Python Dependency Inversion Principle

Summary: in this tutorial, you'll learn about the Python dependency inversion principle to make your code hi

Introduction to the dependency inversion principle

The dependency inversion principle is one of the five SOLID principles in object-oriented programming:

- **S** – Single responsibility Principle (https://www.pythontutorial.net/python-oop/python-single-responsibility-principle/)
- **O** – Open-closed Principle (https://www.pythontutorial.net/python-oop/python-open-closed-principle/)
- **L** – Liskov Substitution Principle (https://www.pythontutorial.net/python-oop/python-liskov-substitution-principle/)
- **I** – Interface Segregation Principle (https://www.pythontutorial.net/python-oop/python-interface-segregation-principle/)
- **D** – Dependency Inversion Principle

The dependency inversion principle states that:

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

The dependency inversion principle aims to reduce the coupling between classes by creating an abstraction layer between them.

See the following example:

```python
class FXConverter:
    def convert(self, from_currency, to_currency, amount):
        print(f'{amount} {from_currency} = {amount * 1.2} {to_currency}')
        return amount * 1.2


class App:
    def start(self):
        converter = FXConverter()
        converter.convert('EUR', 'USD', 100)


if __name__ == '__main__':
    app = App()
    app.start()
```

In this example, we have two classes `FXConverter` and `App`.

The `FXConverter` class uses an API from an imaginary FX third-party to convert an amount from one currency to another. For simplicity, we hardcoded the exchange rate as `1.2`. In practice, you will need to make an API call to get the exchange rate.

The `App` class has a `start()` method that uses an instance of the `FXconverter` class to convert 100 EUR to USD.

The `App` is a high-level module. However, The `App` depends heavily on the `FXConverter` class that is dependent on the FX's API.

In the future, if the FX's API changes, it'll break the code. Also, if you want to use a different API, you'll need to change the `App` class.

To prevent this, you need to **invert the dependency** so that the `FXConverter` class needs to adapt to the `App` class.

To do that, you define an interface and make the `App` dependent on it instead of `FXConverter` class. And then you change the `FXConverter` to comply with the interface.

First, define an abstract class (https://www.pythontutorial.net/python-oop/python-abstract-class/) `CurrencyConverter` that acts as an interface. The `CurrencyConverter` class has the `convert()` method that all of its subclasses must implement:

```
from abc import ABC


class CurrencyConverter(ABC):
    def convert(self, from_currency, to_currency, amount) -> float:
        pass
```

Second, redefine the `FXConverter` class so that it inherits (https://www.pythontutorial.net/python-oop/python-inheritance/) from the `CurrencyConverter` class and implement the `convert()` method:

```
class FXConverter(CurrencyConverter):
    def convert(self, from_currency, to_currency, amount) -> float:
        print('Converting currency using FX API')
        print(f'{amount} {from_currency} = {amount * 1.2} {to_currency}')
        return amount * 2
```

Third, add the `__init__` (https://www.pythontutorial.net/python-oop/python-__init__/) method to the `App` class and initialize the `CurrencyConverter` 's object:

```
class App:
    def __init__(self, converter: CurrencyConverter):
        self.converter = converter

    def start(self):
        self.converter.convert('EUR', 'USD', 100)
```

Now, the `App` class depends on the `CurrencyConverter` interface, not the `FXConverter` class.

The following creates an instance of the `FXConverter` and pass it to the `App` :

```
if __name__ == '__main__':
    converter = FXConverter()
    app = App(converter)
    app.start()
```

Output:

```
Converting currency using FX API
100 EUR = 120.0 USD
```

In the future, you can support another currency converter API by subclassing the `CurrencyConverter` class. For example, the following defines the `AlphaConverter` class that inherits from the `CurrencyConverter` .

```python
class AlphaConverter(CurrencyConverter):
    def convert(self, from_currency, to_currency, amount) -> float:
        print('Converting currency using Alpha API')
        print(f'{amount} {from_currency} = {amount * 1.2} {to_currency}')
        return amount * 1.15
```

Since the `AlphaConvert` class inherits from the `CurrencyConverter` class, you can use its object in the `App` class without changing the `App` class:

```python
if __name__ == '__main__':
    converter = AlphaConverter()
    app = App(converter)
    app.start()
```

Output:

```
Converting currency using Alpha API
100 EUR = 120.0 USD
```

Put it all together.

```python
from abc import ABC


class CurrencyConverter(ABC):
    def convert(self, from_currency, to_currency, amount) -> float:
        pass


class FXConverter(CurrencyConverter):
    def convert(self, from_currency, to_currency, amount) -> float:
        print('Converting currency using FX API')
        print(f'{amount} {from_currency} = {amount * 1.2} {to_currency}')
        return amount * 1.15


class AlphaConverter(CurrencyConverter):
    def convert(self, from_currency, to_currency, amount) -> float:
        print('Converting currency using Alpha API')
        print(f'{amount} {from_currency} = {amount * 1.2} {to_currency}')
        return amount * 1.2


class App:
    def __init__(self, converter: CurrencyConverter):
        self.converter = converter

    def start(self):
        self.converter.convert('EUR', 'USD', 100)
```

```
if __name__ == '__main__':
    converter = AlphaConverter()
    app = App(converter)
    app.start()
```

Summary

- Use the dependency inversion principle to make your code more robust by making the high-level module dependent on the abstraction, not the concrete implementation.