

✎ Abstraction

- `dunder method` 는 `__` 로 시작해서 `__` 로 끝나는 메서드이고, 매직 메서드라고도 불리운다.
 - `dunder`는 double underscore의 의미 입니다 😊
 - `dir()` 함수를 통해 확인을 할 수 있습니다. 😎
- 정식 인터페이스를 사용하고 싶다면 아래 3가지를 해야 한다.
 - `abc.ABCMeta` 상속
 - `__subclasshook__()` 에 구현해야할 메서드 정의
 - `abc.abstractMethod` 추가
- 아래 4가지를 익힐 수 있다.
 - **Understand** how interfaces work and the caveats of Python interface creation
 - **Comprehend** how useful interfaces are in a dynamic language like Python
 - **Implement** an informal Python interface
 - **Use** `abc.ABCMeta` and `@abc.abstractmethod` to implement a formal Python interface

📄 Info

- 파이썬의 경우 다른 언어의 인터페이스 구현과는 다르고, 디자인의 복잡성에 따라 다양하게 구현이 가능하다.
- 파이썬은 interface 지시어가 없다.
- abstract method는 interface를 구현하는 단순한 방식이다.

01.1 Informal interface

✎ informal interface

```
python
1 class InformalParserInterface:
2     def load_data_source(self, path: str, file_name: str) -> str:
3         """Load in the file for extracting text."""
4         pass
5
6     def extract_text(self, full_file_name: str) -> dict:
7         """Extract text from the currently loaded file."""
8         pass
```

구현은 concrete class 에서 `InformalParserInterface` 를 구현하면 된다.

- `duck typing` 으로 보면 된다.

✎ 인터페이스를 구현한 PdfParser

```
class PdfParser(InformalParserInterface):
    """Extract text from a PDF"""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides InformalParserInterface.load_data_source()"""
        pass

    def extract_text(self, full_file_path: str) -> dict:
        """Overrides InformalParserInterface.extract_text()"""
        pass
```

인터페이스를 구현한 EmlParser

```
class EmlParser(InformalParserInterface):
    """Extract text from an email"""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides InformalParserInterface.load_data_source()"""
        pass

    def extract_text_from_email(self, full_file_path: str) -> dict:
        """A method defined only in EmlParser.
        Does not override InformalParserInterface.extract_text()
        """
        pass
```

isinstance 함수를 통해 확인

```
>>> # Check if both PdfParser and EmlParser implement
InformalParserInterface
>>> isinstance(PdfParser, InformalParserInterface)
True

>>> isinstance(EmlParser, InformalParserInterface)
True
```

- 하지만 실제로는 subclass가 아니다.
 - `extract_text` 메서드를 오버라이딩 하지 않고 `extract_text_from_email`를 구현함

use MRO(Method Resolution Order)

- `__mro__`를 dunder method라고 부름

```
>>> PdfParser.__mro__
(__main__.PdfParser, __main__.InformalParserInterface, object)
```

```
>>> EmlParser.__mro__
(__main__.EmlParser, __main__.InformalParserInterface, object)
```

위와 같은 informal interface 구현은 소규모 개발에서 작은 프로젝트 진행 시 괜찮다.
하지만!! 점점 프로젝트가 커지기 시작하면 에러 찾는데 수많은 시간을 허비해야할 수도 있다.

02.2 Using Meta Class

Informal Interface 구현의 문제점들...

- 이상적으로는 `issubclass(EmlParser, InformalParserInterface)` 가 구현되지 않은 인터페이스에 대한 정보를 정확히 전달해 주어야 합니다.
- `metaclass` 를 활용해서 이를 구현할 수 있습니다.
- 방법
 - `ParserMeta` 상속
 - dunder method 2개 구현 (`__instancecheck__()`, `__subclasscheck__()`)

ParseMeta를 상속? 받는 코드

```
class ParserMeta(type):
    """A Parser metaclass that will be used for parser class creation.
    """
    def __instancecheck__(cls, instance):
        return cls.__subclasscheck__(type(instance))

    def __subclasscheck__(cls, subclass):
        return (hasattr(subclass, 'load_data_source') and
                callable(subclass.load_data_source) and
                hasattr(subclass, 'extract_text') and
                callable(subclass.extract_text))

class UpdatedInformalParserInterface(metaclass=ParserMeta):
    """This interface is used for concrete classes to inherit from.
    There is no need to define the ParserMeta methods as any class
    as they are implicitly made available via .__subclasscheck__().
    """
    pass
```

새로운 PdfParserNew 클래스

```
class PdfParserNew:
    """Extract text from a PDF."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides
UpdatedInformalParserInterface.load_data_source()"""
        pass

    def extract_text(self, full_file_path: str) -> dict:
        """Overrides UpdatedInformalParserInterface.extract_text()"""
        pass
```

Info

- PdfParserNew overrides .load_data_source() and .extract_text()
- isinstance(PdfParserNew, UpdatedInformalParserInterface) return True

새로운 EmlParserNew 클래스

```
class EmlParserNew:
    """Extract text from an email."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides UpdatedInformalParserInterface.load_data_source()"""
        pass

    def extract_text_from_email(self, full_file_path: str) -> dict:
        """A method defined only in EmlParser.
Does not override UpdatedInformalParserInterface.extract_text()
"""
        pass
```

Info


- UpdatedInformalParserInterface 를 구현하기 위한 메타 클래스를 이용해서 클래스 상속을 명시적으로 작성(클래스의 함수처럼 인자에 슈퍼 클래스 넣는 것)하지 않고도 상속 관계를 만들 수 있다.
- 단지 필요한 메서드만 구현하면 된다. (hasattr, callable)
 - 구현되어 있지 않다면 isinstance 함수는 False를 반환할 것입니다.

required method 구현 여부에 따라 isinstance 의 반환 값이 결정되는 것을 확인


- EmlParserNew는 .extract_text() 를 구현하고 있지 않음

```
>>> isinstance(PdfParserNew, UpdatedInformalParserInterface)
True
```

```
>>> issubclass(EmlParserNew, UpdatedInformalParserInterface)
False
```

 그러나 저러나 mro는 명확히 상속관계에 따른 값을 반환해주지 못합니다.

```
>>> PdfParserNew.__mro__
(<class '__main__.PdfParserNew'>, <class 'object'>)
```

 Note

- UpdatedInformalParserInterface is a **virtual base class** of PdfParserNew.

01.3 Using Virtual Base Classes

 Info

- 위와 같은 차이점은 `__subclasscheck__`의 경우 `issubclass()`를 통해 체크가 되지만 `__mro__`의 경우 정식 상속(클래스의 인자에 슈퍼클래스를 넣는 방식)을 통해서만

 Explain

1. The metaclass PersonMeta
2. The base class PersonSuper
3. The Python interface Person

```
class PersonMeta(type):
    """A person metaclass"""
    def __instancecheck__(cls, instance):
        return cls.__subclasscheck__(type(instance))

    def __subclasscheck__(cls, subclass):
        return (hasattr(subclass, 'name') and
                callable(subclass.name) and
                hasattr(subclass, 'age') and
                callable(subclass.age))

class PersonSuper:
    """A person superclass"""
    def name(self) -> str:
        pass

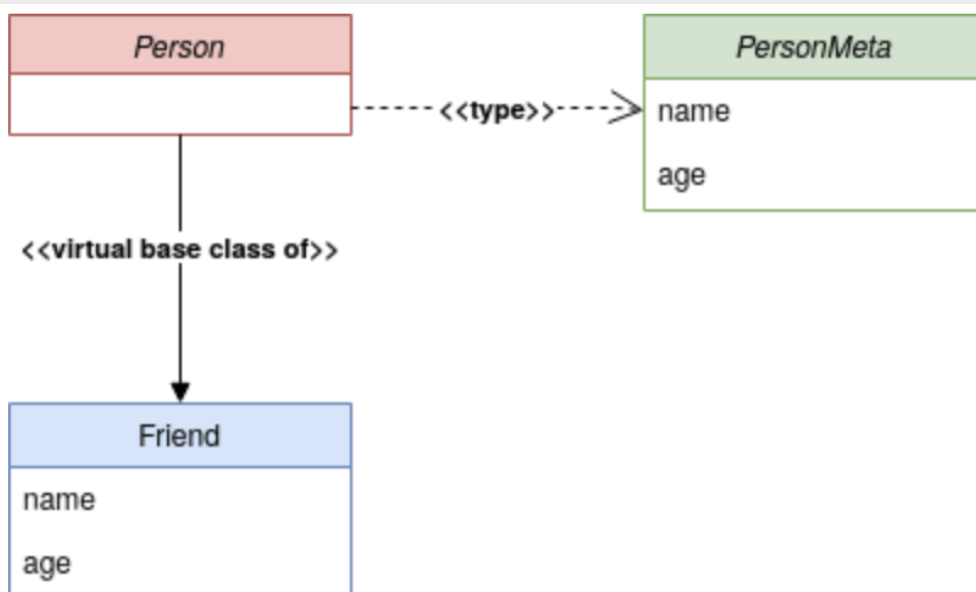
    def age(self) -> int:
```

pass

```
class Person(metaclass=PersonMeta):  
    """Person interface built from PersonMeta metaclass."""  
    pass
```

✎ 두 가지 방식의 상속 방법

```
# Inheriting subclasses  
class Employee(PersonSuper):  
    """Inherits from PersonSuper  
    PersonSuper will appear in Employee.__mro__  
    """  
    pass  
  
class Friend:  
    """Built implicitly from Person  
    Friend is a virtual subclass of Person since  
    both required methods exist.  
    Person not in Friend.__mro__  
    """  
    def name(self):  
        pass  
  
    def age(self):  
        pass
```



[그림1] 정식 상속과 virtual base class를 통한 구현

01.4 Formal Interface

Info

- 작은 회사 혹은 프로젝트에서는 Informal Interface를 사용해도 좋다.
- abc 모듈로 시작해 여러 도구를 통해 Formal Interface를 구현하는 방법을 알아보자.

01.4.1 Using `abc.ABCMeta`

Info

- 앞서 직접 metaclass를 작성한 것을 해 보았는데 `abc.ABCMeta`를 사용해 보자.
- you'll overwrite `__subclasshook__()` in place of `__instancecheck__()` and `__subclasscheck__()`, as it creates a more reliable implementation of these dunder methods.

01.4.1 Using `__subclasshook__()`

 `abc.ABCMeta`를 활용해서 활용하는 방법 (`__subclasshook__()`)을 구현

```
import abc

class FormalParserInterface(metaclass=abc.ABCMeta):
    @classmethod
    def __subclasshook__(cls, subclass):
        return (hasattr(subclass, 'load_data_source') and
                callable(subclass.load_data_source) and
                hasattr(subclass, 'extract_text') and
                callable(subclass.extract_text))

class PdfParserNew:
    """Extract text from a PDF."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides FormalParserInterface.load_data_source()"""
        pass

    def extract_text(self, full_file_path: str) -> dict:
        """Overrides FormalParserInterface.extract_text()"""
        pass

class EmlParserNew:
    """Extract text from an email."""
    def load_data_source(self, path: str, file_name: str) -> str:
```

```

    """Overrides FormalParserInterface.load_data_source()"""
    pass

    def extract_text_from_email(self, full_file_path: str) -> dict:
        """A method defined only in EmlParser.
        Does not override FormalParserInterface.extract_text()
        """
    pass

```

Info

- `issubclass`를 PdfParserNew, EmlParserNew 클래스에 적용해보면 각각 True, False를 반환
- `register()` 함수를 사용해서 virtual subclass로 등록할 수 있습니다.

 meta 함수인 `register`를 사용해서 Double의 virtual Subclass로 등록합니다.

```

class Double(metaclass=abc.ABCMeta):
    """Double precision floating point number."""
    pass

```

```
Double.register(float)
```

```

>>> issubclass(float, Double)
True

```

```

>>> isinstance(1.2345, Double)
True

```

 decorator를 통해서도 Virtual Subclass를 등록할 수 있습니다. 😊

```

@Double.register
class Double64:
    """A 64-bit double-precision floating-point number."""
    pass

print(issubclass(Double64, Double))  # True

```

 duck typing과 decorator를 이용한 방식 모두 `issubclass()` 를 만족시킬 수 있다.

- 만약 `__subclasshook__()` 과 Decorator를 같이 사용할 때 을 경우 `NotImplemented` 를 추가해 주어야 한다.
- PdfParserNew 클래스의 경우 `load_data_source` 와 `extract_text` 를 구현 하였다.
- EmlParserNew 클래스의 경우 `extract_text` 를 구현하지 않았지만 Decorator를 통해 상속 관계를 맺고 있다.


```

class FormalParserInterface(metaclass=abc.ABCMeta):
    @classmethod
    def __subclasshook__(cls, subclass):
        return (hasattr(subclass, 'load_data_source') and
                callable(subclass.load_data_source) and
                hasattr(subclass, 'extract_text') and
                callable(subclass.extract_text) or
                NotImplemented)

class PdfParserNew:
    """Extract text from a PDF."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides FormalParserInterface.load_data_source()"""
        pass

    def extract_text(self, full_file_path: str) -> dict:
        """Overrides FormalParserInterface.extract_text()"""
        pass

@FormalParserInterface.register
class EmlParserNew:
    """Extract text from an email."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides FormalParserInterface.load_data_source()"""
        pass

    def extract_text_from_email(self, full_file_path: str) -> dict:
        """A method defined only in EmlParser.
        Does not override FormalParserInterface.extract_text()
        """
        pass

print(issubclass(PdfParserNew, FormalParserInterface)) # True
print(issubclass(EmlParserNew, FormalParserInterface)) # True

```

Info

`extract_text()` 를 구현하지 않은 `EmlParserNew` 도 `subclass` 로 인정하는 것은 클라이언트의 의도라고 보기 어렵기 때문에 주의하여야 한다.

01.4.2 Using abstract method declaration

Info

- `@abc.abstractmethod` 데코레이터 사용을 통해 abstract method를 구현할 수 있다.
-

```
class FormalParserInterface(metaclass=abc.ABCMeta):
    @classmethod
    def __subclasshook__(cls, subclass):
        return (hasattr(subclass, 'load_data_source') and
                callable(subclass.load_data_source) and
                hasattr(subclass, 'extract_text') and
                callable(subclass.extract_text) or
                NotImplemented)

    @abc.abstractmethod
    def load_data_source(self, path: str, file_name: str):
        """Load in the data set"""
        raise NotImplementedError

    @abc.abstractmethod
    def extract_text(self, full_file_path: str):
        """Extract text from the data set"""
        raise NotImplementedError

class PdfParserNew(FormalParserInterface):
    """Extract text from a PDF."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides FormalParserInterface.load_data_source()"""
        pass

    def extract_text(self, full_file_path: str) -> dict:
        """Overrides FormalParserInterface.extract_text()"""
        pass

class EmlParserNew(FormalParserInterface):
    """Extract text from an email."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides FormalParserInterface.load_data_source()"""
        pass

    def extract_text_from_email(self, full_file_path: str) -> dict:
        """A method defined only in EmlParser.
        Does not override FormalParserInterface.extract_text()
        """
        pass
```

```
>>> pdf_parser = PdfParserNew()  
>>> eml_parser = EmlParserNew()  
Traceback (most recent call last):  
...  
TypeError: Can't instantiate abstract class EmlParserNew with abstract  
methods extract_text
```

references

<https://realpython.com/python-interface/>