What is Object-Oriented Programming

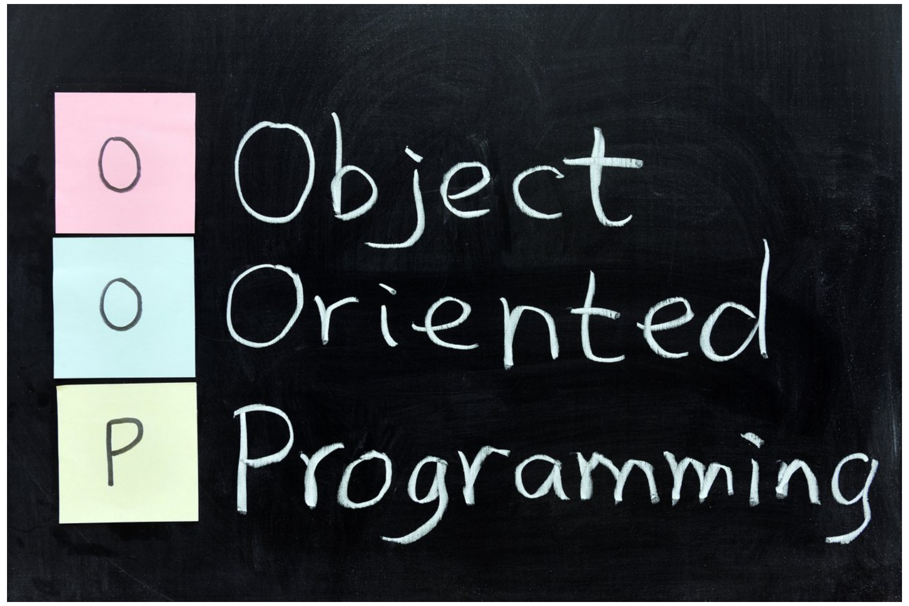
PROGRAMMING PARADIGM CONCEPTS

Eleanor Thomas
Senior Data Analytics Engineer

datacamp

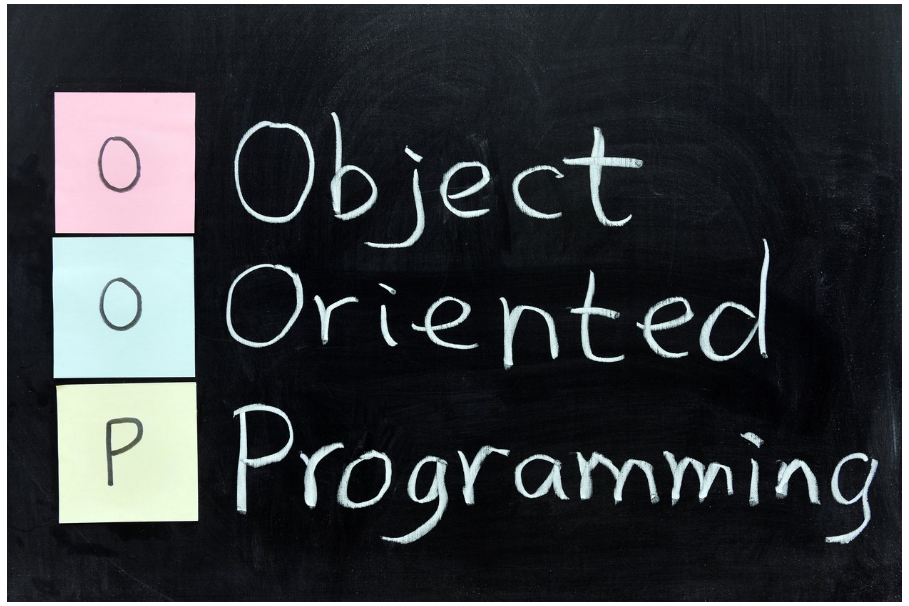
What is object-oriented programming

- **Object-Oriented Programming:** style of programming organized into "objects"
- **Objects:** basic unit of object-oriented programming, can contain data and code



What are classes?

- **Classes:** categories of objects containing shared functionality and information among all objects from that class
- Neither classes nor objects are *processes* like functions, but can *contain* information about processes



Classes vs. objects

Classes

- General category of objects
- Includes potential for many examples of that class

Objects

- A specific example of a particular class
- Reflects common traits among all members of the class as well as individual characteristics

Object-oriented programming in Python

```python
class Dog():
    def __init__(self, name):
        self.name = name

    def bark(self):
        print("Arf!")

lacy = Dog("Lacy")
lacy.bark()
```

Output:

```
Arf!
```

Let's practice!

PROGRAMMING PARADIGM CONCEPTS

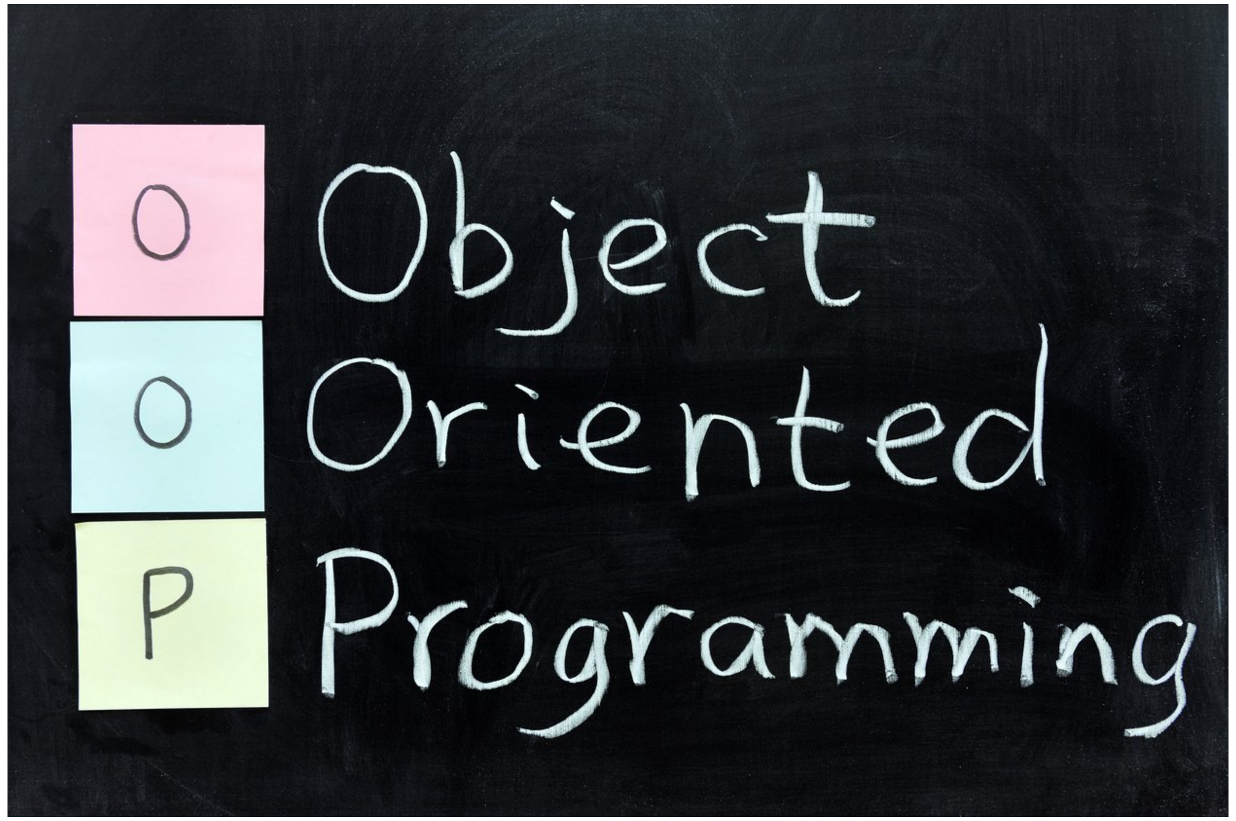
Examples of Object-Oriented Programming

PROGRAMMING PARADIGM CONCEPTS

Eleanor Thomas
Senior Data Analytics Engineer

Applications of object-oriented programming

- Simulations of various types (e.g. stock prices, physics problems)
- Object-oriented databases (a specific type of database) like MongoDB
- Problems where many small, similar tasks need to run at the same time (e.g. sorting many independent lists)



Pros and cons of object-oriented programming

Pros

- Maintains security of data
- Allows for parallel development
- Reusable and maintainable

Cons

- Code can be slow to run
- Programs can be longer (more lines of code)
- Not appropriate for all applications -- inappropriate use results in long, slow programs

Public and private attributes

- **Public** attributes and methods are available throughout the program (default in Python)
  - **Dog example:** "name" = public attribute
  - **Dog example:** "bark" = public method
- **Private** attributes and methods are only accessible within the class itself (denoted with `__` prefix in Python)

Public vs. private example

```python
class Dog():
    def __init__(self, name):
        self.name = name
        self.__hungry = True
    def eat(self):
        self.__hungry = False

lacy = Dog("Lacy")
lacy.__hungry = False # This line won't work!!
```

Let's practice!

PROGRAMMING PARADIGM CONCEPTS

Class Inheritance in Object-Oriented Programs

PROGRAMMING PARADIGM CONCEPTS

Eleanor Thomas
Senior Data Analytics Engineer

Class inheritance in object-oriented programming

- **Class inheritance:** when one class "inherits" methods and attributes from another, parent class
- Example:
  - **Poodle** class inherits from **Dog** (and has curly hair)
  - **Dog** class inherits from **Pet** (and "barks")
  - **Pet** class inherits from **Animal**
- No limit to the number of layers of inheritance, but more is not always better

Class inheritance example

```python
class Dog():
    def __init__(self, name):
        self.name = name
    def bark(self):
        print("Arf!")

lacy = Dog("Lacy")
lacy.bark()
```

```python
class Pet():
    def __init__(self, name):
        self.name = name

class Dog(Pet):
    def bark(self):
        print("Arf!")

lacy = Dog("Lacy")
lacy.bark()
```

Class inheritance example continued

```python
class Cat(Pet):
    def meow(self):
        print("Meow!")

class Horse(Pet):
    def neigh(self):
        print("Neigh!")
```

```python
fluffy = Cat("Fluffy")
fluffy.meow()
```

```
Meow!
```

```python
midnight = Horse("Midnight")
midnight.neigh()
```

```
Neigh!
```

All of the following produce Errors!

```python
fluffy.neigh()
```

```python
fluffy.bark()
```

```python
midnight.meow()
```

Let's practice!

Congratulations!

PROGRAMMING PARADIGM CONCEPTS

Eleanor Thomas
Senior Data Analytics Engineer

Chapter 1: Introduction to Programming Paradigms

- Concept of programming paradigms
- Paradigms vs. languages
- Imperative and declarative paradigms
- Separation of responsibilities and modular code

Chapter 2: Procedural Programming

- Procedural programming
- Applications of procedural programming
- Pros and cons of procedural programming
- Control flow

Chapter 3: Functional Programming

- Functional programming
- Pure functions
- Applications of functional programming
- Pros and cons
- Recursion

Chapter 4: Object-Oriented Programming

- Object-oriented programming
- Objects and Classes
- Applications and pros/cons
- Public and private attributes
- Class inheritance

Where to go next

- Dive deeper into one of the paradigms
- Practice solving the same problem with more than one paradigm
- Evaluate the best paradigm to choose for your next project

Congratulations!

PROGRAMMING PARADIGM CONCEPTS