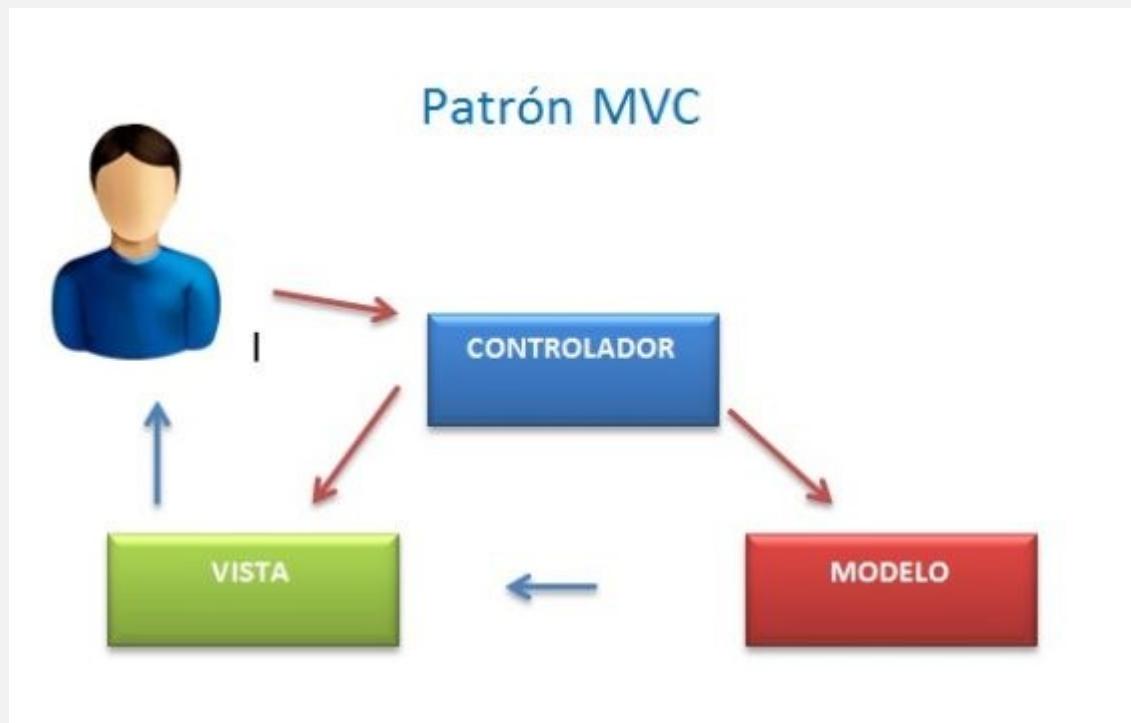
## Patrón Layer



- El **Patrón Layer** o La arquitectura en capas es un patrón de arquitectura software usada en la gran mayoría de sistemas. Se centra en una distribución jerárquica de las roles y responsabilidades proporcionando una separación efectiva de las preocupaciones (cada cual se encarga de lo que le corresponde).

# Patrones Arquitectónico

## Patrón MVC



- El **Patrón MVC** patrón arquitectural, un modelo o guía que expresa cómo organizar y estructurar los componentes de un sistema software, sus responsabilidades y las relaciones existentes entre cada uno de ellos.

# 9 – Aplicaciones Web

# Utilizando Flask

## Flask



Flask es un mini-framework escrito en Python y concebido para facilitar el desarrollo de aplicaciones Web bajo el patrón MVC

- <https://flask.palletsprojects.com>
- pip install Flask

# Utilizando Flask

- Practica realizar por cada ejercicio fundamentos de programación un programa en Python en entorno web.
- Para ello deberá seguir el patron del GitHub [GitHub - BillyClassTime/BillyClassTime-StartWithPythonFromCLAndWeb](#)
- El ejercicio No. 3 Determinar la ruta para llegar a una ciudad por avión es opcional.
- Fecha de entrega: 14 de abril de 2023

# 10 – Consumiendo API RESTful

# Consumiendo API RESTful

## Definiciones

- **Front end** y **back end** front end o frontal o interfaz de usuario, mientras que a back end se le llama servidor, motor o modo administrador.
- **MVC** (Interfaz de programación de aplicaciones)
- **API RESTful** es útil tanto para un humano como para el procesamiento automático de máquinas

# Consumiendo API RESTful

## requests



### Requests: HTTP para Humanos

Versión v1.1.0. ([Installation](#))

Requests es una librería para HTTP, ref:licenciada 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.

- <https://docs.python-requests.org/es/latest/>
- pip install requests

# Consumiendo API RESTful

## requests.get()

```
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'>
{'Server': 'nginx/1.10.3', 'Date': 'Mon, 23 May 2022 05:40:27
3', 'Connection': 'keep-alive', 'access-control-allow-origin': 'application/json
200/utf-8
{"message": "success", "iss_position": {"longitude": "160.6013
success
```

- **requests** es el objeto que representa la conexión http
- **get** Es utilizado únicamente para consultar información al servidor, muy parecidos a realizar un SELECT a la base de datos..
- 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

# Consumiendo API RESTful

## requests.post()

```
import requests
u={'Nombre':'Billy','Apellidos':'Vanegas'}
r=requests.post('http://postman-echo.com/post',data=u)
print(r.text)
```

- **requests** es el objeto que representa la conexión http
- **post** utilizado para solicitar la creación de un nuevo registro, es decir, algo que no existía previamente, es decir, es equivalente a realizar un INSERT en la base de datos.
- **data** es el parámetro utilizado para enviar los datos en la petición

```
{"args":{}, "data": "", "files": {}, "form": {"Nombre": "Billy", "Apellidos": "Vanegas"}, "headers": {"x-forwarded-proto": "http", "x-forwarded-port": "80", "host": "postman-echo.com", "x-amzn-trace-id": "Root=1-628b23e3-0098c0222ad9f1375c4ccebb", "content-length": "30", "user-agent": "python-requests/2.27.1", "accept-encoding": "gzip, deflate", "accept": "*/*", "content-type": "application/x-www-form-urlencoded"}, "json": {"Nombre": "Billy", "Apellidos": "Vanegas"}, "url": "http://postman-echo.com/post"}
```

# Consumiendo API RESTful

## headers

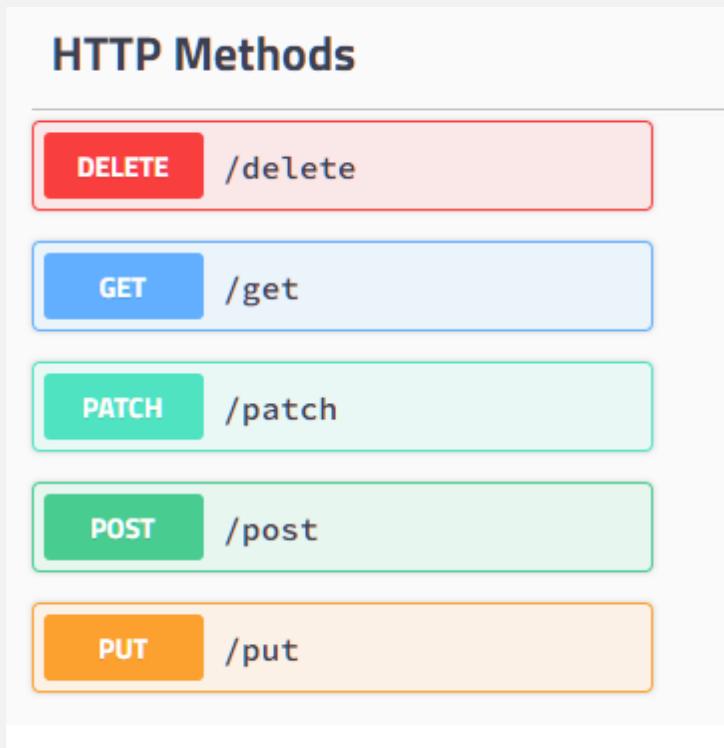
```
import requests
import json
url='http://httpbin.org/post'
user = json.dumps({'Username':'Billy'})
headers={'content-type':'application/json'}
r = requests.post(url,data=user,headers=headers)
print(r.text)
```

```
{
  "args": {},
  "data": "{\"Username\": \"Billy\"}",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Content-Length": "21",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
```

- **headers** es el parámetro utilizado para enviar la información de la cabecera en la petición, personalizando la información de envío.
- La información de la cabecera se representa mediante diccionarios.

# Consumiendo API RESTful

## Resumen métodos HTTP



- **DELETE**: se utiliza para eliminar un registro existente, es similar a DELETE a la base de datos.
- **GET**: Es utilizado únicamente para consultar información al servidor, muy parecidos a realizar un SELECT a la base de datos.
- **PATCH**: Es utilizado para actualizar un registro existente, es equivalente a realizar un UPDATE a la base de datos.
- **POST**: Es utilizado para solicitar la creación de un nuevo registro, equivalente a realizar un • **INSERT** en la base de datos.
- **PUT**: Se utiliza para actualizar por completo un registro existente, es decir, es parecido a realizar un UPDATE a la base de datos

© Copyright Microsoft Corporation. All rights reserved.

