Universidad de los Andes | Educación Continua
Vicerrectoría Académica

Introducción al Backend y Arquitectura Cliente-Servidor

Curso: Desarrollo Backend con Django

Protocolo HTTP y APIs REST

Desarrollo Backend con Django

Educación Continua Uniandes

Orígenes y Propósito de HTTP

HTTP (Hypertext Transfer Protocol) fue desarrollado por Tim Berners-Lee en 1989 como parte del proyecto World Wide Web (WWW) en el CERN.

Propósito principal:

- Permitir la transferencia de hipertexto entre sistemas distribuidos
- Establecer una comunicación estandarizada cliente-servidor
- Facilitar el acceso a recursos en la web mediante URLs

Actualmente, HTTP/2 (2015) y HTTP/3 (2022) han mejorado significativamente el rendimiento y seguridad del protocolo original.

Métodos HTTP: Cómo Funciona una Petición

1

GET

Solicita una representación de un recurso específico. Las peticiones GET solo deben recuperar datos y no tener otros efectos.

```
GET /api/usuarios/123 HTTP/1.1
```

2

POST

Envía datos para crear un nuevo recurso. Los datos se incluyen en el cuerpo de la petición.

```
POST /api/usuarios HTTP/1.1
```

3

PUT

Actualiza un recurso existente completo. Si el recurso no existe, puede crearlo.

```
PUT /api/usuarios/123 HTTP/1.1
```

4

DELETE

Elimina el recurso especificado.

```
DELETE /api/usuarios/123 HTTP/1.1
```

Otros métodos: PATCH (actualización parcial), HEAD (igual que GET pero sin cuerpo), OPTIONS (información sobre opciones de comunicación).

Métodos HTTP extendido: https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Methods

Anatomía de Peticiones y Respuestas HTTP

Petición HTTP

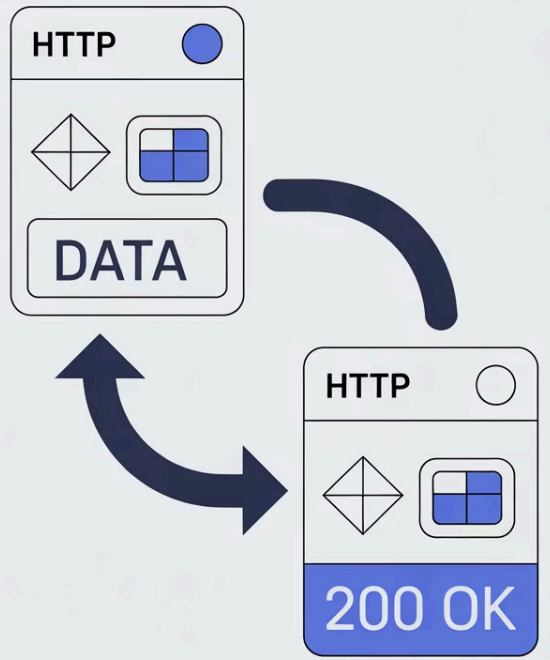
- **Línea de inicio:** Método, URI, versión
- **Headers:** Metadatos (Content-Type, Authorization)
- **Cuerpo:** Datos enviados (opcional)

Respuesta HTTP

- **Línea de estado:** Versión, código, mensaje
- **Headers:** Metadatos de respuesta
- **Cuerpo:** Contenido solicitado (opcional)

Códigos de estado HTTP: 1xx (Informativo), 2xx (Éxito), 3xx (Redirección), 4xx (Error del cliente), 5xx (Error del servidor)

Códigos de estado http extendido: https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Status



¿Qué es una API?

Una **API (Application Programming Interface)** es un conjunto de reglas y protocolos que permite que diferentes aplicaciones se comuniquen entre sí.

Relación con HTTP:

- HTTP proporciona el transporte para las APIs web
- Los verbos HTTP se mapean a operaciones CRUD
- La combinación método + URI identifica operaciones específicas sobre recursos

Las APIs modernas utilizan HTTP como capa de transporte para exponer funcionalidades y datos de forma controlada y estandarizada.

Principios Arquitectónicos: REST

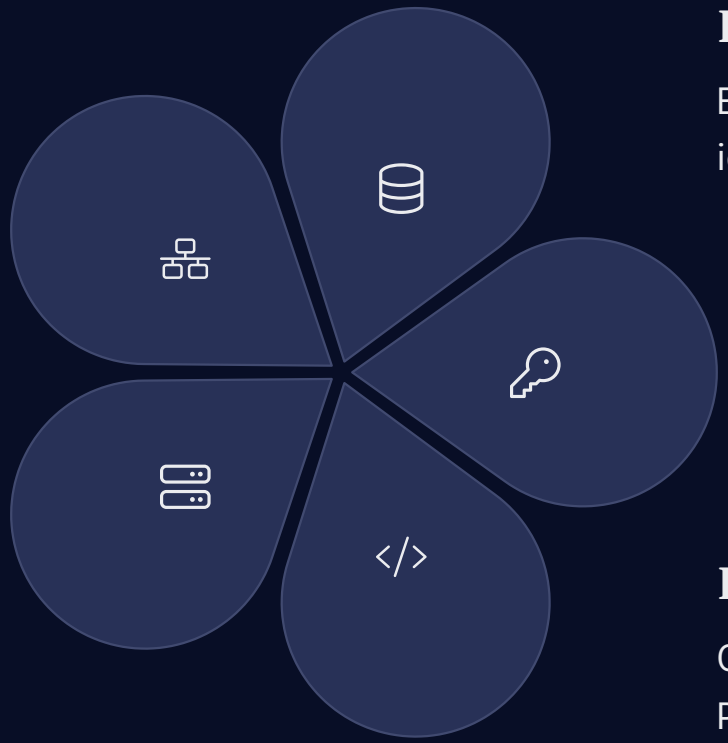
REST (Representational State Transfer) es un estilo arquitectónico para sistemas distribuidos creado por Roy Fielding en 2000.

Cliente-Servidor

Separación clara de responsabilidades entre cliente y servidor

Sistema en Capas

Componentes intermedios (cachés, balanceadores) transparentes al cliente



Recursos

Entidades o conceptos del dominio identificados mediante URIs

Sin Estado

Cada petición contiene toda la información necesaria para procesarla

Interfaz Uniforme

Operaciones estandarizadas (GET, POST, PUT, DELETE) sobre recursos

Buenas Prácticas en APIs REST

Nombrado de Recursos

- Usar sustantivos en plural para colecciones
- Evitar verbos en las URIs
- Ejemplo: `/users` en lugar de `/getUsers`

Uso Correcto de Métodos HTTP

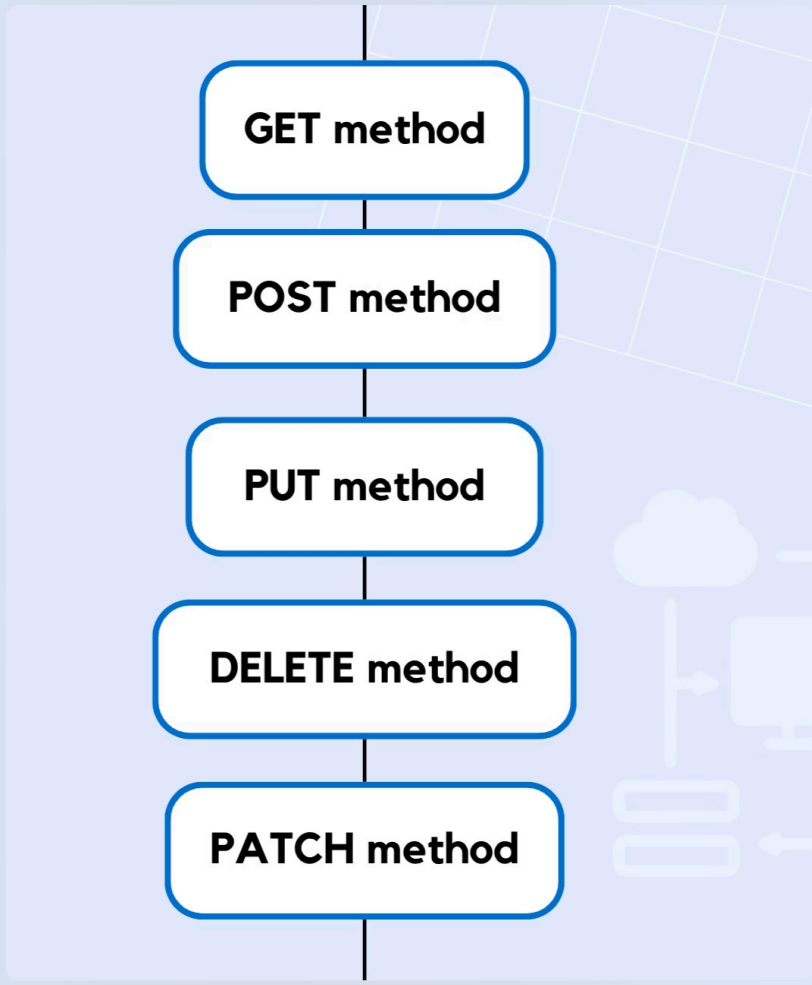
- GET para obtener
- POST para crear nuevos recursos
- PUT/PATCH para actualizar

Códigos de Estado Significativos

- 200 OK: Operación exitosa
- 201 Created: Recurso creado
- 404 Not Found: Recurso no existe

Versiones en la API

- Incluir número de versión en la URL
- Ejemplo: `/api/v1/usuarios`
- O usar encabezado Accept personalizado



Ejemplos de Peticiones y Respuestas REST

Petición GET

```
GET /api/v1/products/123 HTTP/1.1
Host: api.mitienda.com
Accept: application/json
Authorization: Bearer eyJhbGciOiJ9...
```

Respuesta

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 123,
  "nombre": "Laptop XPS",
  "precio": 1299.99,
  "stock": 15
}
```

Petición POST

```
POST /api/v1/orders HTTP/1.1
Host: api.mitienda.com
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJ9...

{
  "cliente_id": 456,
  "productos": [
    {"id": 123, "cantidad": 1},
    {"id": 789, "cantidad": 2}
  ]
}
```

Respuesta

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: /api/v1/orders/789

{
  "id": 789,
  "fecha": "2023-11-04T10:30:15Z",
  "estado": "pendiente"
}
```

Conceptos Clave a Recordar

HTTP es el protocolo fundamental de la web

Permite la comunicación entre clientes y servidores mediante un conjunto estandarizado de métodos y códigos de estado.

REST es un estilo arquitectónico, no un protocolo

Proporciona principios para diseñar APIs web escalables, mantenibles y desacopladas usando HTTP como transporte.

Las APIs RESTful se basan en recursos, no en acciones

Manipulan recursos mediante URIs y métodos HTTP estándar, siguiendo principios como stateless y cacheable.

La documentación y consistencia son fundamentales

Una API bien diseñada sigue patrones consistentes y está bien documentada para facilitar su uso y mantenimiento.

Preguntas para Reflexionar

¿Qué ventajas ofrece REST sobre otras arquitecturas?

Simplicidad, escalabilidad, independencia de plataforma, evolución independiente cliente-servidor

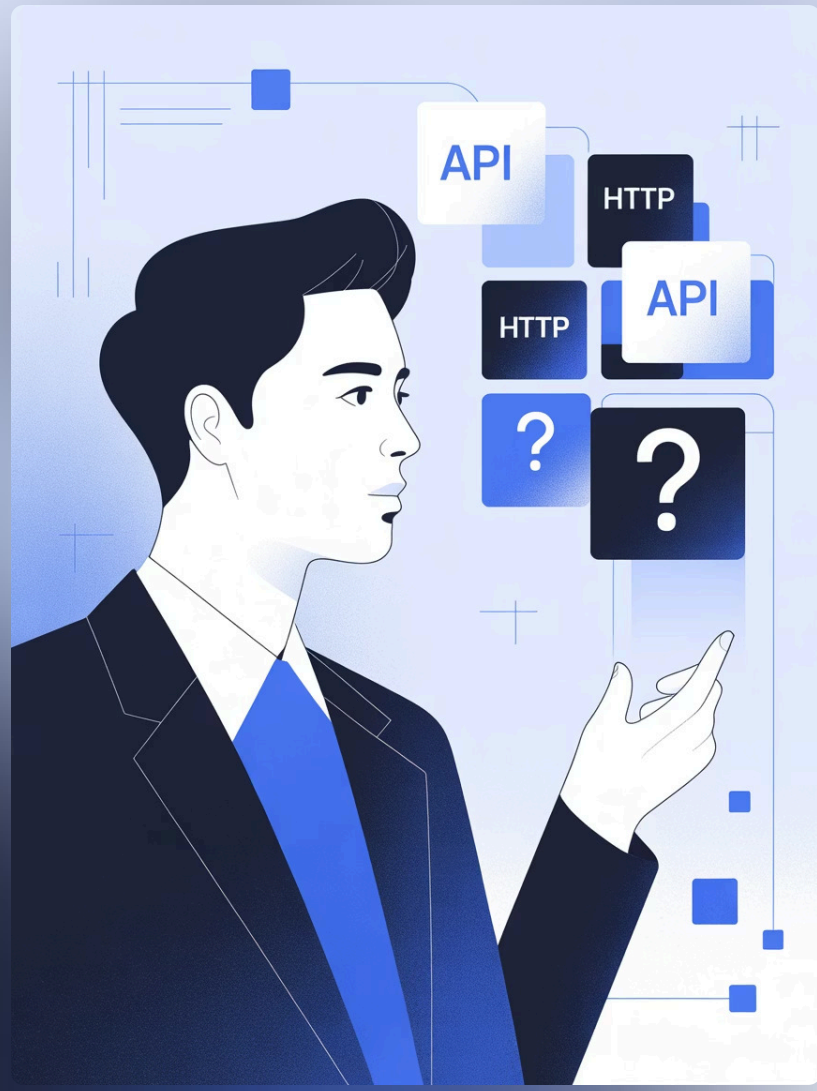
¿Cuál es la diferencia principal entre REST y SOAP?

REST es un estilo arquitectónico ligero basado en recursos, mientras que SOAP es un protocolo más pesado basado en acciones con más estructura formal

¿Cómo diseñarías una API RESTful para un sistema de reservas?

Recursos: /hoteles, /reservas, /clientes. Métodos: GET para consultar disponibilidad, POST para crear reservas, etc.

¡Gracias por su atención!



Introducción a Python para el Desarrollo Backend

Curso: Desarrollo Backend con Django

Educación Continua Uniandes

Agenda

Fundamentos de Python

Historia, filosofía y características principales

Python en el Backend

Ventajas y ecosistema para desarrollo de servidores

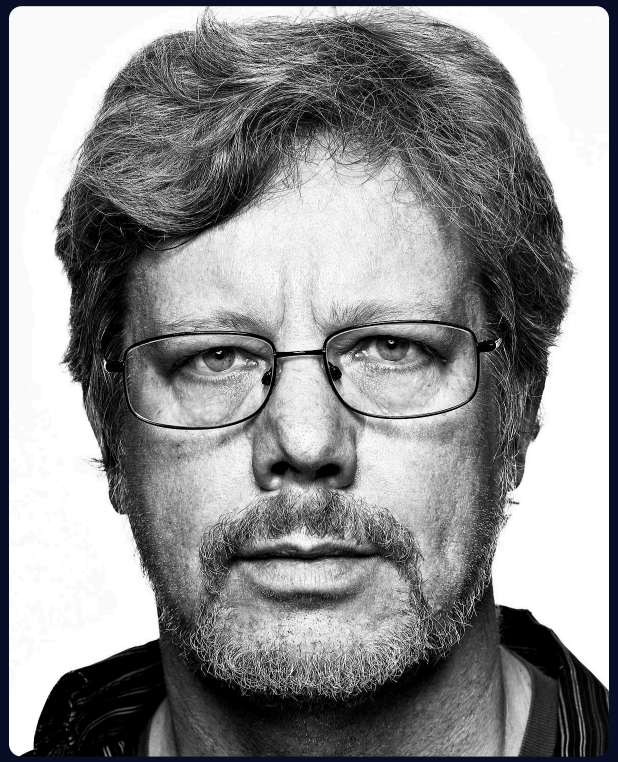
Configuración y Herramientas

Instalación, entornos virtuales y editores

Estructura y Buenas Prácticas

Sintaxis, convenciones y errores comunes

Historia y Filosofía de Python



Orígenes

- Creado por Guido van Rossum en 1991
- Nombre inspirado en "Monty Python"
- Diseñado para ser legible y expresivo

Filosofía "Zen de Python"

Escribe `import this` en un intérprete Python para ver los principios de diseño del lenguaje

Características Clave de Python

Simplicidad

Sintaxis clara y legible que prioriza la comprensión humana

Versatilidad

Aplicable en web, ciencia de datos, IA, automatización y más

Comunidad Activa

Extensa documentación, bibliotecas y foros de soporte

Tipado Dinámico

No requiere declarar tipos de variables explícitamente

¿Por qué Python para el Backend?

Ventajas en el Servidor

- Desarrollo rápido y productivo
- Excelente para prototipos y MVPs
- Potente manejo de datos y conexiones
- Integración sencilla con bases de datos
- Amplio soporte para APIs RESTful

Limitaciones a Considerar

- Rendimiento menor que lenguajes compilados
- Global Interpreter Lock (GIL) en CPython
- Mayor consumo de memoria

A pesar de estas limitaciones, Python es usado por Instagram, Spotify, Dropbox y otras plataformas a gran escala

Frameworks Python para Backend

Django

Framework "baterías incluidas" con ORM, admin y seguridad incorporados

Ideal para aplicaciones complejas y a gran escala

Flask

Microframework minimalista y flexible

Perfecto para APIs y servicios pequeños

FastAPI

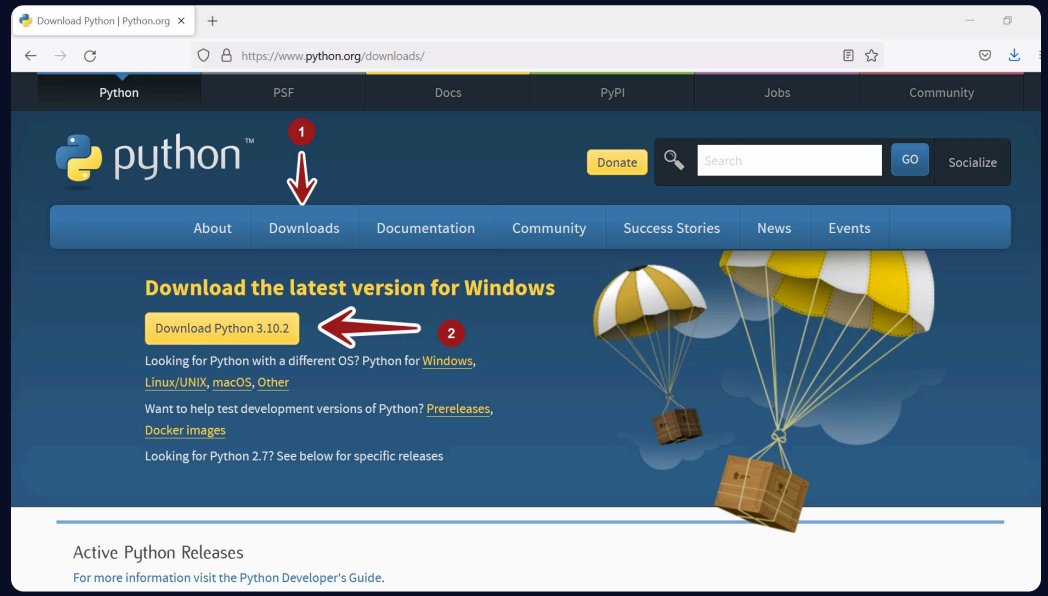
Framework moderno con alto rendimiento

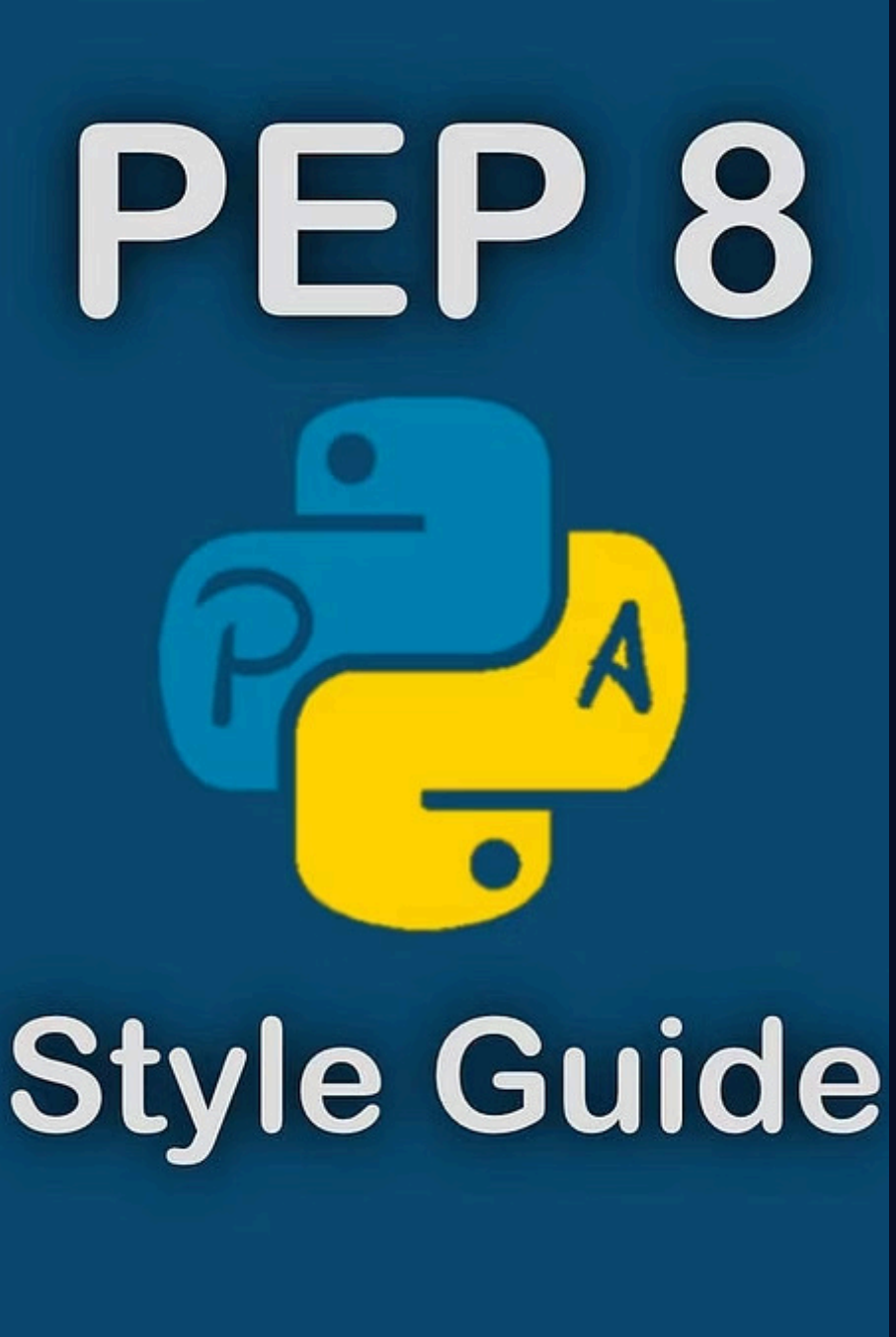
Validación automática y documentación OpenAPI

Instalación de Python

Instalación del Intérprete

1. Visita python.org/downloads
2. Descarga la última versión estable (3.13.+)
3. Durante la instalación, marca "Add Python to PATH"
4. Verifica con `python --version`





Convenciones de Estilo de código en python: PEP 8

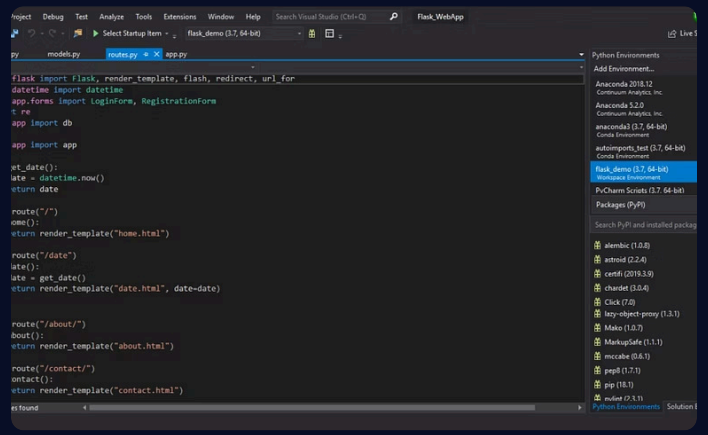
Reglas Principales

- Indentación: 4 espacios (no tabuladores)
- Líneas de máximo 79 caracteres
- Dos líneas en blanco entre funciones de nivel superior
- Una línea en blanco entre métodos de clase

Convenciones de Nombres

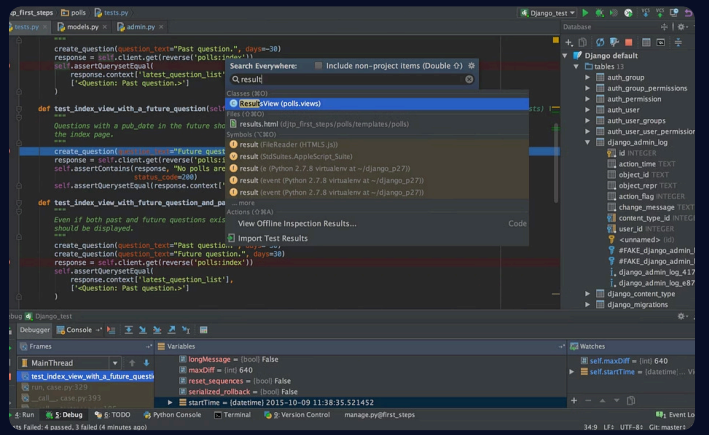
- `snake_case` para variables y funciones
- `PascalCase` para clases
- `MAYUSCULAS_CON_GUIONES` para constantes
- `_variable` (guión bajo) para privados

Herramientas de Desarrollo



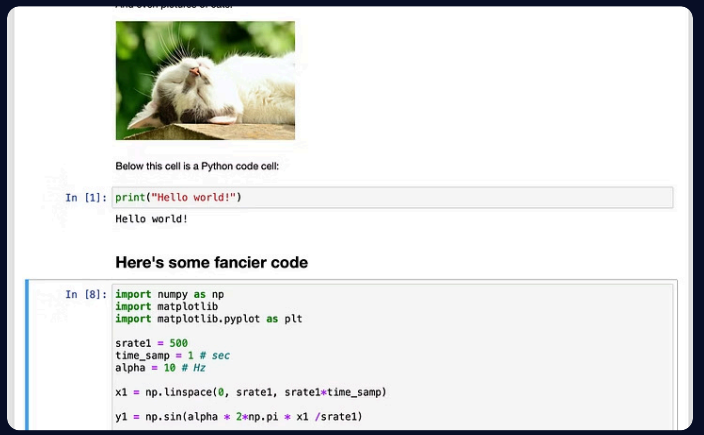
VS Code

Editor liviano con excelente soporte para Python mediante extensiones



PyCharm

IDE completo con análisis de código, depuración avanzada y refactorización



Jupyter Notebook

Ideal para prototipado, visualización y análisis exploratorio

Herramientas para la Calidad del Código

Extensiones y/o herramientas que podemos utilizar con python y vscode:

Linters y Formatters

- **pylint**: Verificador de estilo y errores
- **flake8**: Combinación de herramientas de verificación
- **black**: Formateador de código automático
- **isort**: Ordena automáticamente las importaciones

Testing

- **pytest**: Framework de pruebas moderno
- **unittest**: Módulo de pruebas estándar
- **coverage**: Mide cobertura de código en pruebas

Recursos de Aprendizaje

Documentación Oficial

- Python.org - Tutoriales y referencia
- Documentación de Django
- Python Package Index (PyPI)
- PEP 8 – Style Guide for Python Code

Resumen y Reflexión

1 **Python es un lenguaje versátil y productivo**

Su sintaxis clara y extensa biblioteca estándar lo hacen ideal para el backend

2 **Amplio ecosistema de frameworks y herramientas**

Django, Flask y FastAPI cubren diferentes necesidades y estilos de desarrollo

3 **Las buenas prácticas importan desde el principio**

Entornos virtuales, convenciones de estilo y arquitectura bien definida

Preguntas para reflexionar

- ¿Qué diferencias notas con otros lenguajes que has usado?
- ¿Para qué tipo de proyectos backend crees que Python es más adecuado?

Universidad de los Andes | Educación Continua
Vicerrectoría Académica

#EDCOUNIANDES

https://educacioncontinua.uniandes.edu.co/

Contacto: educacion.continua@uniandes.edu.co

Universidad de los Andes | Vigilada Mineducación. Reconocimiento como Universidad: Decreto 1297 del 30 de mayo de 1964. Reconocimiento personería jurídica: Resolución 28 del 23 de febrero de 1949 MinJusticia.