Workshop

DESARROLLO BACKEND CON PYTHON

con el aval de Tecnológico Espíritu Santo

CapaciES
Entrenando a Grandes Ejecutivos

Workshop

INSTRUCTOR

GABRIEL VILLACIS

Ingeniero en Sistemas Computacionales
Candidato a Máster en Dirección y Gestión de TI
Líder Técnico
Arquitecto, Desarrollador políglota, Instructor y Conferencista de Software
#pythonista

Workshop

ENCUESTA INICIAL

Workshop

PYTHON

Python es un lenguaje de programación de alto nivel creado por Guido van Rossum en la década del 90.
Es conocido por su simplicidad y legibilidad, lo que lo hace ideal para principiantes en programación, sin embargo, también es utilizado ampliamente en aplicaciones de alto rendimiento.

Su nombre proviene de la serpiente, sino que se inspiró en el grupo de comedia británico "Monty Python" del cual van Rossum era fanático.

CARACTERÍSTICAS DE PYTHON (I)

1. **Sintaxis clara y legible**
   Python se destaca por su sintaxis fácil de leer y escribir.

2. **Lenguaje interpretado**
   El código se ejecuta línea por línea por un intérprete.

3. ***Tipado dinámico y fuerte***
   No necesitas declarar el tipo de dato de una variable antes de usarla. Además, no se permiten conversiones automáticas entre tipos de datos incompatibles.

4. ***Orientado a objetos***
   Python es orientado a objetos, organiza el código en torno a clases y objetos, lo que facilita la modularidad y la reutilización del código.

CARACTERÍSTICAS DE PYTHON (II)

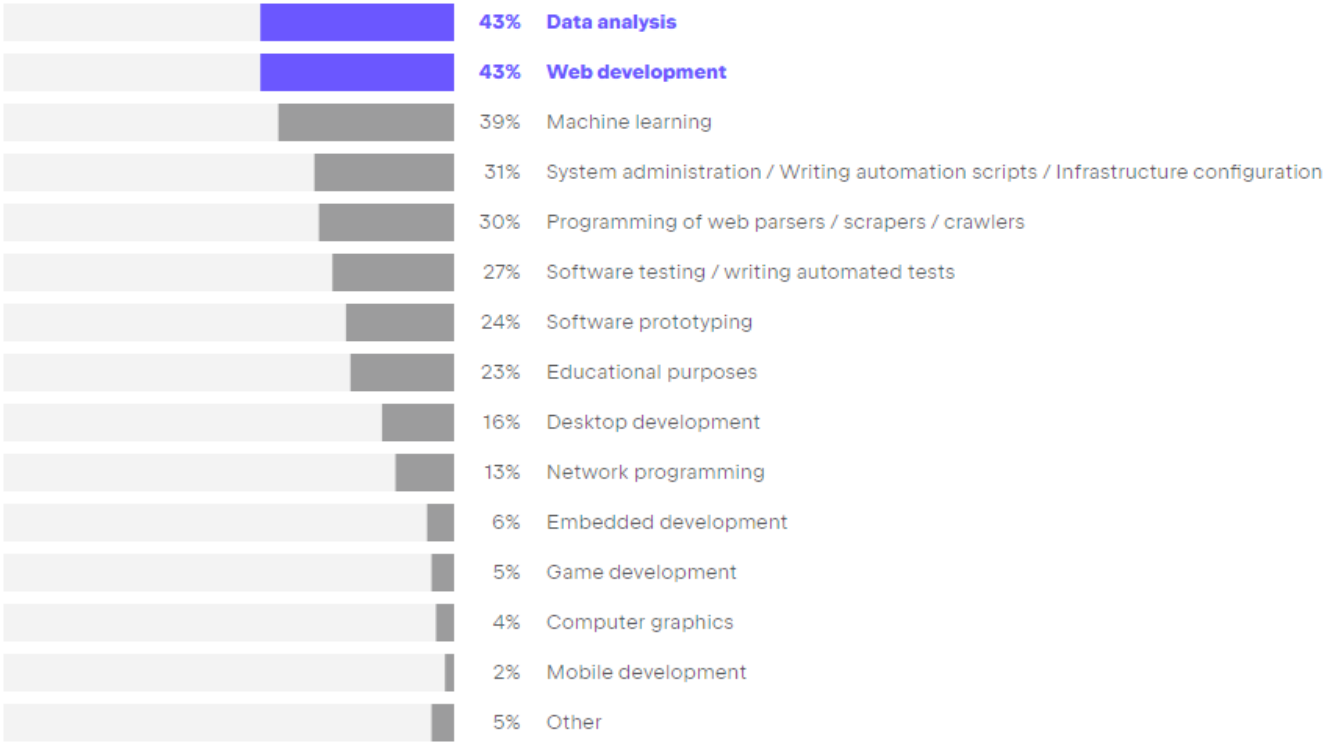
5. **Soporte para programación funcional**
   Python admite la programación funcional. Esto es especialmente útil para trabajar con transformaciones de datos, filtrado y cálculos matemáticos.

6. **Amplia biblioteca estándar y ecosistema**
   La biblioteca estándar de Python incluye una gran variedad de módulos y funciones listas para usar, abarcando tareas básicas y avanzadas. Además, la comunidad desarrolla continuamente bibliotecas adicionales, lo cual amplía su capacidad y versatilidad.

7. **Multiplataforma**
   Es compatible con una gran variedad de sistemas operativos.

8. **Comunidad activa**
   Tiene una comunidad de desarrollo muy activa y multitud de recursos en línea.

CASOS DE USO DE PYTHON



Python Progamming Survey 2022 – Jetbains

Workshop

ENTORNO DE DESARROLLO

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *VARIABLES Y TIPOS DE DATOS*

VARIABLES

Las variables son como etiquetas que se utilizan para almacenar datos en la memoria de la computadora. Podemos pensar en ellas como cajas en las que guardamos valores, como números, texto o cualquier otro tipo de información.

TIPOS DE DATOS

Python es un lenguaje de tipado dinámico, lo que significa que no es necesario declarar el tipo de una variable de antemano; Python lo deduce automáticamente.

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *DEFINIENDO VARIABLES*

```
# Entero (int)
edad = 30

# Flotante (float)
altura = 1.75

# Cadena de Caracteres (str)
nombre = "Juan Pérez"

# Booleano (bool)
es_mayor_de_edad = True

# Lista (list)
numeros = [1, 2, 3, 4, 5]

# Tupla (tuple)
coordenadas = (10, 20)

# Diccionario (dict)
persona = {'nombre': 'Ana', 'edad': 25}

# None (NoneType)
resultado = None
```

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *MANIPULACIÓN DE CADENAS DE CARACTERES*

Es la capacidad de trabajar con texto de diversas formas. Las operaciones comunes incluyen calcular la longitud de una cadena, concatenar cadenas, cambiar entre mayúsculas y minúsculas, reemplazar partes de una cadena y dividir una cadena en partes más pequeñas, entre otras. Estas operaciones son fundamentales al trabajar con datos de texto y son ampliamente utilizadas en aplicaciones de procesamiento de lenguaje natural, análisis de datos, desarrollo web, entre otros.

Manejo de cadenas de caracteres en Python

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *OPERADORES MATEMÁTICOS*

Python admite una amplia variedad de operaciones matemáticas, que incluyen:

Operador	Descripción
+	Suma dos operandos.
–	Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	Producto/Multiplicación de dos operandos.
/	Divide el operando de la izquierda por el de la derecha (el resultado siempre es un `float`).
%	Operador módulo. Obtiene el resto de dividir el operando de la izquierda por el de la derecha.
//	Obtiene el cociente entero de dividir el operando de la izquierda por el de la derecha.
**	Potencia. El resultado es el operando de la izquierda elevado a la potencia del operando de la derecha.

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *OPERADORES DE COMPARACIÓN*

Los operadores de comparación se utilizan, como su nombre indica, para comparar dos o más valores. El resultado de estos operadores siempre es True o False.

Operador	Descripción
>	Mayor que. `True` si el operando de la izquierda es estrictamente mayor que el de la derecha; `False` en caso contrario.
>=	Mayor o igual que. `True` si el operando de la izquierda es mayor o igual que el de la derecha; `False` en caso contrario.
<	Menor que. `True` si el operando de la izquierda es estrictamente menor que el de la derecha; `False` en caso contrario.
<=	Menor o igual que. `True` si el operando de la izquierda es menor o igual que el de la derecha; `False` en caso contrario.
==	Igual. `True` si el operando de la izquierda es igual que el de la derecha; `False` en caso contrario.
!=	Distinto. `True` si los operandos son distintos; `False` en caso contrario.

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *OPERADORES LÓGICOS*

Los operadores lógicos permiten comparar valores que se encuentran representados de forma explícita o a través de variables; esta comparación arroja un resultado True o False dependiendo del operador.

Operador	Nombre	Ejemplo
and	Devuelve True si ambos operandos son True	True and True = True
or	Devuelve True si al menos un operando es True	True or False = True
not	Devuelve el contrario, True si es Falso y viceversa	not True = False

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *OPERADORES DE PERTENENCIA*

Los operadores de pertenencia se utilizan para comprobar si un valor o variable se encuentran en una secuencia (list, tuple, dict, set o str).

Operador	Descripción
in	Devuelve True si el valor se encuentra en una secuencia; False en caso contrario.
not in	Devuelve True si el valor no se encuentra en una secuencia; False en caso contrario.

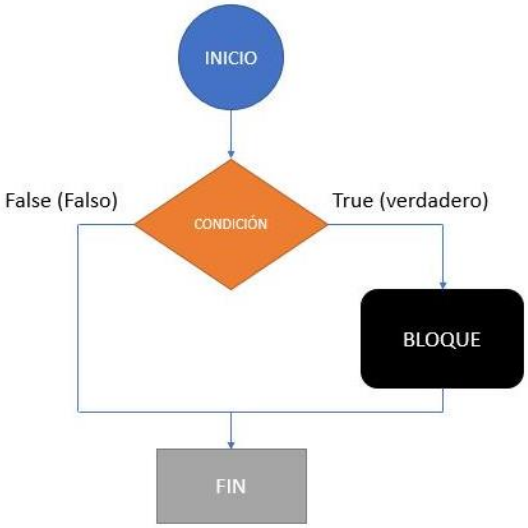
Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *ESTRUCTURA DE CONTROL IF/ELIF/ELSE (I)*

Las sentencias de control (if/elif/else) se utilizan para controlar el flujo de un programa en función de una condición. Permiten que el programa tome decisiones y ejecute diferentes bloques de código según si una condición es verdadera o falsa.

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *CONDICIONAL SIMPLE*

Condicional simple:
Ejecuta un bloque de instrucciones cuando la proposición (condición) es verdadera; si es falsa, no hace nada. En inglés "if" significa "si" (condición).

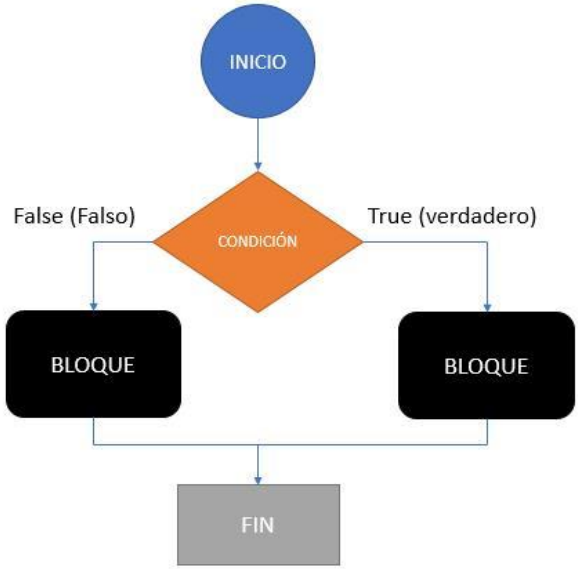



```
if condicion:
    # sentencia1
    # sentencia2
    # sentencia3
```

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *CONDICIONAL DOBLE*

Condicional doble:

La estructura de control if... else... permite que el programa ejecute unas instrucciones cuando se cumple una condición y otras instrucciones cuando no se cumple esa condición. En ingles "if" significa "si" (condición) y "else" significa "si no".

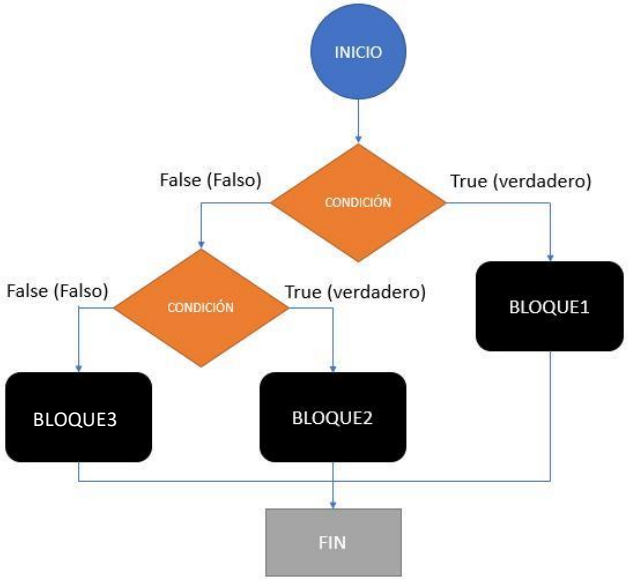



```
if condicion:
        # bloque de sentencias
else:
        # otro bloque de sentencias
```

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *CONDICIONAL MÚLTIPLE*

Condicional múltiple:

La estructura de control if...elif...elif... permite la ejecución de múltiples condiciones de forma jerárquica, es decir, si no se cumple la primera condición se evalúa la siguiente condición y así sucesivamente.

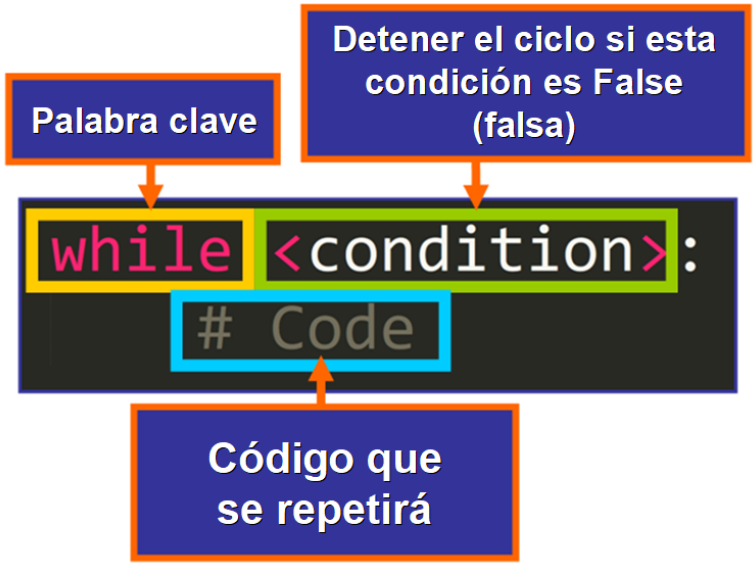



```
if condicion 1:
        # bloque de sentencias 1

elif condicion 2:
        # bloque de sentencias 2
.
.
.
else:
        # bloque de sentencias 3
```

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *ESTRUCTURA REPETITIVA WHILE*

La sentencia o bucle while en Python es una sentencia de control de flujo que se utiliza para ejecutar un bloque de instrucciones de forma continuada mientras se cumpla una condición determinada.

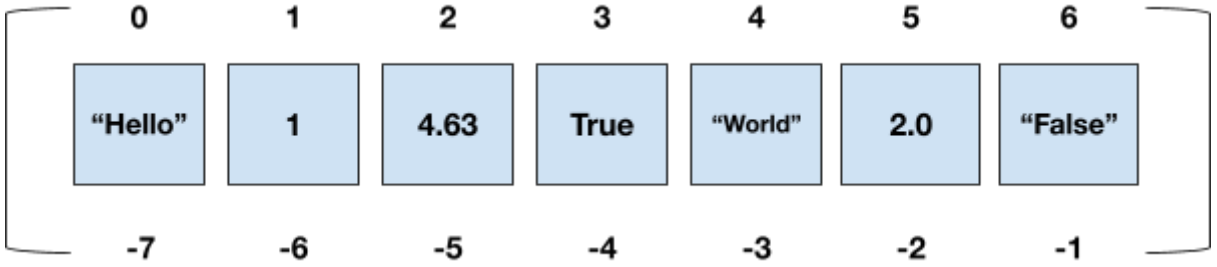


Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *LISTAS (I)*

Las listas en Python son un tipo contenedor, compuesto, que se usan para almacenar conjuntos de elementos relacionados del mismo tipo o de tipos distintos. Los elementos de una lista están ordenados y se puede acceder a ellos mediante el número de índice.

Junto a las clases tuple, range y str, son uno de los tipos de secuencia, con la particularidad de que son mutables. Esto último quiere decir que su contenido se puede modificar después de haber sido creada.



Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *LISTAS (II)*

Para crear una lista en Python, simplemente hay que encerrar una secuencia de elementos separados por comas entre corchetes [].

Por ejemplo, para crear una lista con los números del 1 al 10 se haría del siguiente modo:
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Las listas pueden almacenar elementos de distinto tipo:
elementos = [3, 'a', 8, 7.2, 'hola']

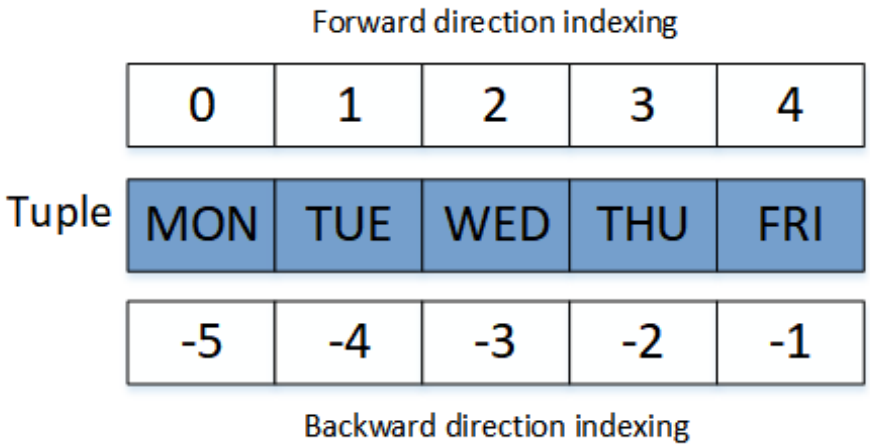
Incluso pueden contener otros elementos compuestos, como objetos u otras listas:
lista = [1, ['a', 'e', 'i', 'o', 'u'], 8.9, 'hola']

Manejo de listas en Python

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *TUPLAS (I)*

Puede contener elementos de diferentes tipos de datos. Los elementos de una tupla están ordenados y se puede acceder a ellos mediante su número de índice. A diferencia de las listas, las tuplas son inmutables y, por tanto, los elementos de una tupla no se pueden cambiar.



Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *TUPLAS (II)*

En general, para crear una tupla en Python simplemente hay que definir una secuencia de elementos separados por comas.

Por ejemplo, para crear una tupla con los números del 1 al 5 se haría del siguiente modo:
numeros = 1, 2, 3, 4, 5

Como te indicaba, la clase tuple también puede almacenar elementos de distinto tipo:
elementos = 1, ['a', 'e', 'i', 'o', 'u'], 8.9, ‘hola’

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *TUPLAS (III)*

Para crear una tupla vacía, usa paréntesis () o el constructor de la clase tuple() sin parámetros.

Para crear una tupla con un único elemento: elem, o (elem,). Observa que siempre se añade una coma.

Para crear una tupla de varios elementos, sepáralos con comas: a, b, c o (a, b, c). El uso de parámetros es opcional.

Las tuplas también se pueden crear usando el constructor de la clase, tuple(iterable). En este caso, el constructor crea una tupla cuyos elementos son los mismos y están en el mismo orden que los ítems del iterable. El objeto iterable puede ser una secuencia, un contenedor que soporte la iteración o un objeto iterador.

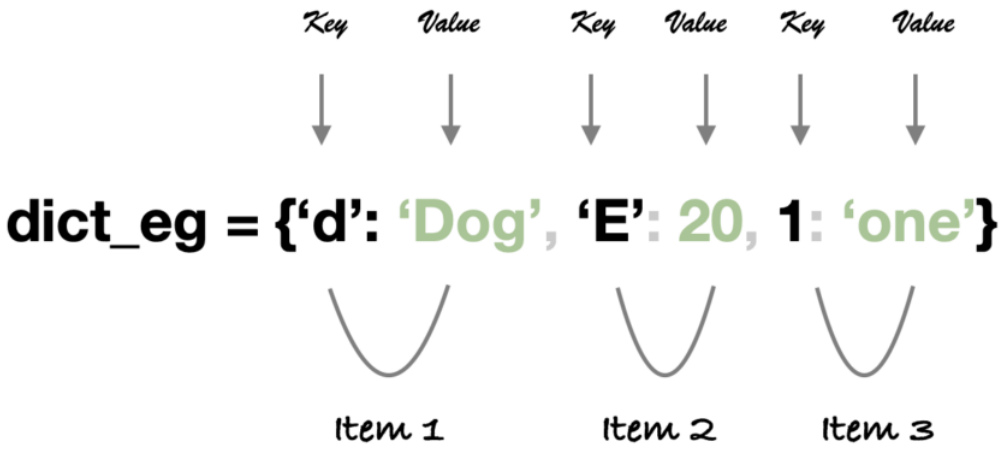
Manejo de tuplas en Python

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *DICCIONARIOS (I)*

Los diccionarios en Python son una estructura de datos que permite almacenar su contenido en forma de llave y valor.

Un diccionario en Python es una colección de elementos, donde cada uno tiene una llave key y un valor value. Los diccionarios se pueden crear con paréntesis {} separando con una coma cada par key: value.



Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *DICCIONARIOS (II)*

Tipos de clave-valor permitidos:

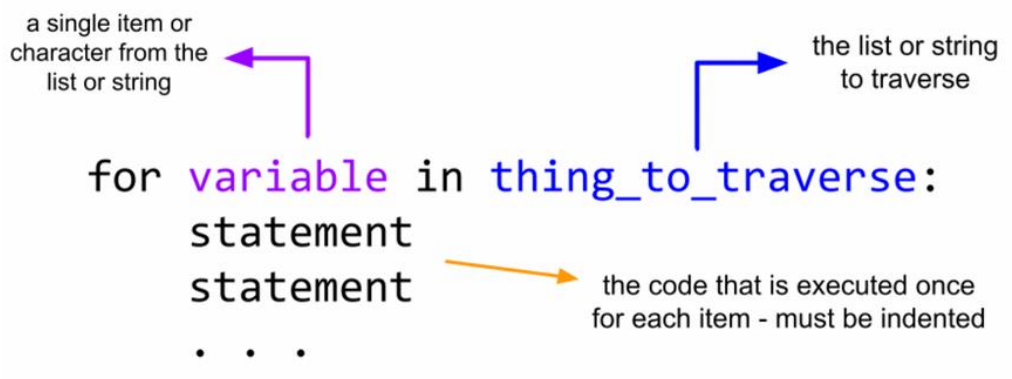
- Claves: no puede haber claves duplicadas y éstas tienen que ser objetos inmutables.
- Valores: no existe ninguna restricción en cuanto a los tipos de datos.

Manejo de diccionarios en Python

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *BUCLE FOR*

El bucle for se utiliza para recorrer los elementos de un objeto iterable (lista, tupla, conjunto, diccionario u otro iterable) y ejecutar un bloque de código. En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.



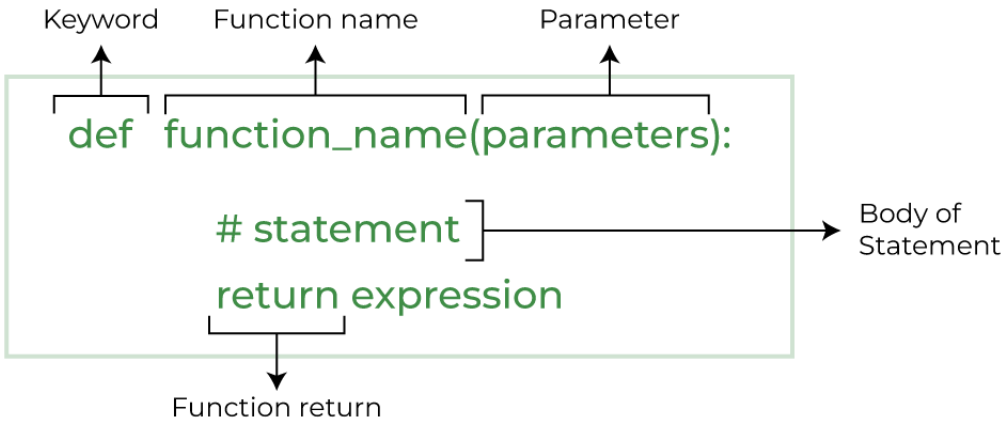
Usos del bucle for

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *FUNCIONES (I)*

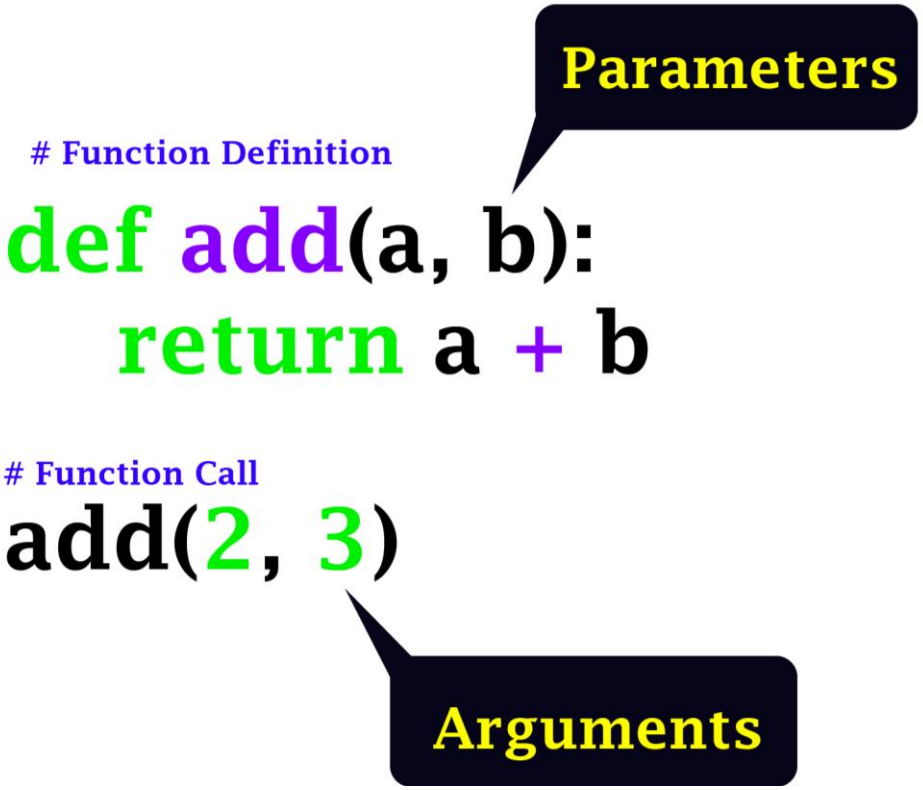
Una función es un bloque de código que sólo se ejecuta cuando se llama. Puede pasar datos, conocidos como parámetros, a una función.

Una función puede devolver datos como resultado.



Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *FUNCIONES (II)*



Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *BLOQUE TRY/EXCEPT*

Al programar en Python algunas veces podemos anticipar errores de ejecución, incluso en un programa sintáctica y lógicamente correcto, pueden llegar a haber errores causados por entrada de datos inválidos o inconsistencias predecibles.

En Python, puedes usar los bloques try y except para manejar estos errores como excepciones.

Manejo de excepciones

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *PROGRAMACIÓN ORIENTADA A OBJETOS*

La programación orientada a objetos (POO) **es un enfoque** (método/paradigma) para el desarrollo de sistemas **que se centra en los objetos y la forma en que interactúan**.

Un objeto es cualquier cosa que podamos nombrar y describir. Los objetos son "cosas" que queremos modelar dentro de nuestros programas y, por lo tanto, **el objetivo de un programa orientado a objetos es representar el mundo real en código.**

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *¿QUÉ ES UN OBJETO?*

Un objeto representa una entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema.

Puede ser: una persona, un lugar, una cuenta bancaria, una tabla de datos o cualquier elemento que el programa deba manejar.

Los objetos constituyen la unidad básica de un sistema orientado a objetos en tiempo de ejecución.

Tienen dos componentes:

Datos: conocidos como atributos.
Comportamientos: conocidos como métodos.

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON:
ATRIBUTOS Y MÉTODOS DE UN OBJETO: <TAXI>

Atributos

Definen las propiedades o características del objeto

Conductor
Está disponible?
Tarifa
No. de Pasajeros
Ubicación

Métodos

Definen los comportamientos del objeto

Recorrer
Mostrar disponibilidad
Embarcar pasajeros
Encender taxímetro
Dejar pasajeros

Los valores actualmente asignados a los atributos de un objeto constituyen su Estado.

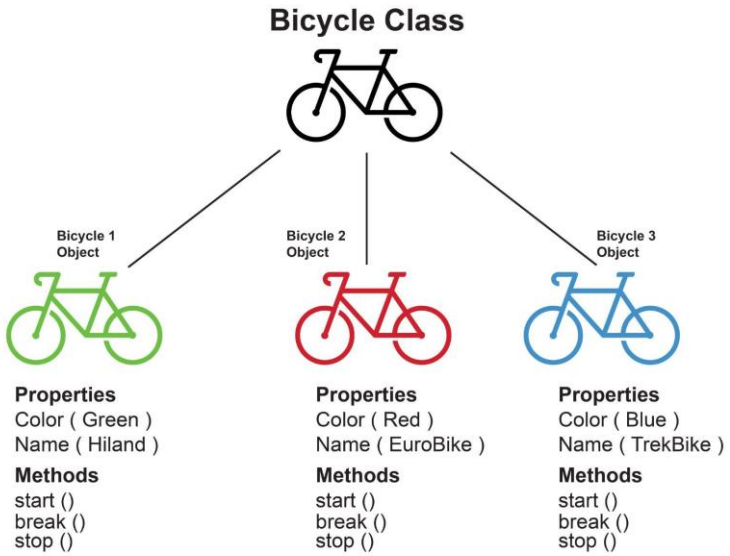
Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON:
ATRIBUTOS Y MÉTODOS DE UN OBJETO: TARJETA

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *¿QUÉ ES UN CLASE?*

Una clase es una plantilla (molde/prototipo/tipo de datos personalizado) que describe los atributos y métodos de un conjunto de objetos homogéneos.



Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *TERMINOLOGÍA CLASE/OBJETO*

Después de crear la clase “Bicicleta”, se debe crear una instancia de ella antes de poder usarla.

Cuando se crea una instancia de la clase, se está creando un objeto de ese tipo.

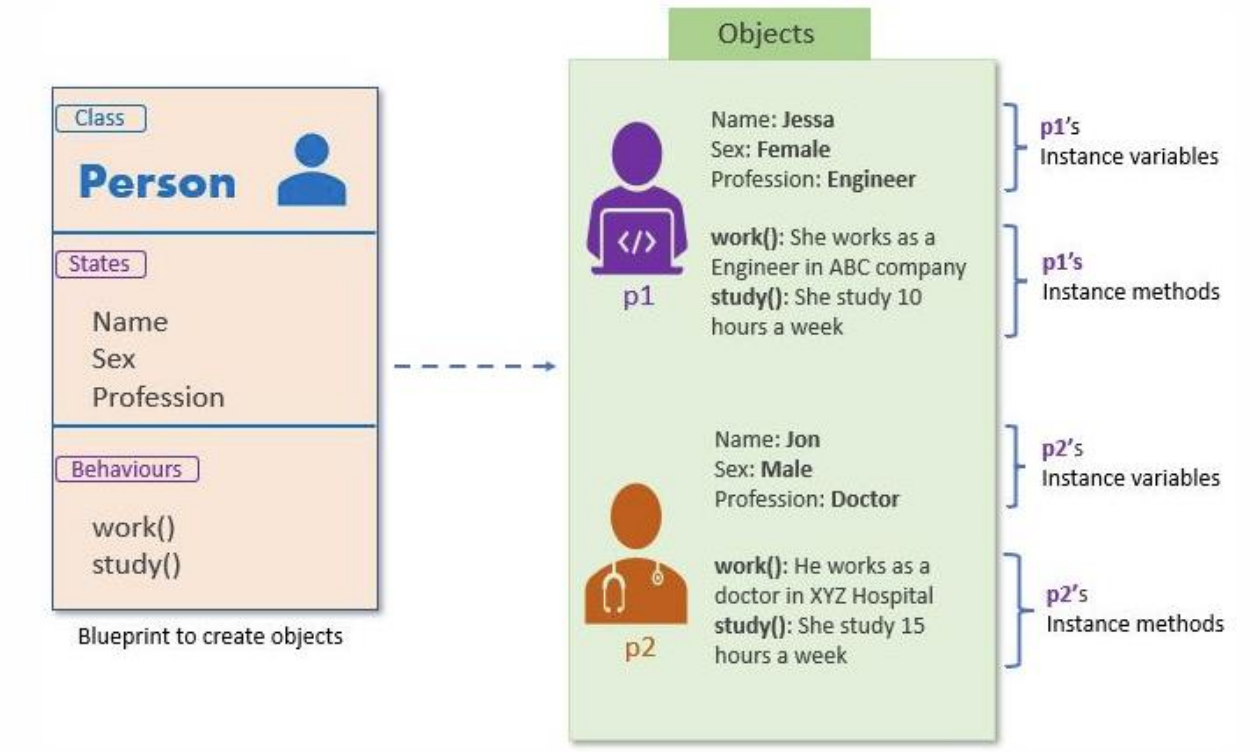
Una clase es una plantilla que define de forma genérica cómo van a ser los objetos de un determinado tipo y un objeto es una instancia de esa clase.

Cuando se crean objetos, éstos implícitamente obtienen todos los atributos y métodos definidos en la clase.

“La clase es lo que defines en el código. El objeto es lo que obtienes cuando ejecutas el código.”

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *EJEMPLO CLASE/OBJETO*

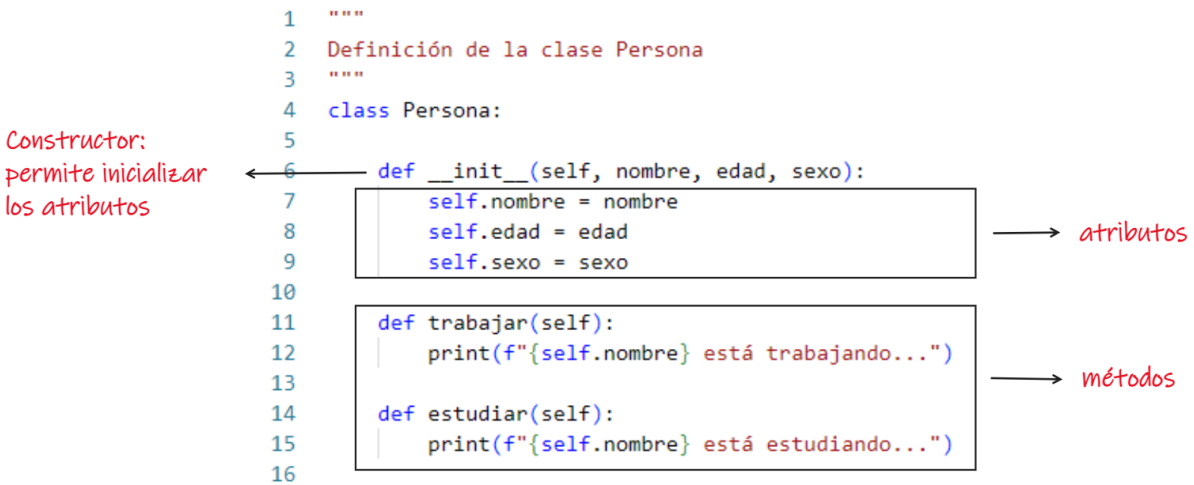


Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *SINTAXIS DE UNA CLASE*

Para definir una clase en Python, se utiliza la palabra reservada ***class***.
Por convención, el nombre de la clase debe iniciar con mayúscula.

La sintaxis/estructura es la siguiente:




```
1  """
2  Definición de la clase Persona
3  """
4  class Persona:
5
6      def __init__(self, nombre, edad, sexo):
7          self.nombre = nombre
8          self.edad = edad
9          self.sexo = sexo
10
11     def trabajar(self):
12         print(f"{self.nombre} está trabajando...")
13
14     def estudiar(self):
15         print(f"{self.nombre} está estudiando...")
16
```

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *GETTERS Y SETTERS*

En POO, los métodos que se dedican a acceder y cambiar atributos generalmente se denominan getters y setters.

Getters (Acceder a los atributos): Se utilizan para evitar el acceso directo a los atributos privados.

Setters (Cambiar los atributos): Permiten agregar lógica de validación para establecer valores a los atributos.

```python
class Billetera:
    def __init__(self) -> None:
        self.__saldo = 0

    def get_saldo(self) -> float:
        return self.__saldo

    def set_saldo(self, nuevo_saldo: float) -> None:
        if nuevo_saldo > 0:
            self.__saldo = nuevo_saldo
```

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *@property (ATRIBUTOS ADMISTRADOS)*

El decorador @property es la forma pitónica de implementar métodos getter y setter.

```python
class Empleado:
    def __init__(self):
        self.__sueldo = 425

    # Método getter para atributo sueldo
    @property
    def sueldo(self) -> float:
        return self.__sueldo

    # Método setter para atributo sueldo
    @sueldo.setter
    def sueldo(self, nuevo_sueldo: float) -> None:
        if nuevo_sueldo > 0:
            self.__sueldo = nuevo_sueldo
```

El método getter, es decir, el decorador @property, debe introducirse antes que el método setter o habrá un error cuando se ejecute la clase. Esto se debe a que el decorador @property define el nombre del atributo ofrecido al cliente. El método setter, agregado con .setter, simplemente le agrega una nueva funcionalidad.

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *HERENCIA*

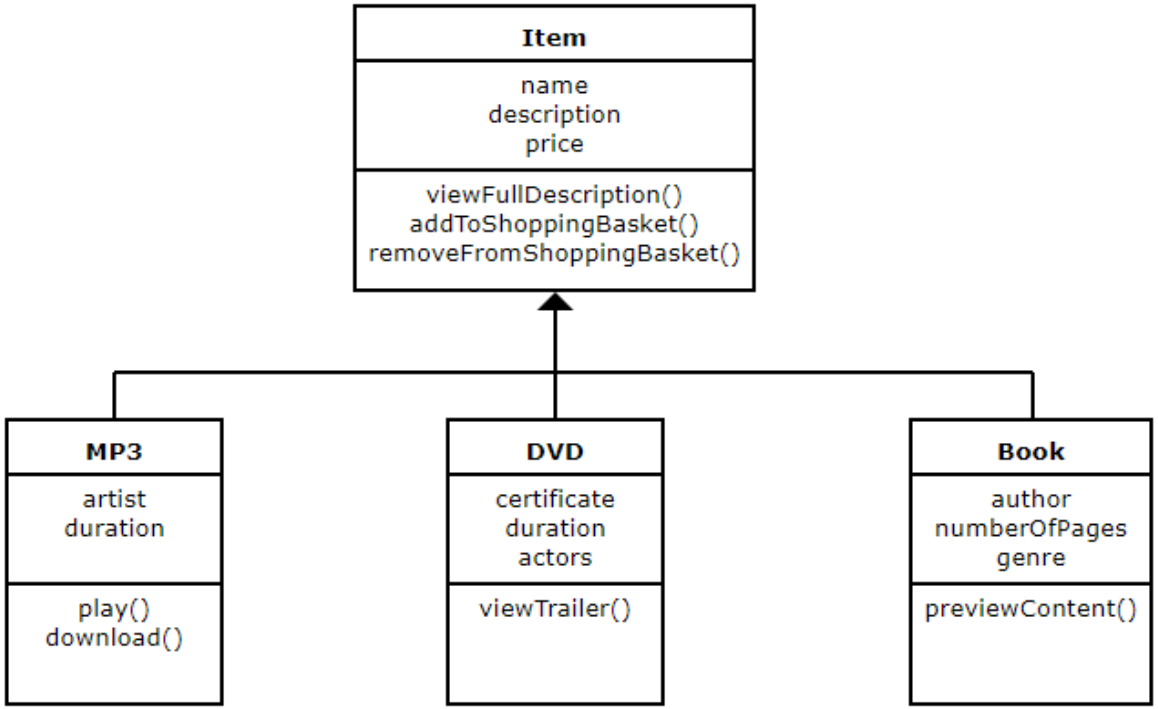
Una clase puede heredar los rasgos (atributos y métodos) de otra clase.

El proceso de heredar los atributos y métodos de la clase principal en una clase secundaria se denomina herencia.

La clase existente se denomina clase base, clase principal o superclase y la nueva clase se denomina subclase, clase secundaria o clase derivada.

Además de las definiciones heredadas, una clase también puede contener definiciones que son exclusivas de ella.

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *EJEMPLO HERENCIA*



Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *IMPLEMENTANDO LA HERENCIA*

```python
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * self.length + 2 * self.width

class Square(Rectangle):
    def __init__(self, length):
        super().__init__(length, length)
```

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *IMPORTANDO MÓDULOS*

Module1.py

```
# A module named Module1.py

# The module declares a class
# named Hello
class Hello:
    def Show(self):
        print("Module1.Hello.Show")
```

Module2.py

```
# A module named Module2.py

# To access the Hello class
# of Module1, you must
# import it
import Module1

# Referring to the Hello class
# of Module1.py
obj = Module1.Hello()
obj.Show() # Module1.Hello.Show
```

[Manejo de módulos en Python]

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *LIBRERÍA ESTÁNDAR*

Algunos de los módulos/paquetes más utilizados son:

random	Se utiliza para generar números pseudo aleatorios.
datetime	Permite trabajar con fechas y horas.
pathlib	Permite manipular rutas de sistemas de archivos de forma agnóstica al sistema operativo.
csv	Para lectura y escritura de archivos con formato CSV.
email	Permite manejar mensajes de correo electrónico.
re	Para realizar operaciones con expresiones regulares.
math	Facilita la ejecución de cálculos matemáticos.
sqlite3	Para trabajar con bases de datos SQLite.
venv	Permite la creación de entornos virtuales.

Librería estándar de Python

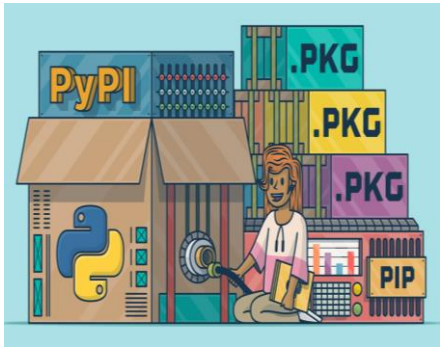
Workshop

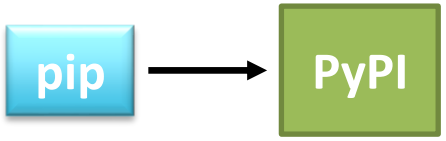
FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *PYTHON PACKAGE INDEX (PyPI)*

- https://pypi.org/
- Es el repositorio oficial de software (paquetes) de terceros para el lenguaje de programación Python.
- PyPI le ayuda a encontrar e instalar software desarrollado y compartido por la comunidad de Python.
- Los autores de paquetes usan PyPI para distribuir su software.
- Paquetes más utilizados: https://hugovk.github.io/top-pypi-packages/

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *PIP*

- pip = gestor de paquetes. Es una herramienta que le permite instalar y administrar bibliotecas y dependencias adicionales que no se distribuyen como parte de la biblioteca estándar de Python.
- La administración de paquetes es tan importante que pip se ha incluido con el instalador de Python desde la versión 3.4.
- Es utilizado por la mayoría de proyectos de Python, lo que lo convierte en una herramienta esencial para todos los Pythonistas.
- Puede verificar que pip está disponible ejecutando el siguiente comando en su consola: pip --version





Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *USANDO PIP*

- **python -m pip install –upgrade pip:** Actualizar pip
- **pip help:** Mostrar la ayuda de pip
- **pip install “paquete1” “paquete2”:** Instalar uno o más paquetes (separados por espacio vacío)
- **pip install –r FILENAME:** Instalar todos los paquetes enumerados en el archivo dado. Esta opción se puede utilizar varias veces.
- **pip list:** Mostrar los paquetes instalados
- **pip freeze:** Mostar los paquetes instalados en formato “requirements”
- **pip show “paquete”:** Mostrar el detalle de un paquete
- **pip uninstall “paquete”:** Desinstalar un paquete
- **pip freeze > requirements.txt**
- **pip uninstall -r requirements.txt -y**

Workshop

FUNDAMENTOS DE PROGRAMACIÓN PYTHON: *MANEJO DE LAS VERSIONES DE LOS PAQUETES*

- **pip install “paquete”:** Instala la última versión del paquete.

- **pip install –upgrade “paquete”:** Actualiza a la última versión del paquete.

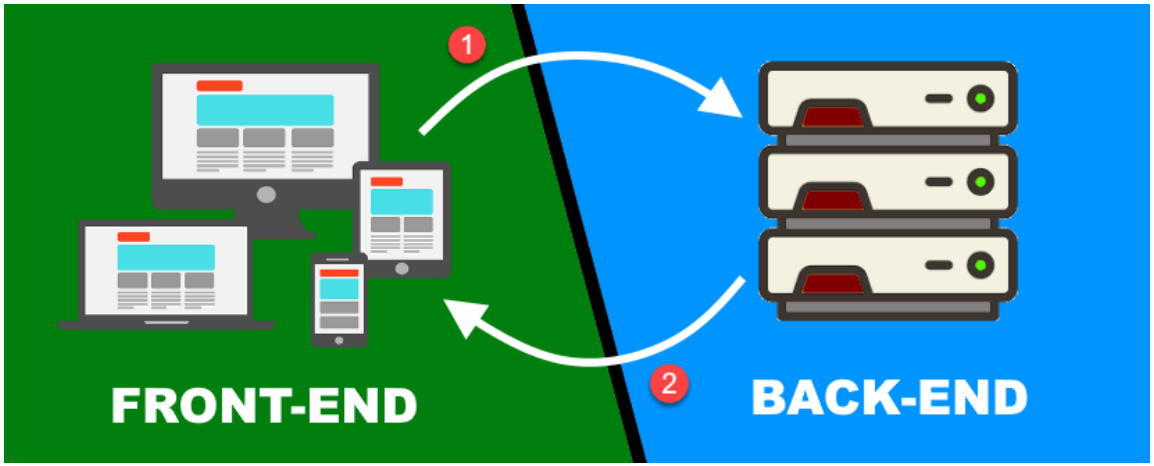
- **pip install “paquete==1.4.0”:** Instala la versión específica del paquete (1.4.0)

- **pip install “paquete>=1,<2”:** Instala la versión mayor o igual que una (1) y menor que otra (2)

- **pip install “paquete~=1.4.2”:** Instala una versión compatible. En este caso, esto significa instalar cualquier versión "==1.4.*" versión que también sea ">=1.4.2".

Workshop

ARQUITECTURA DE UNA APLICACIÓN WEB (I)

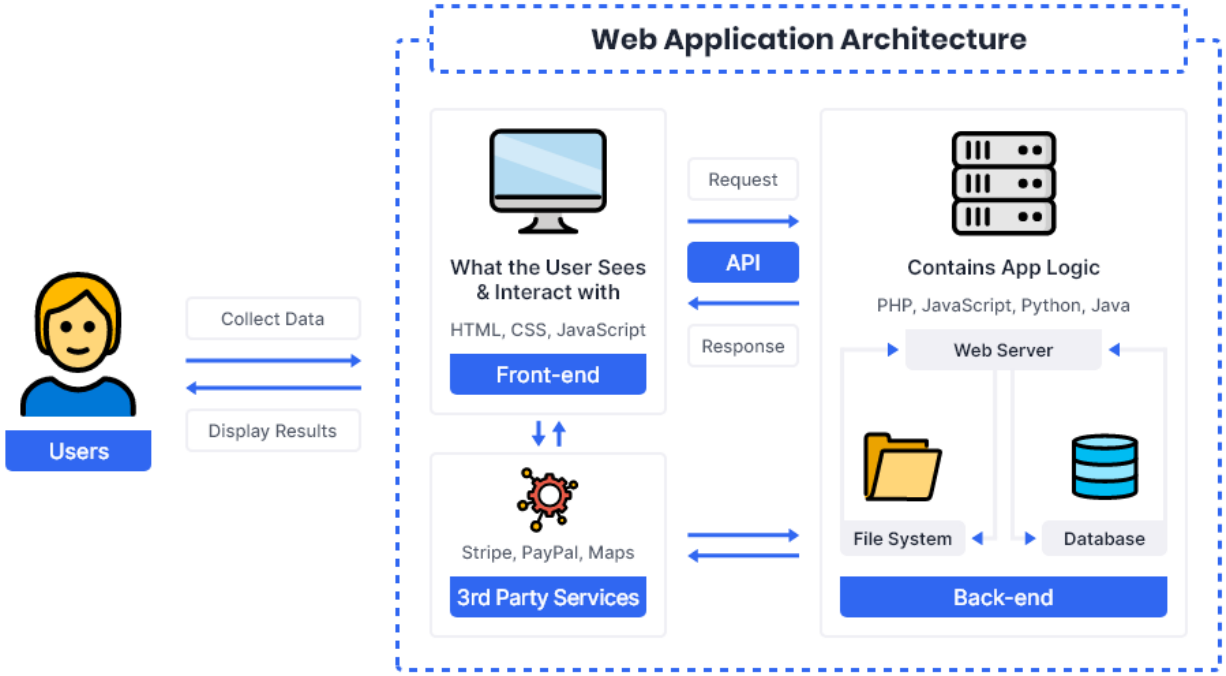


- Apariencia
- Interacción con los usuarios

- Cerebro
- Invisible para los usuarios

Workshop

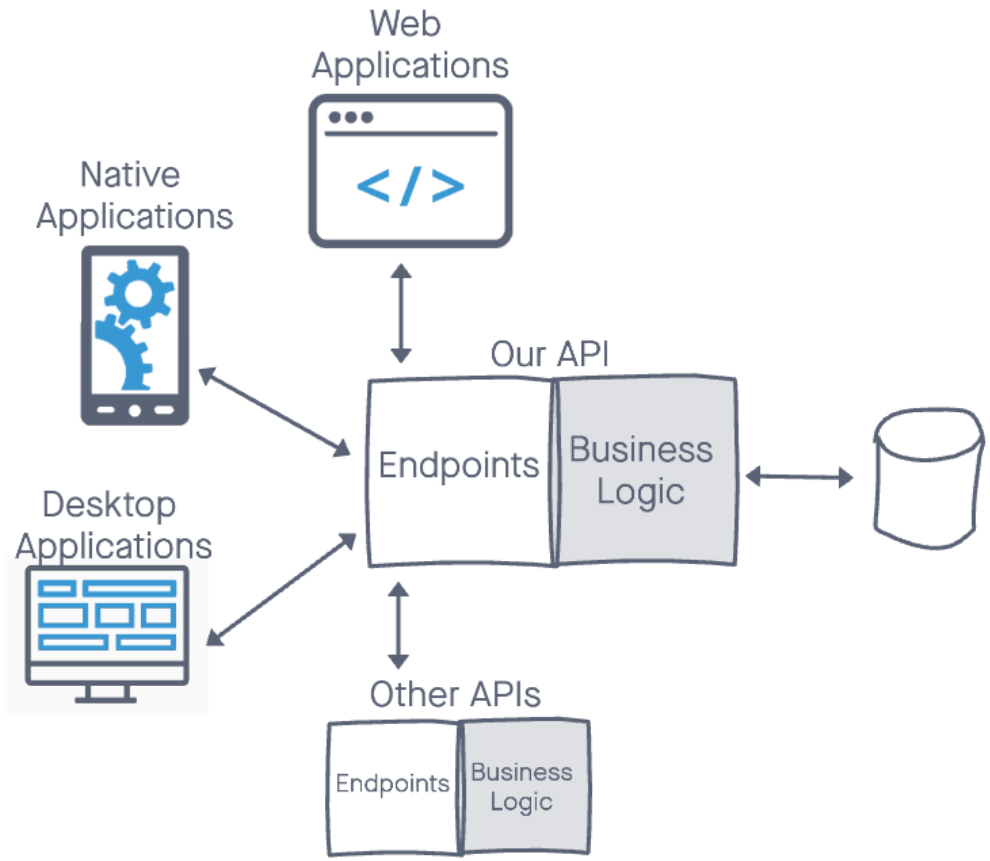
ARQUITECTURA DE UNA APLICACIÓN WEB (II)



RESPONSABILIDADES DEL BACKEND

- Desarrollo de API y Lógica de negocio
- Manejo de bases de datos
- Seguridad
- Integración de terceros
- Registro y Monitoreo
- Optimización de rendimiento
- Escalabilidad
- Gestión de servidores
- Documentación

RESTFUL APIS (I)

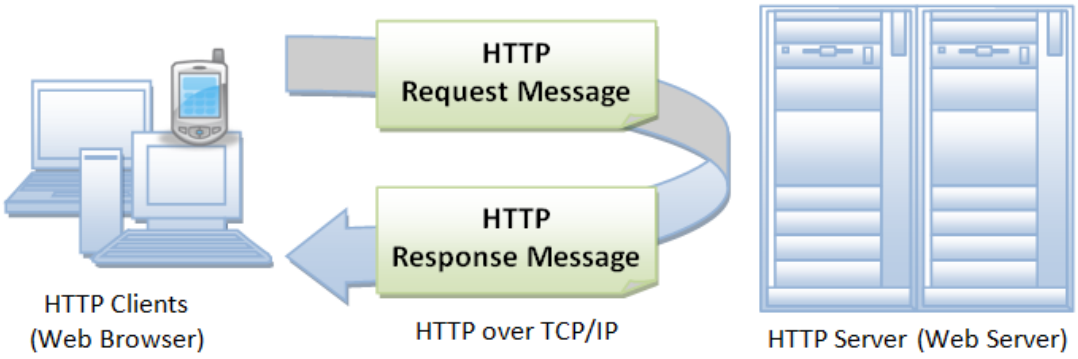


Workshop

RESTFUL APIS (II)

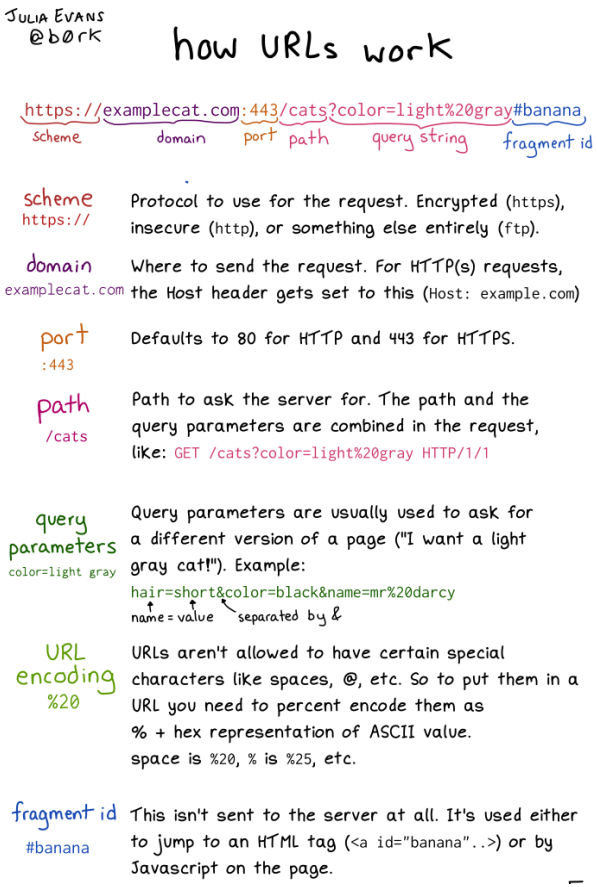
PROTOCOLO HTTP

- HTTP (Protocolo de Transferencia de Hipertexto).
- **WWW** se trata de la comunicación entre clientes y servidores web.
- La comunicación entre las computadoras cliente y los servidores web se realiza mediante el envío de **solicitudes HTTP** y la recepción de **respuestas HTTP**.



Workshop

PROTOCOLO HTTP: URLS



Julia Evans
@b0rk

how URLs work

https://examplecat.com:443/cats?color=light%20gray#banana
scheme domain port path query string fragment id

scheme (https://)
Protocol to use for the request. Encrypted (https), insecure (http), or something else entirely (ftp).

domain (examplecat.com)
Where to send the request. For HTTP(s) requests, the Host header gets set to this (Host: example.com)

port (:443)
Defaults to 80 for HTTP and 443 for HTTPS.

path (/cats)
Path to ask the server for. The path and the query parameters are combined in the request, like: GET /cats?color=light%20gray HTTP/1/1

query parameters (color=light gray)
Query parameters are usually used to ask for a different version of a page ("I want a light gray cat!"). Example:
hair=short&color=black&name=mr%20darcy
name = value, separated by &

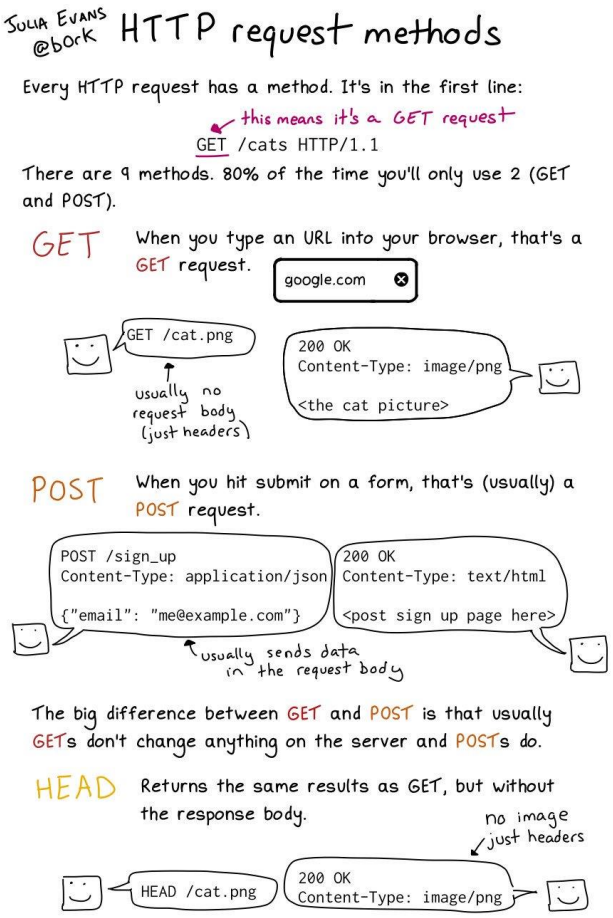
URL encoding (%20)
URLs aren't allowed to have certain special characters like spaces, @, etc. So to put them in a URL you need to percent encode them as % + hex representation of ASCII value.
space is %20, % is %25, etc.

fragment id (#banana)
This isn't sent to the server at all. It's used either to jump to an HTML tag (<a id="banana"..>) or by Javascript on the page.

Workshop

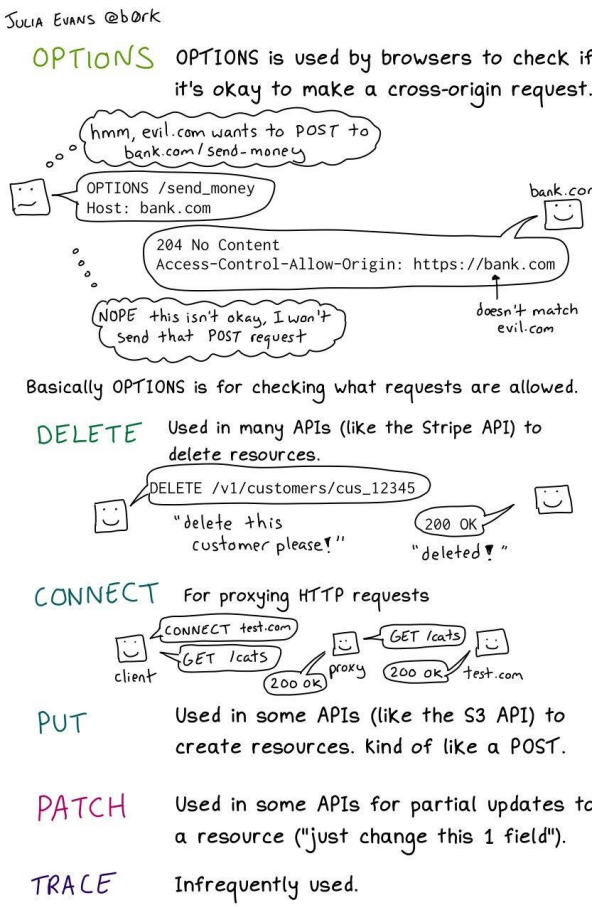
ANATONÍA DE UN HTTP REQUEST

MÉTODOS HTTP (I)



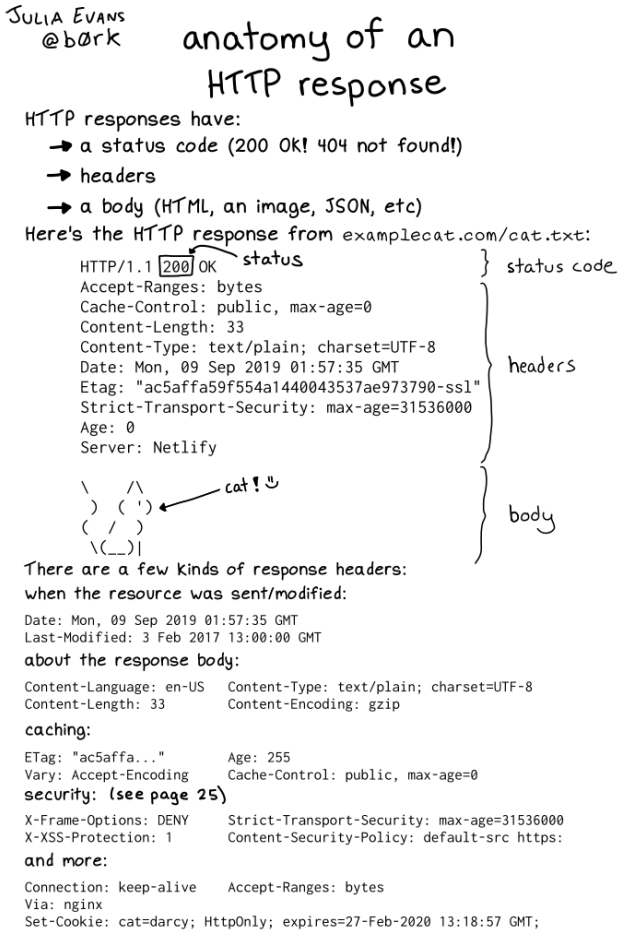
Clic aquí para ver más información sobre HTTP Request Methods

MÉTODOS HTTP (II)



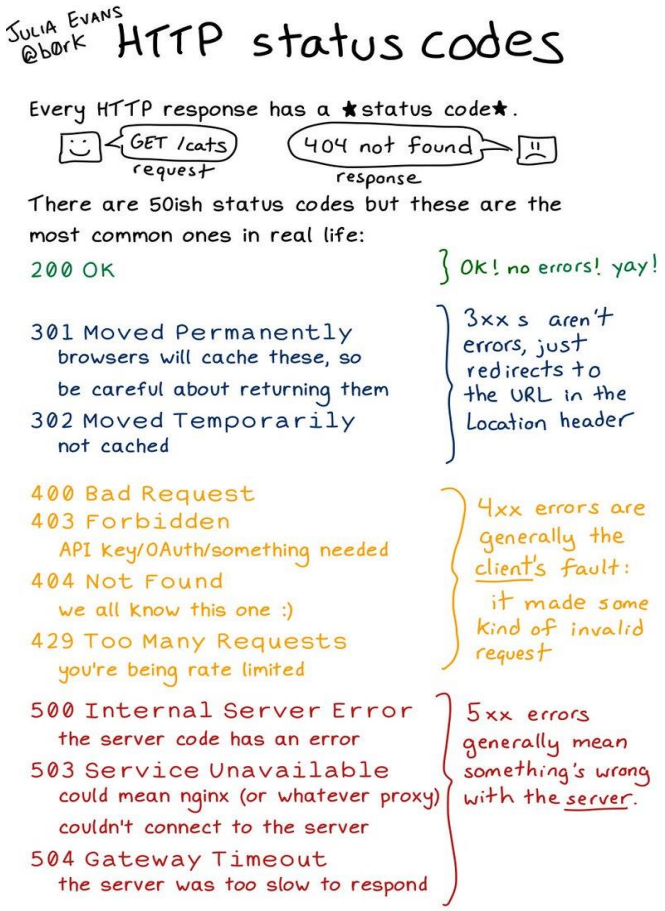
CABECERAS DE UN HTTP REQUEST

ANATOMÍA DE UN HTTP RESPONSE



CABECERAS DE HTTP RESPONSE

CÓDIGOS DE RESPUESTA HTTP



Workshop

DISEÑO DE RESTFUL APIS

REST: ROUTES DEFINITION

Method	EndPoint	Description
GET	/products	List of products
GET	/products/{id}	View a product
POST	/products	Create new product
PUT	/products/{id}	Update a product
DELETE	/products/{id}	Delete a product

REST: HTTP STATUS CODES

HTTP Status Codes	Informational
200	OK
201	Resource created
204	No content
400	Bad Request
401	Unauthorised
404	Not found
405	Method Not allowed
500	Internal Server Error

[Buenas prácticas para diseño de APIS]

FLASK

Flask es un ***framework web ligero*** escrito en Python. Fue creado en el 2004 por Armin Ronacher.

Su núcleo posee un diseño simple pero altamente extensible.

Permite iniciar el desarrollo de aplicaciones web de forma rápida y fácil, con la capacidad de escalar a aplicaciones complejas.

https://flask.palletsprojects.com

CARACTERÍSTICAS DE FLASK

- Ligero, simple y flexible.
- Extensible con una estructura personalizable.
- Puede manejar fácilmente aplicaciones grandes.
- Amigable para principiantes, su curva de aprendizaje es baja.
- Pitónico: es muy explícito, lo que aumenta la legibilidad.
- Brinda a los desarrolladores libertad para crear.
- Excelente documentación.
- Amplio ecosistema de extensiones.
- Soporte nativo para pruebas unitarias.

Desarrollo web, “una gota a la vez”

APLICACIÓN FLASK

Para crear una aplicación Flask, primero debemos instalarlo usado pip.

pip install flask

A continuación, podemos crear una aplicación:

```
from flask import Flask

app = Flask(__name__)

@app.route('/welcome/', methods=['GET'])
def welcome():
        return "Welcome to localhost:5050"

if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5050)
```

APLICACIÓN FLASK

Para crear una aplicación Flask, primero debemos instalarlo usado pip.

pip install flask

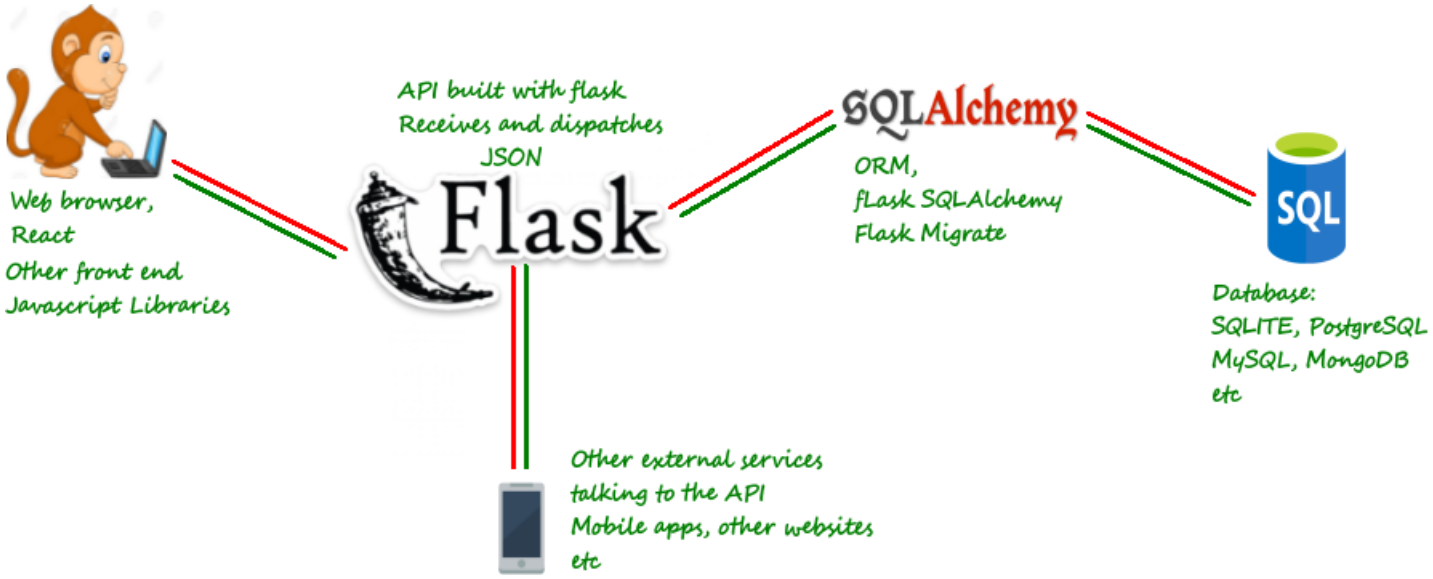
A continuación, podemos crear una aplicación:

```
from flask import Flask

app = Flask(__name__)

@app.route('/welcome/', methods=['GET'])
def welcome():
        return "Welcome to localhost:5050"

if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5050)
```

Workshop

DESARROLLO DE RESTFUL API CON FLASK

[Tutorial para crear una RestFul API con Flask]



POETRY

Poetry es una herramienta para la gestión y empaquetado de dependencias en Python. Permite declarar las bibliotecas de las que depende su proyecto y las administrará (instalará/actualizará) por usted.

Documentación: https://python-poetry.org/

Uso básico: https://python-poetry.org/docs/basic-usage/