



OOP in Python

PyThing 이승헌

T.O.C

1. Object Oriented Programming in Python

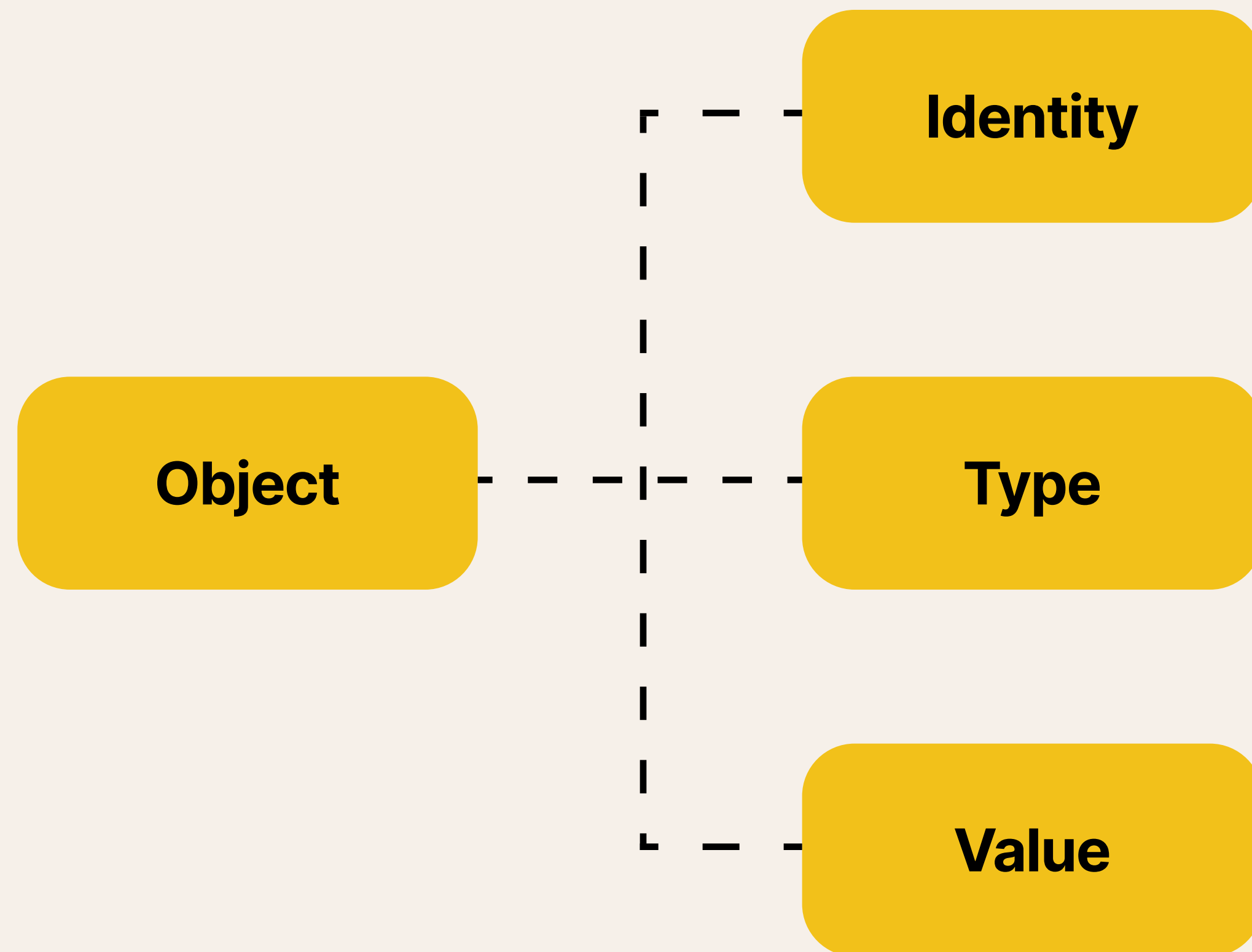
- a. The Whole Thing is Object!
- b. Magic Methods

2. Object Oriented Programming in Python

- a. abc.ABC
- b. typing.Protocol

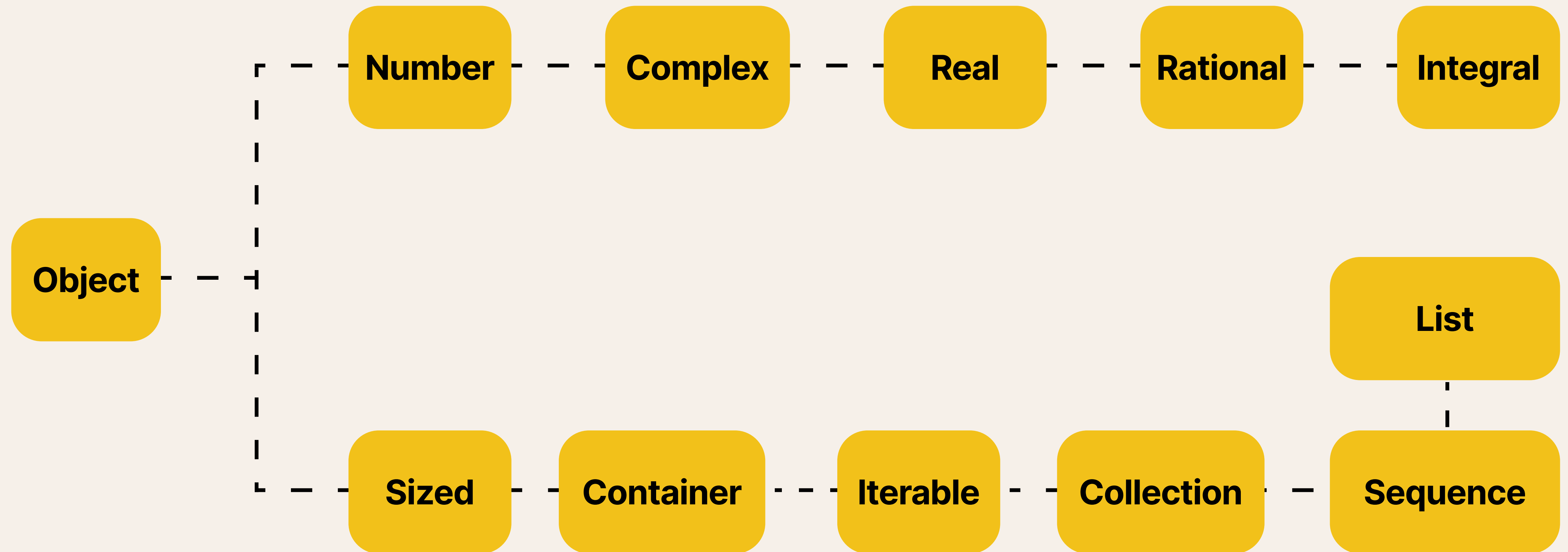
Object Oriented Programming **in Python**

The Whole Thing is Object!



```
1 # in REPL
2
3
4 >>> x = [1,2,3] # object
5 >>> id(x)
6 4342794560      # Identity
7 >>> type(x)
8 <class 'list'> # Type
9 >>> x
10 [1, 2, 3]      # Value
11
12
13
```

The Whole Thing is Object!



Special Methods

```
1 class Car:
2     def __init__(self, engine):
3         self.engine = engine
4
5     def __repr__(self):
6         return f"Car(engine={self.engine})"
7
```

Python's approach to operator overloading, **allowing classes to define their own behavior** with respect to language operators.

dunder method # magic method

Special Methods

```
__new__, __init__, __del__, __repr__, __str__, __bytes__, __format__, __hash__,  
__bool__, __getattr__, __getattribute__, __setattr__, __delattr__, __dir__,  
__eq__, __ne__, __lt__, __le__, __gt__, __ge__, __get__, __set__, __delete__,  
__set_name__, __len__, __length_hint__, __getitem__, __setitem__,  
__delitem__, __missing__, __iter__, __reversed__, __contains__, __call__,  
__enter__, __exit__, __neg__, __pos__, __abs__, __invert__, __add__, __sub__,  
__mul__, __matmul__, __truediv__, __floordiv__, __mod__, __divmod__,  
__pow__, __lshift__, __rshift__, __and__, __xor__, __or__, __radd__, __rsub__,  
__rmul__, __rmatmul__, __rtruediv__, __rfloordiv__, __rmod__, __rdivmod__,  
__rpow__, __rlshift__, __rrshift__, __rand__, __rxor__, __ror__, __iadd__,  
__isub__, __imul__, __imatmul__, __itruediv__, __ifloordiv__, __imod__,  
__ipow__, __ilshift__, __irshift__, __iand__, __ixor__, __ior__, __complex__,  
__int__, __float__, __index__, __round__, __trunc__, __floor__, __ceil__,  
__class_getitem__, __instancecheck__, __subclasscheck__,  
__init_subclass__, __prepare__, __class__, __reduce__, __reduce_ex__,  
__getstate__, __setstate__
```

109개의 Special Methods

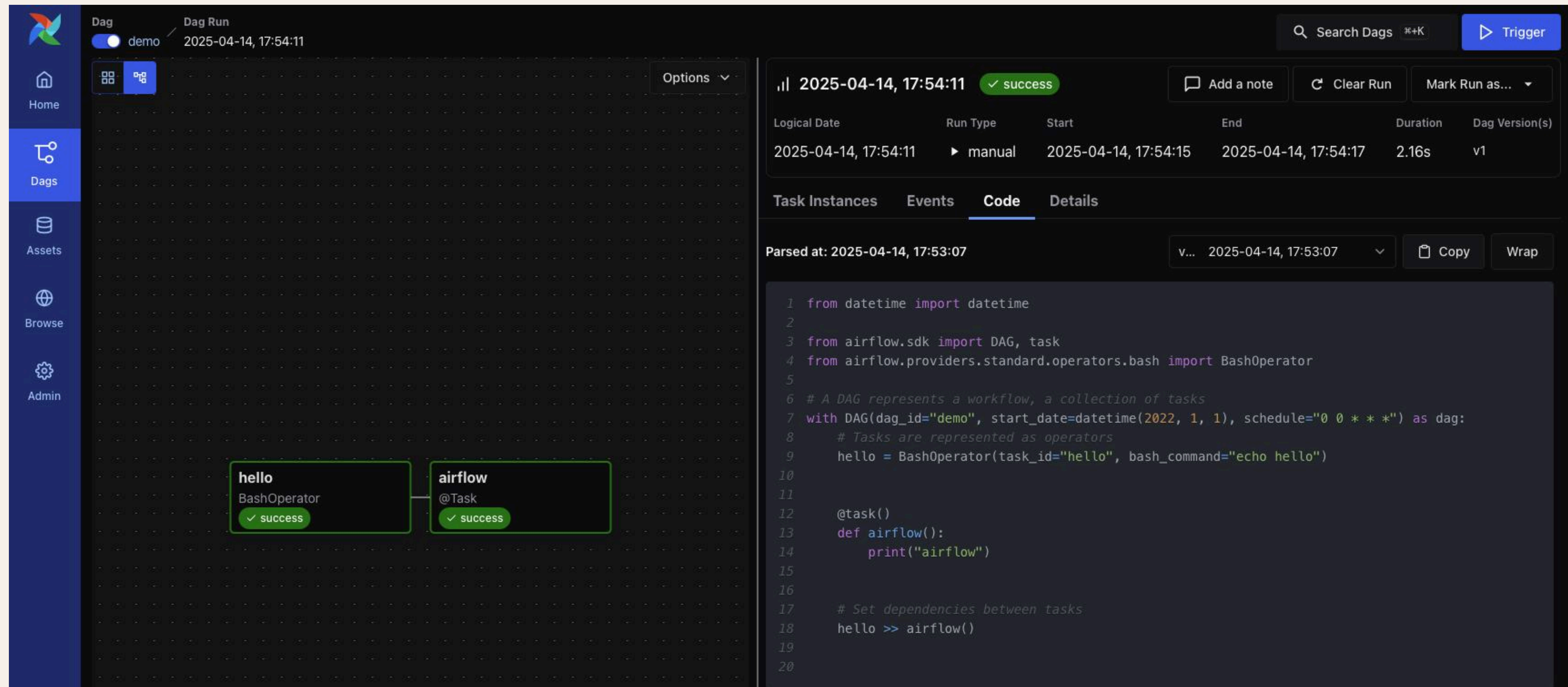
Use Case: Apache Airflow



Apache Airflow[®] is an open-source platform for developing, scheduling, and monitoring batch-oriented workflows.

Data Engineering # ETL # Machine Learning Pipeline # Python

Use Case: Apache Airflow



The screenshot displays the Apache Airflow web interface. On the left, a navigation sidebar includes Home, Dags, Assets, Browse, and Admin. The main area shows a DAG run for 'demo' on 2025-04-14 at 17:54:11, which is successful. Below this, a task graph shows two tasks: 'hello' (BashOperator) and 'airflow' (@Task), both marked as successful. The right panel shows the code for the DAG, which defines a DAG with a single task 'hello' that prints 'airflow'.

```
1 from datetime import datetime
2
3 from airflow.sdk import DAG, task
4 from airflow.providers.standard.operators.bash import BashOperator
5
6 # A DAG represents a workflow, a collection of tasks
7 with DAG(dag_id="demo", start_date=datetime(2022, 1, 1), schedule="0 0 * * *") as dag:
8     # Tasks are represented as operators
9     hello = BashOperator(task_id="hello", bash_command="echo hello")
10
11
12 @task()
13 def airflow():
14     print("airflow")
15
16
17 # Set dependencies between tasks
18 hello >> airflow()
19
20
```

각 Operator 간의 의존성을 DAG(Directed Acyclic Graph)로 표현

Use Case: Apache Airflow

The screenshot displays the Apache Airflow web interface. On the left, a navigation sidebar includes Home, Dags, Assets, Browse, and Admin. The main content area shows a 'Dag Run' for 'demo' on '2025-04-14, 17:56:36' with a 'Task hello' status of 'success'. A bar chart visualizes task durations, with a 'hello' task highlighted. Below the chart, a table lists task instances for 'hello' and 'airflow', all marked as successful. The 'hello' task instance details are shown below, including the operator 'BashOperator', start and end times, duration of 0.06s, and DAG version 'v1'. The 'Logs' tab is active, displaying a log message with source details and a series of INFO-level log entries. The log entries include: 'DAG bundles loaded', 'Filling up the DagBag', 'Tmp dir root location', 'Running command: ['/usr/bin/bash', '-c', 'echo hello!]', 'Output:', 'hello:', and 'Command exited with return code 0'. The final log entry shows 'Pushing xcom' with detailed task instance information.

실행 성공 여부와 로그 확인

Use Case: Apache Airflow

```

1  from datetime import datetime
2
3  from airflow.sdk import DAG, task
4  from airflow.providers.standard.operators.bash
   import BashOperator
5
6  # A DAG represents a workflow, a collection of
   tasks
7  with DAG(dag_id="demo", start_date=datetime(2022,
   1, 1), schedule="0 0 * * *") as dag:
8      # Tasks are represented as operators
9      hello = BashOperator(task_id="hello",
   bash_command="echo hello")
10
11     @task()
12     def airflow():
13         print("airflow")
14
15     # Set dependencies between tasks
16     hello >> airflow()

```

8 >> 3

Set dependencies between tasks
 hello >> airflow()

```

1  class DependencyMixin:
2      """Mixing implementing common dependency setting
   methods like >> and <<."""
3      # ...
4      def __lshift__(self, other: DependencyMixin |
   Sequence[DependencyMixin]):
5          """Implement Task << Task."""
6          self.set_upstream(other)
7          return other
8
9      def __rshift__(self, other: DependencyMixin |
   Sequence[DependencyMixin]):
10         """Implement Task >> Task."""
11         self.set_downstream(other)
12         return other

```

airflow/task-sdk/src/airflow/sdk/definitions/_internal/mixins.py

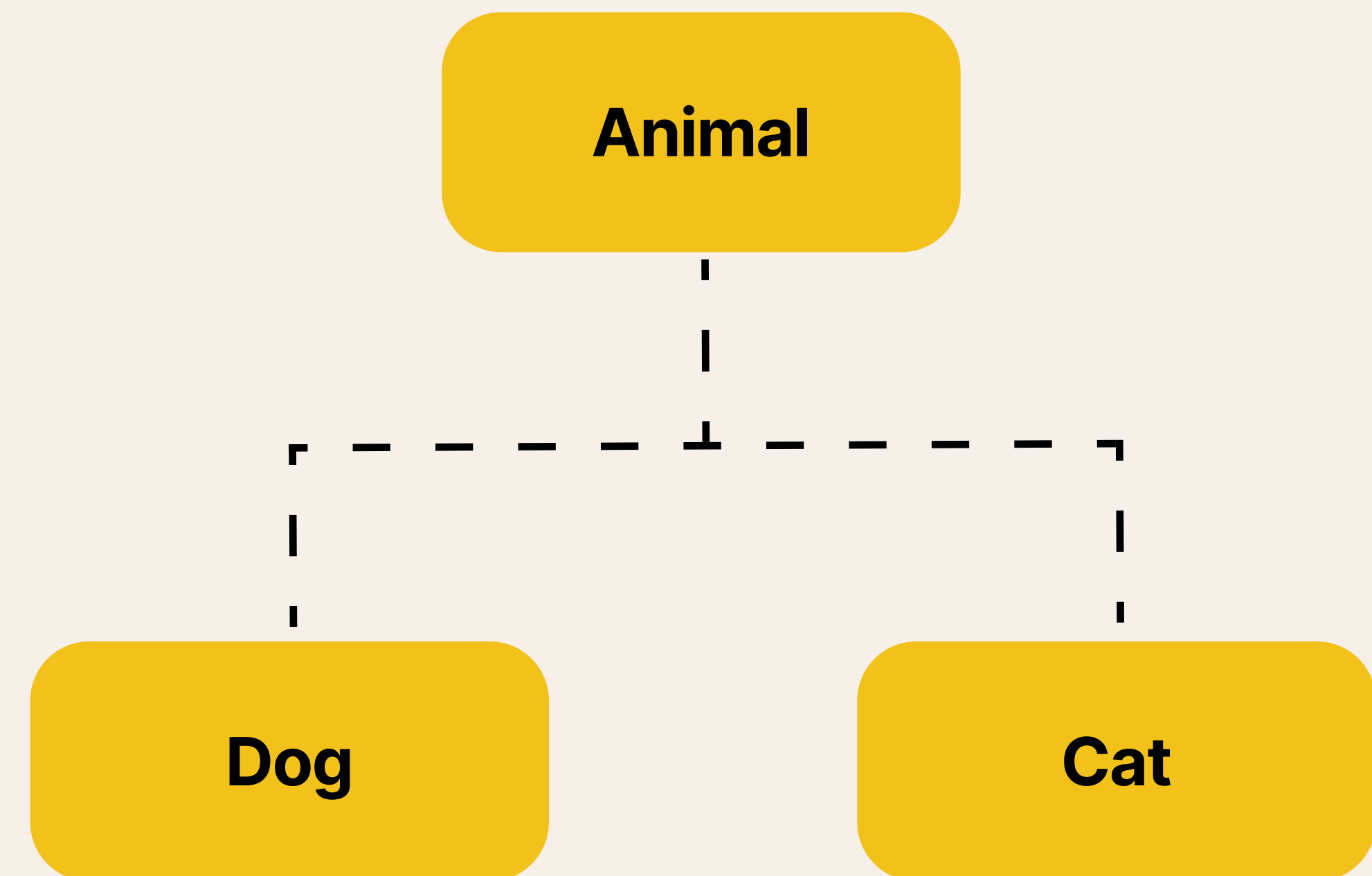
Object Oriented Programming in Python

Object Relationships

- 1. Inheritance**
2. Composition
3. Association
4. Aggregation

Inheritance

```
1 # 부모 클래스 (슈퍼 클래스)
2 class Animal:
3     def __init__(self, name):
4         self.name = name
5
6     def speak(self):
7         return "소리를 낸다"
8
9 # 자식 클래스 (서브 클래스)
10 class Dog(Animal):
11     def speak(self):
12         return f"{self.name}가 말한다: 멍멍!"
13
14 class Cat(Animal):
15     def speak(self):
16         return f"{self.name}가 말한다: 야옹!"
17
18 # 사용 예
19 dog = Dog("초코")
20 cat = Cat("나비")
21
22 print(dog.speak()) # 초코가 말한다: 멍멍!
23 print(cat.speak()) # 나비가 말한다: 야옹!
```



Inheritance

abc — Abstract Base Classes

Source code: [Lib/abc.py](#)

```
1 from abc import ABC, abstractmethod
2
3 class Animal(ABC):
4     @abstractmethod
5     def sound(self):
6         pass
7
8 class Dog(Animal):
9     def sound(self):
10         return "멍멍"
11
```

Nominal Subtyping

Protocols

(Originally specified in [PEP 544](#).)

```
1 from typing import Protocol
2
3 class Speaker(Protocol):
4     def speak(self) -> str:
5         ...
6
7 class Dog:
8     def speak(self) -> str:
9         return "멍멍"
10
11 class Cat:
12     def speak(self) -> str:
13         return "야옹"
14
15 def make_it_talk(animal: Speaker):
16     print(animal.speak())
17
```

Structural Subtyping

abc.ABC

```
1 from abc import ABC, abstractmethod
2
3 class Animal(ABC):
4     @abstractmethod
5     def sound(self):
6         pass
7
8 class Dog(Animal):
9     def sound(self):
10        return "멍멍"
11
12
13 if __name__ == '__main__':
14     dog = Dog()
15     print(dog.sound())
16
```

- **Nominal Subtyping**

- 이름 기반 타입 관계 판단
- 타입 간 상속 관계 명시적 선언

- **ABC**

- Abstract Base Classes를 정의하는 모듈

- **abstractmethod**

- abstract method를 정의할 때 사용
- 반드시 하위 클래스에서 override

- **runtime 검증**

- `__abstractmethods__`을 통해 검증

typing.Protocol

```
1 from typing import Protocol
2
3 class Speaker(Protocol):
4     def speak(self) -> str:
5         ...
6
7 class Dog:
8     def speak(self) -> str:
9         return "멍멍"
10
11 class Cat:
12     def speak(self) -> str:
13         return "야옹"
14
15 def make_it_talk(animal: Speaker):
16     print(animal.speak())
17
18 if __name__ == '__main__':
19     dog = Dog()
20     make_it_talk(dog)
21
```

- **Structural Subtyping**
 - 구조 기반 타입 관계 판단
 - 타입 간 상속 관계 암묵적 선언
- **typing.Protocol**
 - Structural Subtyping을 위한 인터페이스 선언 형식
- **정적분석기를 통한 검증**
 - mypy, pyright를 통한 정적 분석

abc.ABC vs. typing.Protocol

abc.ABC	vs.	typing.Protocol
명시적 상속	기본 철학	구조적 서브타이핑
반드시 상속해야 함	상속 필요 여부	상속하지 않아도 됨
구현 강제, 인터페이스 계약	목적	타입 검사 시 구조 검증
추상 메서드 강제, 인스턴스 생성 차단	런타임 효과	기본적으로는 타입 검사용 (런타임 무관)

E.O.D