Here's a markdown file explaining the use of Typer in Python, with code comments and explanations in both English and Egyptian Arabic.

```
# Using Typer in Python

Typer is a library for building command-line interface (CLI) applications. It is designed to be easy to use and to help developers create user-friendly command-line tools. Below are examples of how to use Typer in Python with comments and explanations.

## Installation
To install Typer, use the following command:
```

pip install Typer

```

## Example 1: Basic Usage
This example demonstrates a basic Typer application that sums two numbers.

### Code:
```python
# Import the typer library
import typer

# Define a function that takes two integers and prints their sum
def sum(x: int, y: int):
    print(x + y)

# Check if the script is run directly (not imported)
if __name__ == '__main__':
    # Run the sum function using Typer
    typer.run(sum)
```

## Explanation:

- `import typer`: Import the Typer library.
- `def sum(x: int, y: int)`: Define a function `sum` that takes two integer arguments `x` and `y`.
- `print(x + y)`: Print the sum of `x` and `y`.
- `if __name__ == '__main__':`: Check if the script is run directly.
- `typer.run(sum)`: Run the `sum` function using Typer.

# Example 2: Multiple Commands

This example shows how to create a Typer application with multiple commands.
```

### Code:

```
# Import the typer library
import typer

# Create a Typer app instance
app = typer.Typer()

# Define a command for summing two numbers
@app.command()
def sum(x: int, y: int):  # type hint
    print(x + y)

# Define a command for multiplying two numbers
@app.command()
def mul(x: int, y: int):
    print(x * y)

# Check if the script is run directly (not imported)
if __name__ == '__main__':
    # Run the Typer app
    app()
```

### Explanation:

- `app = typer.Typer()`: Create an instance of Typer.
- `@app.command()`: Decorator to register the `sum` and `mul` functions as commands.
- `def sum(x: int, y: int)`: Define a function `sum` that takes two integer arguments `x` and `y`.
- `print(x + y)`: Print the sum of `x` and `y`.
- `def mul(x: int, y: int)`: Define a function `mul` that takes two integer arguments `x` and `y`.
- `print(x * y)`: Print the product of `x` and `y`.
- `if __name__ == '__main__':`: Check if the script is run directly.
- `app()`: Run the Typer app.

## Example 3: Registering Commands Later

This example demonstrates how to register commands after defining them.

### Code:

```
# Import the typer library
import typer

# Define functions for sum and multiplication
def sum(x: int, y: int):  # type hint
    print(x + y)

def mul(x: int, y: int):
```

```
    print(x * y)

# Check if the script is run directly (not imported)
if __name__ == '__main__':
    # Create an instance of Typer
    app = typer.Typer()
    # Register the sum and mul functions as commands
    app.command()(sum)
    app.command()(mul)
    # Run the Typer app
    app()
```

### Explanation:

- `app.command()(sum)`: Register the `sum` function as a command.
- `app.command()(mul)`: Register the `mul` function as a command.

## Example 4: Command with Options

This example demonstrates a Typer command that includes an option.

### Code:

```
# Import the typer library
import typer

# Create a Typer app instance
app = typer.Typer()

# Define a command with an option
@app.command()
def calc(salary: int, employee: bool = False):
    if employee:
        tax = salary * 0.85
        print(tax)
        return

    print(salary * 0.90)

# Check if the script is run directly (not imported)
if __name__ == "__main__":
    # Run the Typer app
    app()
```

### Explanation:

- `def calc(salary: int, employee: bool = False)`: Define a function `calc` with an integer `salary` and a boolean `employee` option.
- `if employee: tax = salary * 0.85`: Calculate tax for employees.
- `print(tax)`: Print the tax for employees.
- `print(salary * 0.90)`: Print the tax for non-employees.

- `if __name__ == "__main__": app()` : Check if the script is run directly and run the Typer app.

## Notes:

To run the command with different options:

1. For non-employees: `python script.py calc 100`
2. For employees: `python script.py calc 100 --employee`

## بالعربي المصري

Typer في بايثون. هي مصممة لتكون سهلة الاستخدام وتساعد المطورين في إنشاء أدوات سطر أوامر (CLI) مكتبة لإنشاء تطبيقات سطر الأوامر سهلة الاستخدام. تحت بعض الأمثلة على كيفية استخدام Typer في بايثون مع التعليقات والتوضيحات.

## التثبيت

لتثبيت Typer، استخدم الأمر التالي:

```
pip install Typer
```

## مثال 1: الاستخدام الأساسي

هذا المثال يوضح تطبيق Typer بسيط يجمع رقمين.

### الكود:

```
# استيراد مكتبة Typer
import typer

# تعريف دالة تأخذ رقمين صحيحين وتطبع مجموعهم
def sum(x: int, y: int):
    print(x + y)

# التحقق من تشغيل السكريبت مباشرة (وليس استيراده)
if __name__ == '__main__':
    # تشغيل دالة sum باستخدام Typer
    typer.run(sum)
```

### التوضيح:

- `import typer` : استيراد مكتبة Typer.
- `def sum(x: int, y: int)` : تعريف دالة `sum` التي تأخذ رقمين صحيحين `x` و `y`.
- `print(x + y)` : طباعة مجموع `x` و `y`.
- `if __name__ == '__main__':` : التحقق من تشغيل السكريبت مباشرة.
- `typer.run(sum)` : تشغيل دالة `sum` باستخدام Typer.

## مثال 2: أوامر متعددة

هذا المثال يوضح كيفية إنشاء تطبيق Typer يحتوي على أوامر متعددة.

### الكود:

```
# استيراد مكتبة Typer
import typer

# إنشاء كائن Typer
app = typer.Typer()

# تعريف أمر لجمع رقمين
@app.command()
def sum(x: int, y: int):  # نوع البيانات
    print(x + y)

# تعريف أمر لضرب رقمين
@app.command()
def mul(x: int, y: int):
    print(x * y)

# التحقق من تشغيل السكريبت مباشرة (وليس استيراده)
if __name__ == '__main__':
    # تشغيل تطبيق Typer
    app()
```

### التوضيح:

- `app = typer.Typer()`: إنشاء كائن Typer.
- `@app.command()`: وسم لتسجيل دالتي `sum` و `mul` كأوامر.
- `def sum(x: int, y: int)`: تعريف دالة `sum` التي تأخذ رقمين صحيحين `x` و `y`.
- `print(x + y)`: طباعة مجموع `x` و `y`.
- `def mul(x: int, y: int)`: تعريف دالة `mul` التي تأخذ رقمين صحيحين `x` و `y`.
- `print(x * y)`: طباعة حاصل ضرب `x` و `y`.
- `if __name__ == '__main__'`: التحقق من تشغيل السكريبت مباشرة.
- `app()`: تشغيل تطبيق Typer.

## مثال 3: تسجيل الأوامر لاحقًا

هذا المثال يوضح كيفية تسجيل الأوامر بعد تعريفها.

### الكود:

```
# استيراد مكتبة Typer
import typer

# تعريف دوال للجمع والضرب
def sum(x: int, y: int):  # نوع البيانات
    print(x + y)
```

```
def mul(x: int, y: int):
    print(x * y)

# التحقق من تشغيل السكريبت مباشرةً (وليس استيراده)
if __name__ == '__main__':
    # إنشاء كائن Typer
    app = typer.Typer()
    # تسجيل دالتي sum و mul كأوامر
    app.command()(sum)
    app.command()(mul)
    # تشغيل تطبيق Typer
    app()
```

### التوضيح:

- `app.command()(sum)`: تسجيل دالة `sum` كأمر.
- `app.command()(mul)`: تسجيل دالة `mul` كأمر.

## مثال 4: أمر بخيارات

هذا المثال يوضح أمر Typer يحتوي على خيار.

### الكود:

```
# استيراد مكتبة Typer
import typer

# إنشاء كائن Typer
app = typer.Typer()

# تعريف أمر بخيار
@app.command()
def calc(salary
```