



Python: Conceptos de Aplicaciones en Producción

PROYECTOS EN
PRODUCCIÓN



Recordando

En Python encontramos una serie de herramientas para validar y asegurarnos que haya consistencia en el proyecto.

- Linters: Prevenir errores lógicos y patrones erróneos
- Formatters: Transformar código de manera automática para seguir un patrón
- Chequeo estático: Agregar consistencia en un contexto dinámico
- Verificación: Asegurarnos que exista un proceso de validación de la funcionalidad
- Distribución: Emitir la aplicación de acuerdo al contexto específico



En Python

En Python encontramos una serie de herramientas para validar y asegurarnos que haya consistencia en el proyecto.

- Linters: Prevenir errores lógicos y patrones erróneos *flake8*
- Formatters: Transformar código de manera automática para seguir un patrón *black, ruff*
- Chequeo estático: Agregar consistencia en un contexto dinámico *mypy*
- Verificación: Asegurarnos que exista un proceso de validación de la funcionalidad *pytest*
- Distribución *contenedores en la nube, pypi, encapsulado ejecutable*



Distribución

Dependiendo de la aplicación o herramienta en construcción existen muchas formas de distribuirla

Cubriremos brevemente una serie de ejemplos.



Librerías

Un proyecto de Python enfocado enteramente en ofrecer utilidades a otro proyecto.

Por ejemplo, una librería para procesar datos, o una herramienta utilitaria para transformar imágenes.

Generalmente se publican al repositorio de Python

[PyPI · The Python Package Index](https://pypi.org/)



Librerías: setup.py

La configuración principal que define lo necesario para poder distribuir el código como una librería.

```
# setup.py
from setuptools import setup, find_packages
```

```
setup(
    name="mibiblioteca",
    version="0.1",
    packages=find_packages(),
    author="Tu Nombre",
    author_email="tuemail@example.com",
    description="Una biblioteca de ejemplo",
    long_description=open('README.md').read(),
    long_description_content_type='text/markdown',
    url="https://github.com/tuusuario/mibiblioteca",
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    python_requires='>=3.6',
)
```

→ Versión actual

→ Define metadatos específicos

→ Versión mínima de Python



Librerías: Publicando

Muchos flujos para publicar, por ejemplo usando *twine*

```
pip install setuptools wheel twine  
python setup.py sdist bdist_wheel  
twine upload dist/*
```

```
pip install mibiblioteca
```



Librerías: Licencias

Las licencias son acuerdos legales que permiten a una persona o entidad usar, distribuir o modificar un producto, software, o contenido bajo ciertas condiciones.

Licencia MIT, Apache 2.0, GPL, etc.

1. Incluir un archivo LICENE
2. Incluir referencia en el setup.py

```
from setuptools import setup, find_packages

setup(
    ...
    url="https://github.com/tuusuario/mibiblioteca",
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    python_requires='>=3.6',
    license="MIT",
)
```




Aplicaciones web

Al ser parte de proyectos más complejos. Una aplicación web requerirá diferentes funcionalidades dependiendo del destino de la aplicación web.

Como mínimo necesita un servidor, generalmente a través de una infraestructura como servicio o una plataforma como servicio (IaaS, PaaS).



Aplicaciones web: IaaS

IaaS proporciona infraestructura informática virtualizada a través de Internet. Esto incluye servidores, almacenamiento, y redes. Los usuarios pueden alquilar estos recursos según sea necesario y escalar hacia arriba o hacia abajo según sus necesidades.

Mucho control, pero muchas más responsabilidades.

Ejemplo: Amazon Web Services, Google CP, Azure.



Aplicaciones web: PaaS

PaaS proporciona una plataforma completa para desarrollar, ejecutar y gestionar aplicaciones sin preocuparse por la infraestructura subyacente. Incluye herramientas de desarrollo, sistemas operativos, bases de datos y servidores.

Mucho más sencillas, pero generalmente mucho más costosas.

Ejemplo: **PythonAnywhere**, Heroku

Aplicaciones web: Servidor simple



Sin embargo, como aprendimos en la clase anterior, un servidor no es más que un programa corriendo en computador. Mientras este sistema esté expuesto a la red de interés, cualquier equipo puede ser usado.

Mucho más sencillo, pero potencialmente mucha menos fiabilidad.



Aplicaciones de escritorio

A pesar de que Python nada más requiere utilizar el intérprete y las librerías dependientes, en muchos contextos es necesario incluir un paquete que incluya todas las partes requeridas para ejecutar nuestra aplicación.

Esto es conceptualmente similar a empaquetar todo el entorno de desarrollo de Python, incluyendo el intérprete, junto a los módulos de nuestra aplicación.



Aplicaciones de escritorio

Empaquetando en Windows

PyInstaller

```
pip install pyinstaller
pyinstaller --onefile mi_script.py
cd dist
./mi_script.exe
```

[PyInstaller Manual — PyInstaller 6.10.0 documentation](#)



Empaquetando en general

Sin embargo, no existe una forma estandarizada, ¡ya que todo es contextual!

Por ejemplo, los contenedores aparecen mucho en el contexto de orquestado y distribución en IaaS usando herramientas como **Docker Swarm** o **Kubernetes**.

Un mundo completamente diferente y que se recomienda cubrir si estás interesado en funcionalidades de infraestructura escalable en la nube.

[Kubernetes](#)

CI/CD



Integración y emisión continua

Son prácticas que ayudan a los equipos de desarrollo de software a entregar cambios de código de manera más frecuente y confiable.



CI/CD: Integración continua

- Automatiza la integración de cambios de código en un repositorio compartido.
- Cada cambio se verifica mediante pruebas automáticas para asegurar que no cause errores en la aplicación.



CI/CD: Emisión continua

- Automatiza la entrega de cambios de código a un entorno de producción.
- La **Emisión Continua** asegura que el código esté siempre listo para ser desplegado, mientras que el **Despliegue Continuo** automatiza el despliegue en producción.



CI/CD: Emisión continua

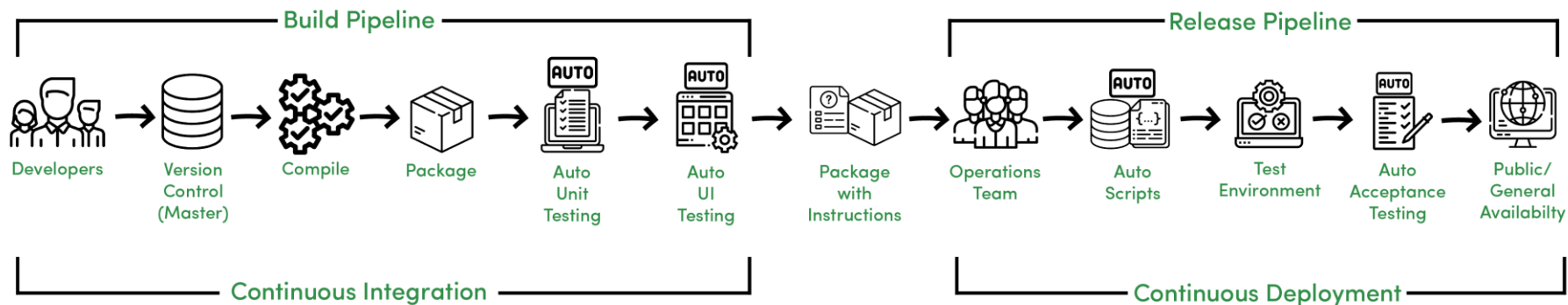
NO EXISTEN 2 FORMAS IGUALES DE UN FLUJO CI/CD.

La realidad es que todo depende del proyecto, la infraestructura disponible, y el tipo de aplicación.



CI/CD: Emisión continua

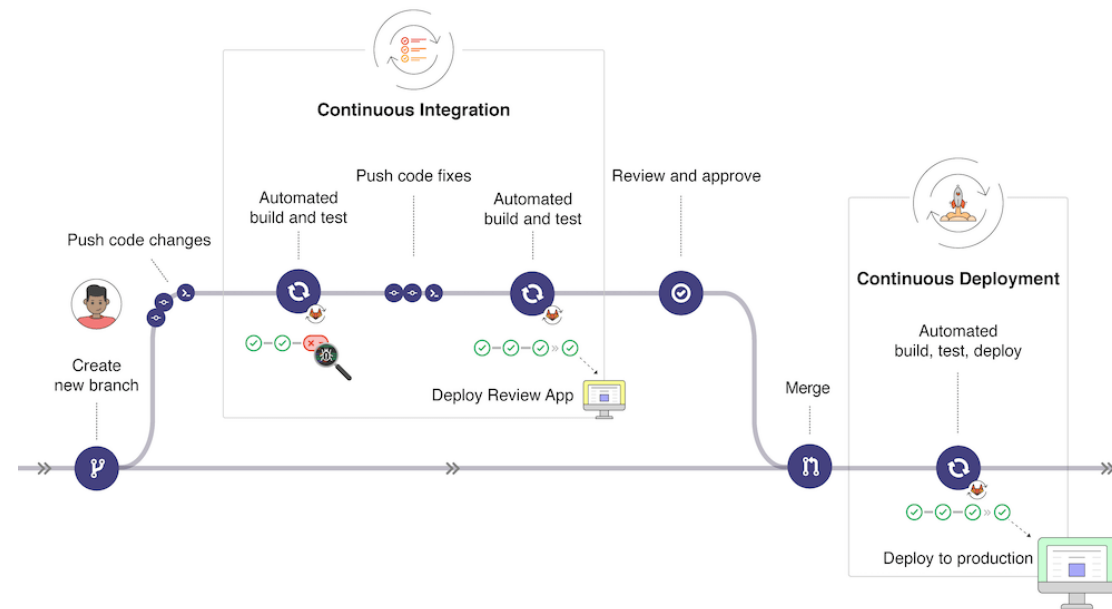
La realidad es que todo depende del proyecto, la infraestructura disponible, y el tipo de aplicación.





CI/CD: Emisión continua

La realidad es que todo depende del proyecto, la infraestructura disponible, y el tipo de aplicación.



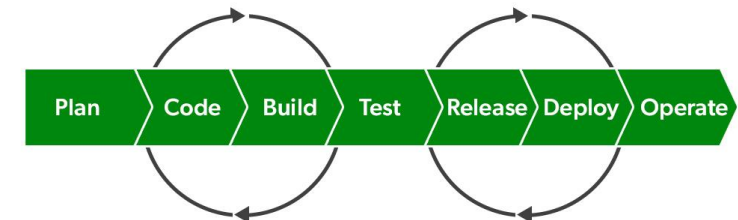


CI/CD: Github Actions

Mediante Github Actions se puede automatizar todo el ciclo de integración y emisión de una aplicación que vive en Github.

Action: Un proceso parte de la tubería de integración o emisión

[GitHub Actions documentation - GitHub Docs](#)





CI/CD: CircleCI

CircleCI es una plataforma de integración continua y entrega continua (CI/CD) que automatiza la construcción, prueba y despliegue de aplicaciones.

Funciona a través de un archivo de configuración en el repositorio de Github.

[Configure a Python application on CircleCI - CircleCI](#)



CI/CD: CircleCI

- **version:** Especifica la versión de configuración de CircleCI.
- **orbs:** El sistema específico preconfigurado.
- **workflows:** Define el flujo de trabajo que ejecuta los trabajos en el orden especificado.

[Configure a Python application on CircleCI - CircleCI](#)



CI/CD: CircleCI

```
version: 2.1

orbs:
  python: circleci/python@1.2.3

jobs:
  lint:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
      - run:
          name: Dependencias
          command: |
            python -m venv venv
            . venv/bin/activate
            pip install black mypy
      - run:
          name: Black
          command: black --check .
      - run:
          name: Mypy
          command: mypy .

  test:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
      - run:
          name: Dependencias
          command: |
            python -m venv venv
            . venv/bin/activate
            pip install pytest
      - run:
          name: Tests
          command: pytest

  deploy:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
      - run:
          name: Dependencias
          command: |
            python -m venv venv
            . venv/bin/activate
            pip install twine
      - run:
          name: Armar
          command: |
            . venv/bin/activate
            python setup.py sdist
            bdist_wheel
      - run:
          name: Subir a PyPI
          command: |
            . venv/bin/activate
            twine upload dist/*

workflows:
  version: 2
  build-and-deploy:
    jobs:
      - lint
      - test
      - deploy:
          requires:
            - lint
            - test
```

[Configure a Python application on CircleCI - CircleCI](#)



Python: Conceptos de Aplicaciones en Producción

HERRAMIENTAS DE
PRODUCTIVIDAD



Modelos ML reusables

En el contexto de desarrollo de aplicaciones que utilizan Machine Learning, es muchas veces necesario utilizar modelos existentes y ajustarlos a los requisitos de nuestro proyecto.

En general, el proceso de producir modelos es un arte y ciencia muy a parte de la producción de una aplicación.

Hugging Face



Una plataforma de inteligencia artificial y aprendizaje automático que permite a la comunidad **colaborar en modelos, conjuntos de datos y aplicaciones**. Es conocida por **su biblioteca de transformadores y modelos reutilizables**.

[Hugging Face – The AI community building the future.](https://huggingface.co)

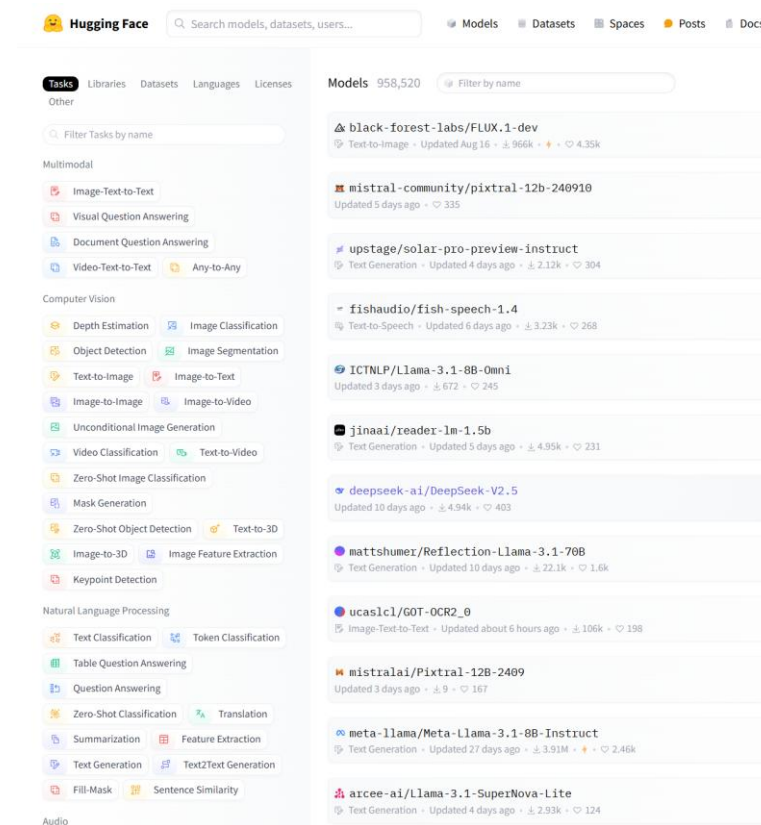
Hugging Face



Repositorio de miles de modelos y sets de datos.

Con licencias específicas e instrucciones de integración.

¡En general, las empresas contribuyen
sus modelos open-source!



Hugging Face



Downloads last month
7,776,462

Safetensors ① Model size 137M params Tensor type F32 ↗

⚡ Inference API ⓘ Warm ▾

📄 Text Generation Examples ▾

My name is Clara and I am a woman. I am a woman who is a woman.
I am a woman who is a woman

Compute ctrl+Enter 0.4

Computation time: cached

</> View Code Maximize

```
pip install transformers
pip install datasets tokenizers
pip install transformers torch
```

```
from transformers import pipeline

# Crear un pipeline de generación de texto usando GPT-2
generador = pipeline('text-generation', model='gpt-2')

# Generar texto
resultado = generador("Érase una vez, en un reino lejano,", max_length=50,
num_return_sequences=1)

# Imprimir el texto generado
print(resultado[0]['generated_text'])
```

ChatGPT para desarrolladores

Una herramienta que utiliza el modelo basado en la arquitectura GPT (Generative Pre-trained Transformer). Utiliza técnicas avanzadas de aprendizaje profundo para generar texto coherente y relevante en respuesta a las entradas del usuario.

ChatGPT se entrena en **grandes cantidades** de datos textuales y puede realizar tareas como responder preguntas, mantener conversaciones, y generar contenido.

Grandes cantidades

¡El modelo en sí surgió a la fama por el producto y su integración, ya que variaciones similares del modelo existen en la academia desde hace bastantes años!

Entrenado en un corpus de 500 mil millones de tokens de texto extraídos de millones de fuentes.

ChatGPT

Es por eso que se ha vuelto una herramienta muy útil en varios contextos. ¡Incluida la programación!

Sin embargo, debido al desconocimiento sobre cómo funciona esta herramienta, ¡es recomendado seguir unos consejos para maximizar sus beneficios!

ChatGPT: Lo que es

Un modelo de lenguaje, es decir produce palabras basado en un modelo matemático optimizado para su caso de uso.

Es excelente cuando se necesita coherencia general al generar contenido basado en patrones de texto.

ChatGPT: Lo que NO es

Una base de datos: ChatGPT no busca información en una base de datos, la genera a partir de su modelo matemático.

Una inteligencia autónoma: A pesar de parecer muy coherente, es una cadena de palabras que siguen un modelo estadístico.

¡Por lo tanto, puede cometer errores (y lo hace muy seguido)!

ChatGPT: Errores



How many countries begin with the letter "Y"?



As of my last knowledge update in January 2022, there are three countries whose names start with the letter "Y." These countries are:



1. Yemen
2. Zambia
3. Zimbabwe

Please note that geopolitical changes can occur, and the number of countries in the world can change over time, so it's a good practice to verify this information with up-to-date sources for the most current and accurate count.

ChatGPT: Alucinaciones

Los modelos de lenguaje se entrenan con grandes cantidades de datos textuales, pero ChatGPT no tiene acceso a información en tiempo real ni a datos fuera de su conjunto de entrenamiento.

Ejemplo: Si el modelo no ha sido entrenado con información actualizada sobre un evento reciente, puede generar respuestas incorrectas sobre ese evento.

ChatGPT: Alucinaciones

Los modelos pueden malinterpretar el contexto de una pregunta o solicitud, especialmente si es ambigua o contradictoria.

Ejemplo: Preguntar “¿Dónde puedo patinar sobre hielo en el Cotopaxi?” puede llevar al modelo a generar una respuesta seria, aunque patinar sobre hielo en el Cotopaxi no sea posible.

ChatGPT: Alucinaciones

Esto es un problema más grande donde interviene la lógica, ya que el modelo no está razonando, sino generando texto basado en patrones.

Pregunta: “¿Cuál es la raíz cuadrada de 16?”

Respuesta Alucinada: “La raíz cuadrada de 16 es 5.”

Explicación: La respuesta correcta es 4. El modelo puede generar un número incorrecto debido a errores en la predicción de la secuencia de números.

ChatGPT: Alucinaciones

Esto es un problema más grande donde interviene la lógica, ya que el modelo no está razonando, sino generando texto basado en patrones.

Pregunta: “¿Cuál es el teorema de Fermat?”

Respuesta Alucinada: “El teorema de Fermat establece que la suma de dos números primos siempre es un número primo.”

Explicación: El teorema de Fermat en realidad se refiere a la imposibilidad de dividir un cubo en dos cubos, una cuarta potencia en dos cuartas potencias, y así sucesivamente, para cualquier número mayor que dos.

ChatGPT: Alucinaciones

Pregunta: “¿Cómo se usa la librería `requests` para hacer una solicitud GET en Python?”

Respuesta Alucinada:

```
import requests

response = requests.get('https://api.example.com/data')
data = response.json()
print(data['nombre'])
```

Aunque el código parece correcto, si la API no devuelve un JSON o la clave **nombre** no existe, el código fallará. El modelo puede no tener en cuenta estos posibles errores.

ChatGPT: Alucinaciones

La realidad es que en general, a pesar de hacer un buen trabajo, a medida que incrementa la complejidad de una solicitud, incrementa la probabilidad de alucinaciones y errores lógicos.

Lo más importante: **Es imposible de comprobar rápidamente a menos que entendamos el tema en detalle.**

ChatGPT: Alucinaciones

Siempre revisa y prueba el código generado por modelos de lenguaje.

ChatGPT: Alucinaciones

Verifica los cálculos matemáticos y las afirmaciones en fuentes confiables.

ChatGPT: Alucinaciones

Utiliza linters y herramientas de análisis estático para verificar el código.

ChatGPT: Alucinaciones

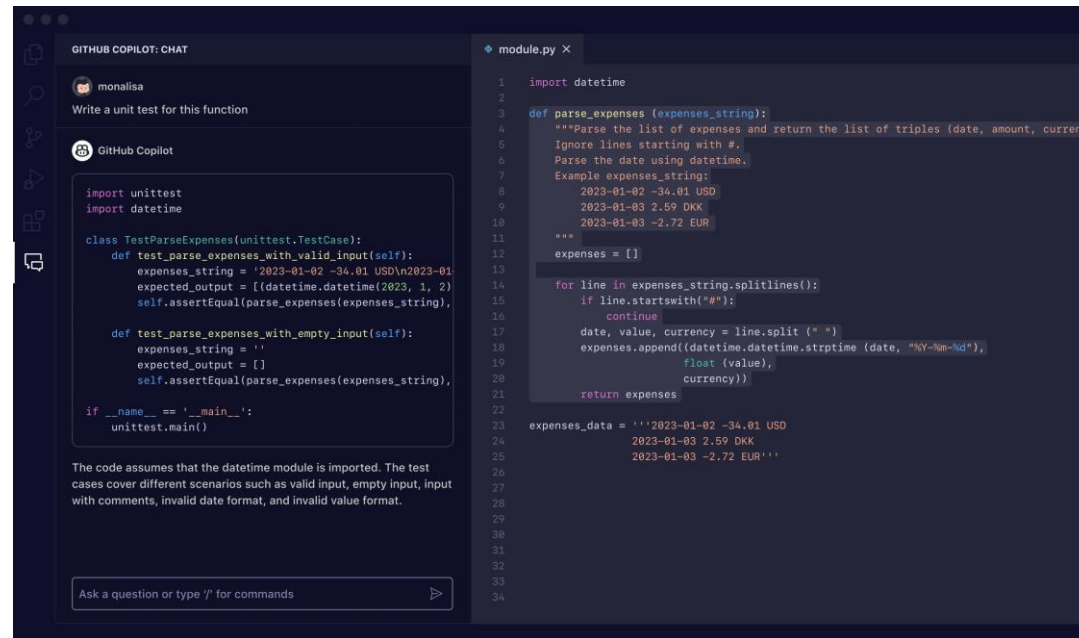
Sé específico en tus preguntas para reducir la ambigüedad, proporciona contexto adicional si es necesario.

ChatGPT: Alucinaciones

Usar ChatGPT es muy recomendado para plantillas y andamios de proyectos, una vez que tengamos una lista de requisitos, es decir cuando podemos operar con conocimiento y no incertidumbre.

Github CoPilot

Integración de ChatGPT para usos de programación. Incluido gratuitamente con los beneficios educación. Recomendación de tests, explicación e interpretación, **¡documentación de funciones!**



The screenshot displays the GitHub Copilot interface within a code editor. On the left, a chat window titled 'GITHUB COPILOT: CHAT' shows a conversation with a user named 'monalisa'. The user has asked 'Write a unit test for this function'. The Copilot has responded with a Python unit test using unittest and datetime modules. The test includes two cases: one with valid input and one with empty input. Below the code, a note states: 'The code assumes that the datetime module is imported. The test cases cover different scenarios such as valid input, empty input, input with comments, invalid date format, and invalid value format.' At the bottom of the chat window is a prompt: 'Ask a question or type "/" for commands'. On the right, the code editor shows a file named 'module.py'. It contains a function 'parse_expenses' that takes a string of expense entries and returns a list of tuples. The function ignores lines starting with '#', parses the date using datetime, and splits the line into date, value, and currency. An example input string is provided: '2023-01-02 -34.01 USD\n2023-01-03 2.59 DKK\n2023-01-03 -2.72 EUR'. The function also includes a test case for the 'parse_expenses' function.

```
import unittest
import datetime

class TestParseExpenses(unittest.TestCase):
    def test_parse_expenses_with_valid_input(self):
        expenses_string = '2023-01-02 -34.01 USD\n2023-01-03 2.59 DKK\n2023-01-03 -2.72 EUR'
        expected_output = [(datetime.datetime(2023, 1, 2), -34.01, 'USD'), (datetime.datetime(2023, 1, 3), 2.59, 'DKK'), (datetime.datetime(2023, 1, 3), -2.72, 'EUR')]
        self.assertEqual(parse_expenses(expenses_string), expected_output)

    def test_parse_expenses_with_empty_input(self):
        expenses_string = ''
        expected_output = []
        self.assertEqual(parse_expenses(expenses_string), expected_output)

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

The code assumes that the datetime module is imported. The test cases cover different scenarios such as valid input, empty input, input with comments, invalid date format, and invalid value format.

```
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and return the list of triples (date, amount, currency)
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
    2023-01-02 -34.01 USD
    2023-01-03 2.59 DKK
    2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                        float(value),
                        currency))

    return expenses

expenses_data = '''2023-01-02 -34.01 USD
2023-01-03 2.59 DKK
2023-01-03 -2.72 EUR'''
```


Ingeniería de prompts

Un prompt es una instrucción que se le provee a una ingeniería artificial. ¡En este caso existe una técnica que se enfoca en maximizar la calidad de las respuestas obtenidas por un sistema!

Ingeniería de prompts implica crear y refinar las entradas para que los modelos de IA comprendan mejor las tareas y produzcan resultados más exactos.

[Prompt Engineering \(learnprompting.org\)](https://learnprompting.org)