Introduction to Inheritance and Object Oriented Programming

Example Instruction Demo

Dr. Tyler Westland

Outline

What Problems Do Objects Solve

- Programs consistent of data...
  - Simple: numbers and letters
  - Complex: measurements associated with timestamps

What Problems Do Objects Solve

- Programs consistent of data...
  - Simple: numbers and letters
  - Complex: measurements associated with timestamps
- and functions
  - Simple: addition and concatenation
  - Complex: sampling measurements to have timestamps in 30 minute intervals

What Problems Do Objects Solve

- Programs consistent of data...
  - Simple: numbers and letters
  - Complex: measurements associated with timestamps
- and functions
  - Simple: addition and concatenation
  - Complex: sampling measurements to have timestamps in 30 minute intervals
- Objects combine functions with data
  - Simple:

```
>>> "cat" + "dog"
'catdog'
```

  - Complex:

```
>>> measurements.resample("30min").bfill()
0    73.0
30   75.0
60   74.0
dtype: float64
```

What Problems Does Inheritance Solve

Inheritance

The process of creating classes of objects from existing classes

- Allows for reuse of data structures and functions
  - Reused portions can be modified

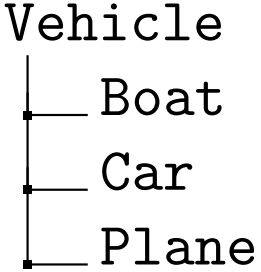
What Problems Does Inheritance Solve

Inheritance

The process of creating classes of objects from existing classes

- Allows for reuse of data structures and functions
  - Reused portions can be modified
- Describes what to expect from every type of an object
  - Every number should allow math operations
  - Every "Vehicle" in a video game should allow "driving"

Example of a Vehicle being the parent to various drivable (children) objects



The Shape Class

code/shapes/shape.py

```
1 from abc import ABC, abstractmethod
2
3 class Shape(ABC):
4     @abstractmethod
5     def area(self) -> float:
6         pass
```

Classes Vs Objects

This is a class, it defines what an object is. This is distinct from being an instance of an object

The Rectangle Class

code/shapes/rectangle.py

```
1 from shapes.shape import Shape
2
3 class Rectangle(Shape):
4     def __init__(self, length: float, width: float):
5         self.length = length
6         self.width = width
7
8     def area(self) -> float:
9         return self.length * self.width
```

Our First Rectangle

code/main.py

```
1 rect = Rectangle(4, 5)
2 print(f"rect area: {rect.area()}")
```

output

```
1 rect area: 20
```

Instances of Objects

rect is an object as it is an instance of Rectangle

Code Snippets

main.py and the output will be shown in snippets. The full text is in the appendix.

Two Rectangles

code/main.py

```
1 rect = Rectangle(4, 5)
2 print(f"rect area: {rect.area()}")
3
4 rect2 = Rectangle(3,8)
5 print(f"rect2 area: {rect2.area():.2f}")
6
7 print(f"rect.area() > rect2.area(): {rect.area() >
      rect2.area()}")
```

output

```
1 rect area: 20
2 rect2 area: 24.00
3 rect.area() > rect2.area(): False
```

The Circle Class

code/shapes/circle.py

```
1 from shapes.shape import Shape
2 import math
3
4 class Circle(Shape):
5     def __init__(self, radius: float):
6         self.radius = radius
7
8     def area(self) -> float:
9         return math.pi * math.pow(self.radius, 2)
```

Comparing Our Shapes

code/main.py

```
rect = Rectangle(4, 5)
print(f"rect area: {rect.area()}")
```

```
circ = Circle(3)
print(f"circ area: {circ.area():.2f}")

print(f"circ.area() > rect.area(): {circ.area() > rect
    .area()}")
```

output

```
rect area: 20
```

```
circ area: 28.27
circ.area() > rect.area(): True
```

The Extended Shape Class

code/shapes/shape.py

```
1  from abc import ABC, abstractmethod
2
3  class Shape(ABC):
4      @abstractmethod
5      def area(self) -> float:
6          pass
7
8      def __gt__(self, other) -> bool:
9          if isinstance(other, Shape):
10             return self.area() > other.area()
11         else:
12             raise ValueError("Can only compare with
       shapes")
```

Comparing Our Shapes, With Style

code/main.py

```
rect = Rectangle(4, 5)
print(f"rect area: {rect.area()}")
```

```
rect2 = Rectangle(3,8)
print(f"rect2 area: {rect2.area():.2f}")
```

```
circ = Circle(3)
print(f"circ area: {circ.area():.2f}")
```

```
print(f"rect > rect2: {rect > rect2}")
print(f"circ > rect: {circ > rect}")
```

output

```
rect area: 20
rect2 area: 24.00
```

```
circ area: 28.27
```

```
rect > rect2: False
circ > rect: True
```

Question Gathering

- Gather into groups of 2-3, assign one person as the recorder
- Follow these steps for 3 minutes:
  1. Ask as many questions as you can
  2. Do not stop to discuss, judge, or answer
  3. Record *exactly* as stated
  4. Change statements into questions

Question Refining

- Follow these steps to refine your questions:
  1. Label each question with a "C" for Closed-Ended, if it can be answered by a yes or no or with one word.
  2. Label each question with an "O" for Open-Ended, if it requires a longer explanation
  3. Choose 1 Close-Ended question and changed it to Open-Ended. Add this new question to the bottom of the list.
  4. Choose 1 Open-Ended question and change it to Close-Ended. Add this new question to the bottom of the list.
  5. Submit questions.

Question Answering

Refer to submitted questions

Folder Structure

```
code/
├── main.py
└── shapes/
    ├── shape.py
    ├── rectangle.py
    └── circle.py
```

main.py

```python
#!/bin/env python3
from shapes.rectangle import Rectangle
from shapes.circle import Circle


rect = Rectangle(4, 5)
print(f"rect area: {rect.area()}")

rect2 = Rectangle(3,8)
print(f"rect2 area: {rect2.area():.2f}")

print(f"rect.area() > rect2.area(): {rect.area() > rect2.area()}")

circ = Circle(3)
print(f"circ area: {circ.area():.2f}")
```

main.py – cont

```
1
2 print(f"circ.area() > rect.area(): {circ.area() > rect
      .area()}")
3
4
5 print(f"rect > rect2: {rect > rect2}")
6 print(f"circ > rect: {circ > rect}")
```

shapes/shape.py

```
1  from abc import ABC, abstractmethod
2
3  class Shape(ABC):
4      @abstractmethod
5      def area(self) -> float:
6          pass
7
8      def __gt__(self, other) -> bool:
9          if isinstance(other, Shape):
10             return self.area() > other.area()
11         else:
12             raise ValueError("Can only compare with
       shapes")
```

shapes/rectangle.py

```python
from shapes.shape import Shape

class Rectangle(Shape):
    def __init__(self, length: float, width: float):
        self.length = length
        self.width = width

    def area(self) -> float:
        return self.length * self.width
```

shapes/circle.py

```python
from shapes.shape import Shape
import math

class Circle(Shape):
    def __init__(self, radius: float):
        self.radius = radius

    def area(self) -> float:
        return math.pi * math.pow(self.radius, 2)
```

output

```
rect area: 20
rect2 area: 24.00
rect.area() > rect2.area(): False
circ area: 28.27
circ.area() > rect.area(): True
rect > rect2: False
circ > rect: True
```