Programming I

Course 10

Introduction to programming

What we talked about?

- Object Oriented Programming
- Classes
- Objects

What we talk today?

- Relation between classes
  - Has a
  - A kinf of
  - Is a

- Inheritance

Classes. Objects

- Abstractization
  - Possibility to add user defined data types (new abstractizations)
- Classes
  - Describe one or more objects
  - A template for creating, or instantiating, specific objects within a program.
- Objects
  - A realization of the class

Classes & Objects

Class Implementation

- implementing a new object type with a class
  - define the class
  - define data attributes (WHAT IS the object)
  - define methods (HOW TO use the object)

Class Usage

- using the new object type in code
  - create instances of the object type
  - do operations with them

Classes & Objects

Class Definition

- class name is the type class Coordinate(object)
- class is defined generically
  - use `self` to refer to some instance while defining the class
  - `self` is a parameter to methods in class definition
- class defines data and methods common across all instances

Class Instantiation

- instance is one specific object `coord = Coordinate(1,2)`
- data attribute values vary between instances
  - `c1 = Coordinate(1,2)`
  - `c2 = Coordinate(3,4)`
  - `c1` and `c2` have different data attribute values `c1.latitude` and `c2.latitude` because they are different objects
- instance has the structure of the class

Objects

- Objects
  - A unique identifier
  - A type
  - A internal representation
  - A set of operations that allows interaction with the information stored in the object

  - INTERACT WITH EACH OTHERS

Objects

- Objects
  - A unique identifier
  - A type
  - A internal representation
  - A set of operations that allows interaction with the information stored in the object

  - INTERACT WITH EACH OTHERS
    - A Bird is a kind of Animal
    - A Team has a list of Employee
    - An Engine is a part of a Car
    - A Shop uses Card Payment

Object Relations

- Inheritance
  - A Bird is a kind of Animal
- Association
  - A Team has a list of Employee
- Composition
  - An Engine is a part of a Car
- Dependency
  - A Shop uses Card Payment

Can be identified by the constructions that are used to describe the relation

Object Relations

- Inheritance
  - A Bird is a kind of Animal
  - Bird class is a subclass of Animal class

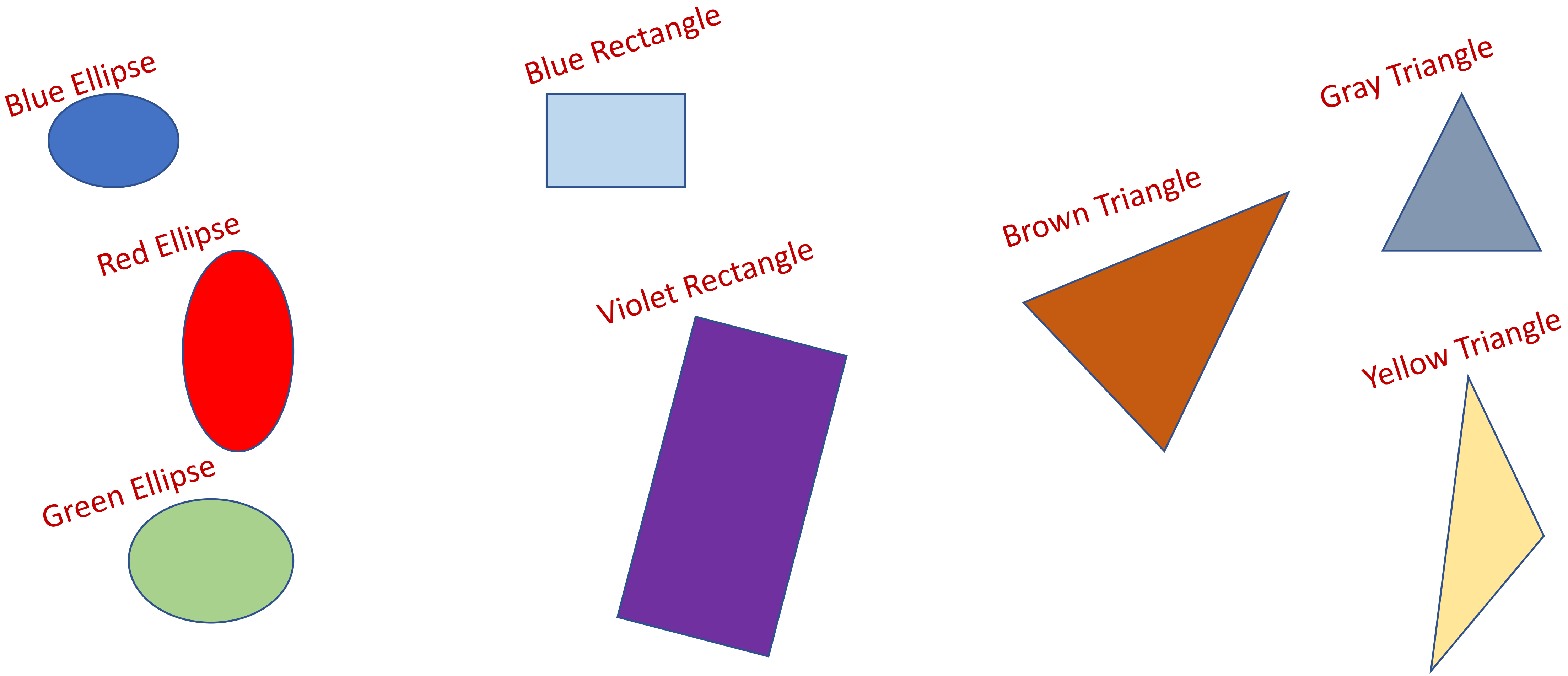
- Association
  - A Team has a list of Employee
  - The Team class has an attribute that contains the list of Employee and does not control the life circle of the employees objects
    - If a team is dissolved the employee are not fired

- Composition
  - An Engine is a part of a Car
  - The Car class has an attribute of type Engine and it controls the life circle of the engine object
    - If the car is destroyed the engine is also destroyed

- Dependency
  - A Shop uses Card Payment
  - One of the methods of Shop class uses a Card Payment object in order to make the payment
  - Card Payment is not an attribute of class Shop

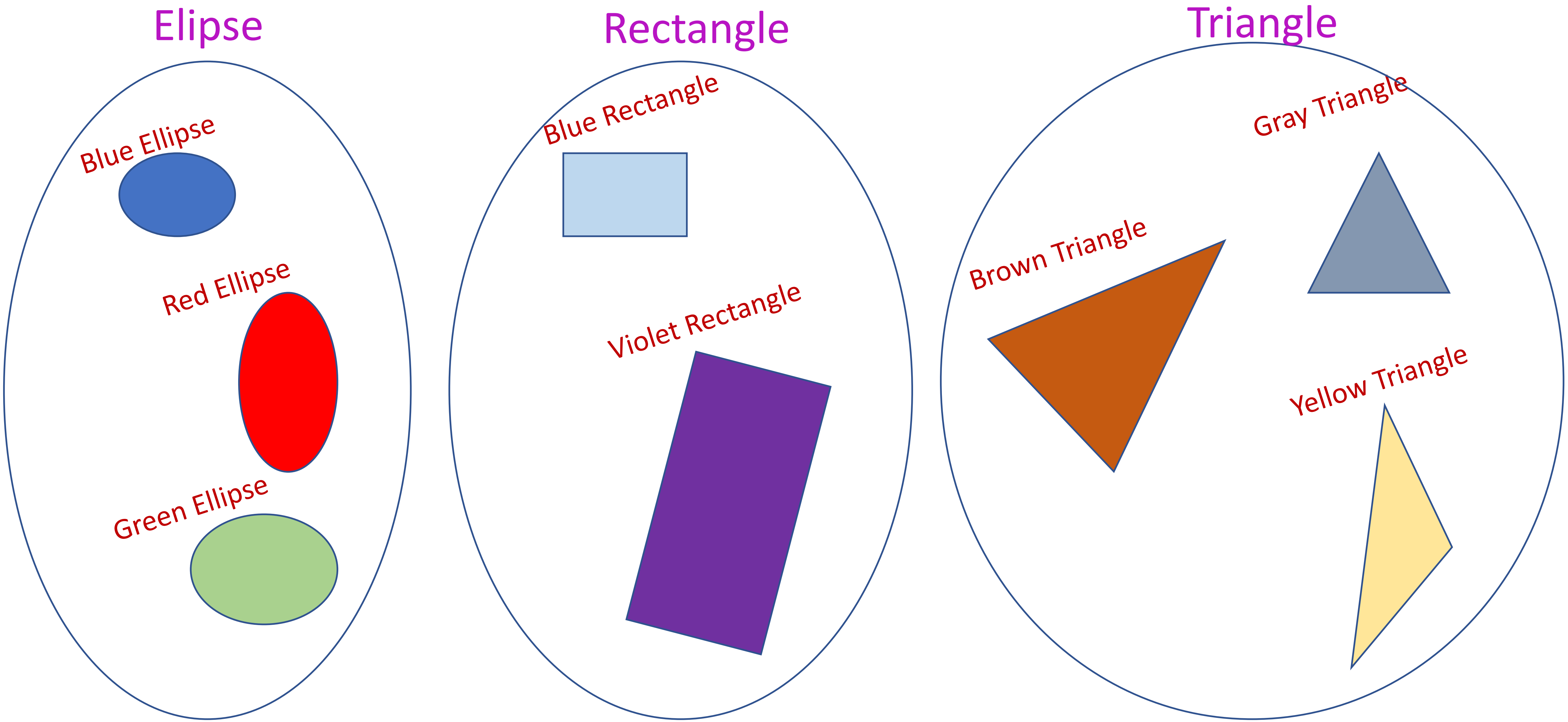
Inheritance

In how many classes can be the objects grouped?



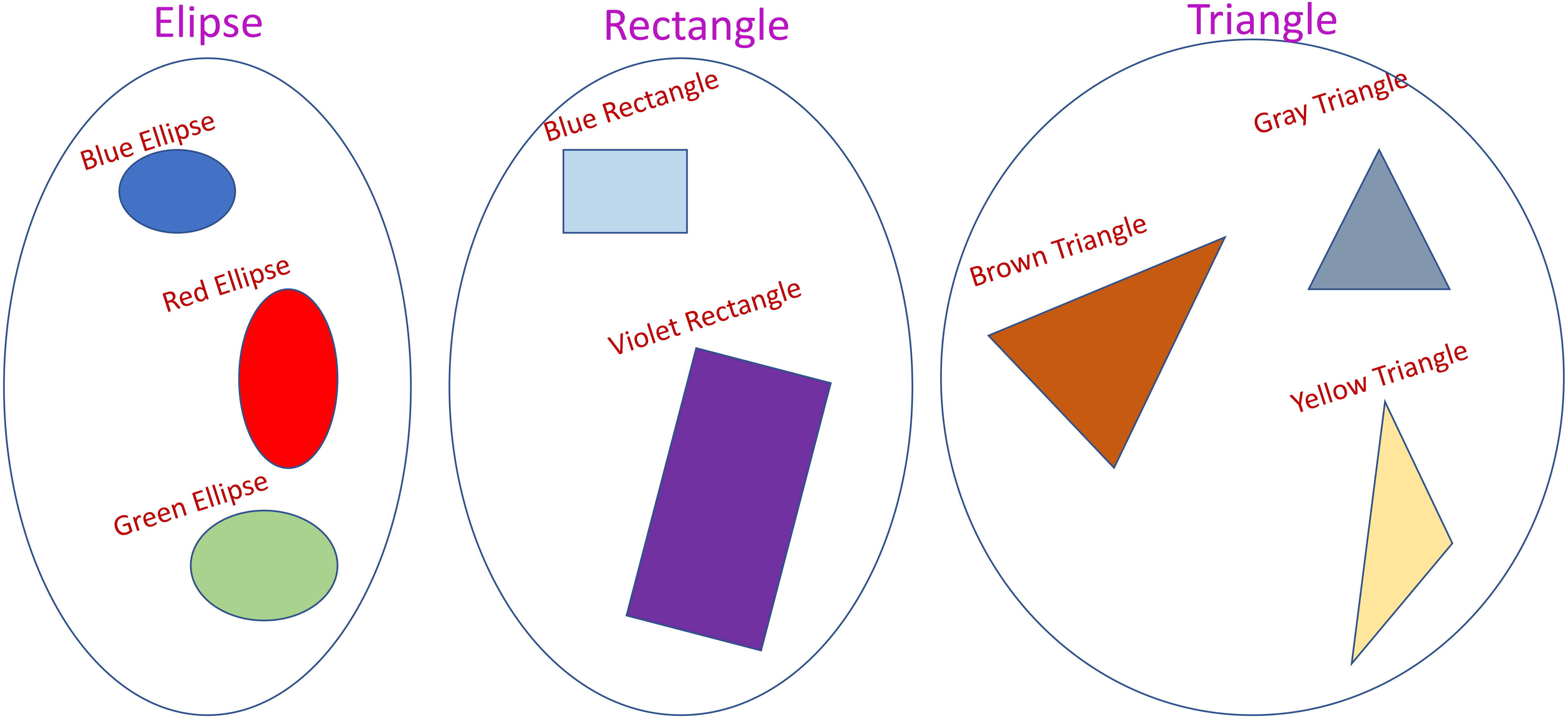
Inheritance

Which could be the properties of each class?



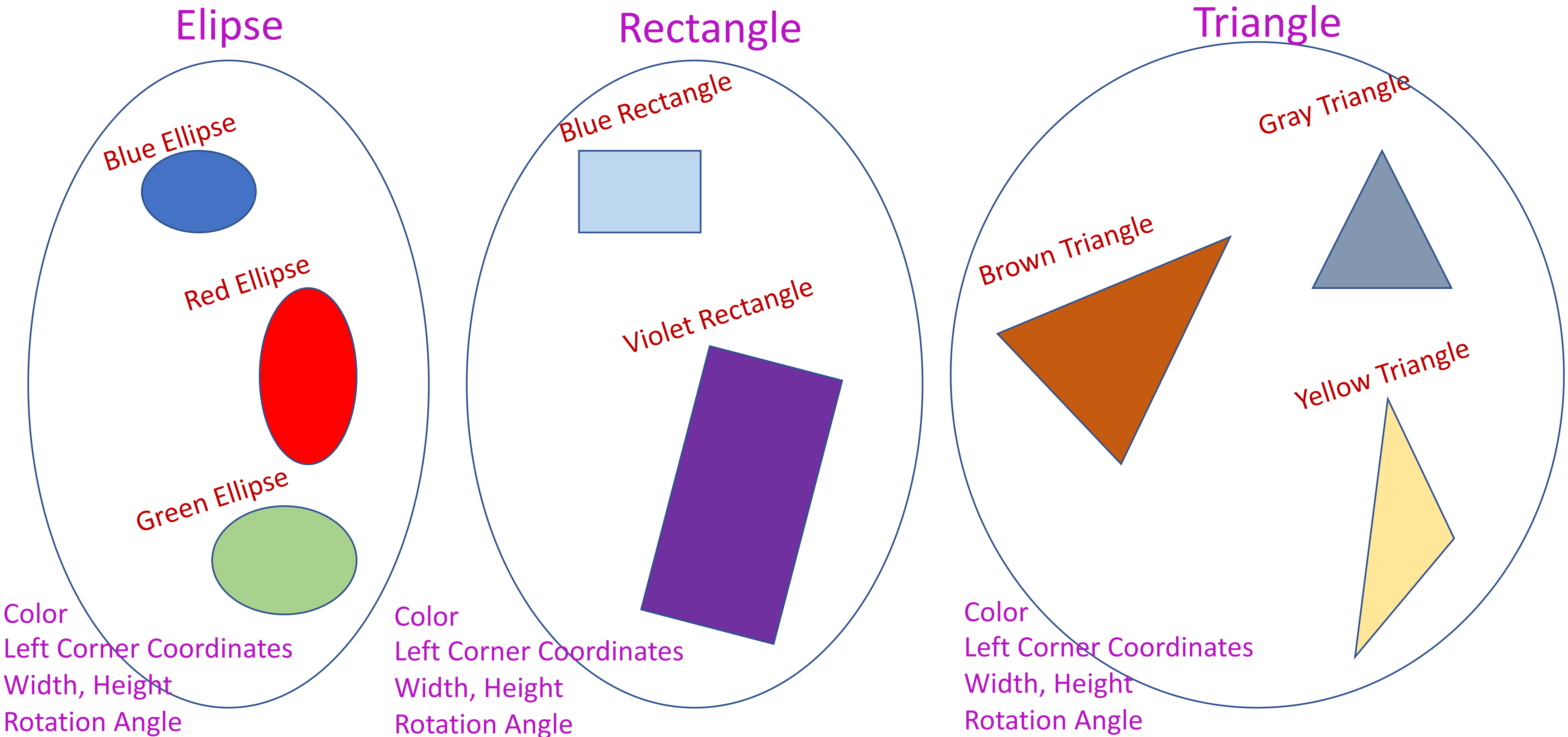
Inheritance

Which could be the properties of each class?



Inheritance

Which is the difference between the classes?



Inheritance

Create three classes or just one?

- In how many classes can be the objects grouped?
  - Elipse
  - Rectangles
  - Triangles
- Which could be the properties of each class?
  - Color
  - Left Corner Coordinates
  - Width
  - Height
  - Rotation Angle
- Which is the difference between the classes?
  - The way in which the figures are rendered

Inheritance. One class

- Creating only one class
  - How we discriminate between different types of figures?
    - Add an attribute
  - How we create objects?
    - Pass the figure type as parameter to `__init__` method
    - Define different functions for creating different types of objects

Disadvantages?
- Figure_type attribute can have any vakue
- Modify the class if new figure is added

```
class Figure(object):
    def __init__(self, figure_type, color, x, y, width, height, rotation_angle):
        self.figure_type = figure_type
        self.color = color
        # … the rest of the assignments here
    def draw(self):
        if self.figure_type == "Elipse":
            print("Eclipse drawing ...")
        elif self.figure_type == "Rectange":
            print("Rectange drawing ...")
        elif self.figure_type == "Triangle":
            print("Triangle drawing ...")
        else:
            print("? ...")
f = Figure("Elipse", "red", 10, 10, 100, 50, 0)
f.draw()
```

Inheritance. One class

- Creating only one class
  - How we discriminate between different types of figures?
    - Add an attribute
  - How we create objects?
    - Pass the figure type as parameter to `__init__` method
    - <u>Define different functions for creating different types of objects</u>

```
class Figure(object):
    def my_init(self, figure_type, color, x, y, width, height, rotation_angle):
        self.figure_type = figure_type
         # … the rest of the assignments here
    def create_eclipse(self, color, x, y, width, height, rotation_angle):
        self.my_init("Elipse", color, x, y, width, height, rotation_angle)
    def create_rectange(self, color, x, y, width, height, rotation_angle):
        self.my_init("Rectange", color, x, y, width, height, rotation_angle)
    def create_triangle(self, color, x, y, width, height, rotation_angle):
        self.my_init("Triangle", color, x, y, width, height, rotation_angle)
    def draw(self):
        if self.figure_type == "Elipse":
            print("Eclipse drawing ...")
        elif self.figure_type == "Rectange":
            print("Rectange drawing ...")
        elif self.figure_type == "Triangle":
            print("Triangle drawing ...")
f = Figure()
f.create_eclipse("red", 10, 10, 100, 50, 0)
f.draw()
```

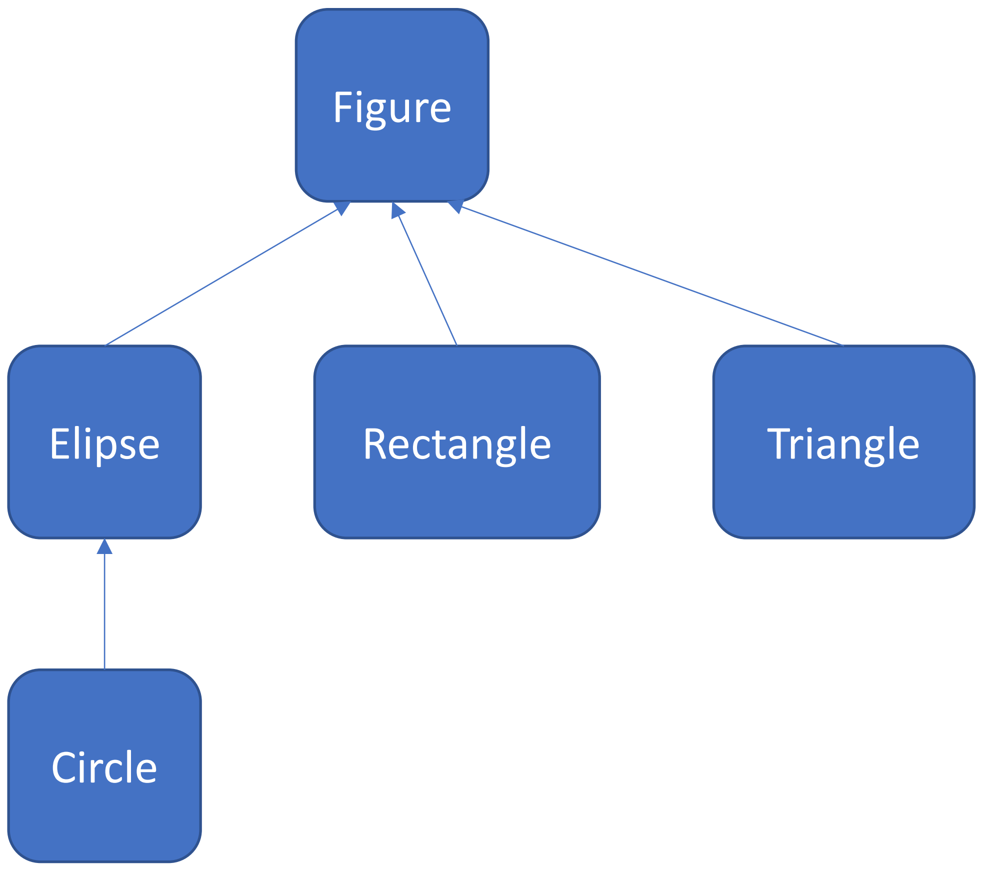
Disadvantages?

- Modify the class if new figure is added

Inheritance

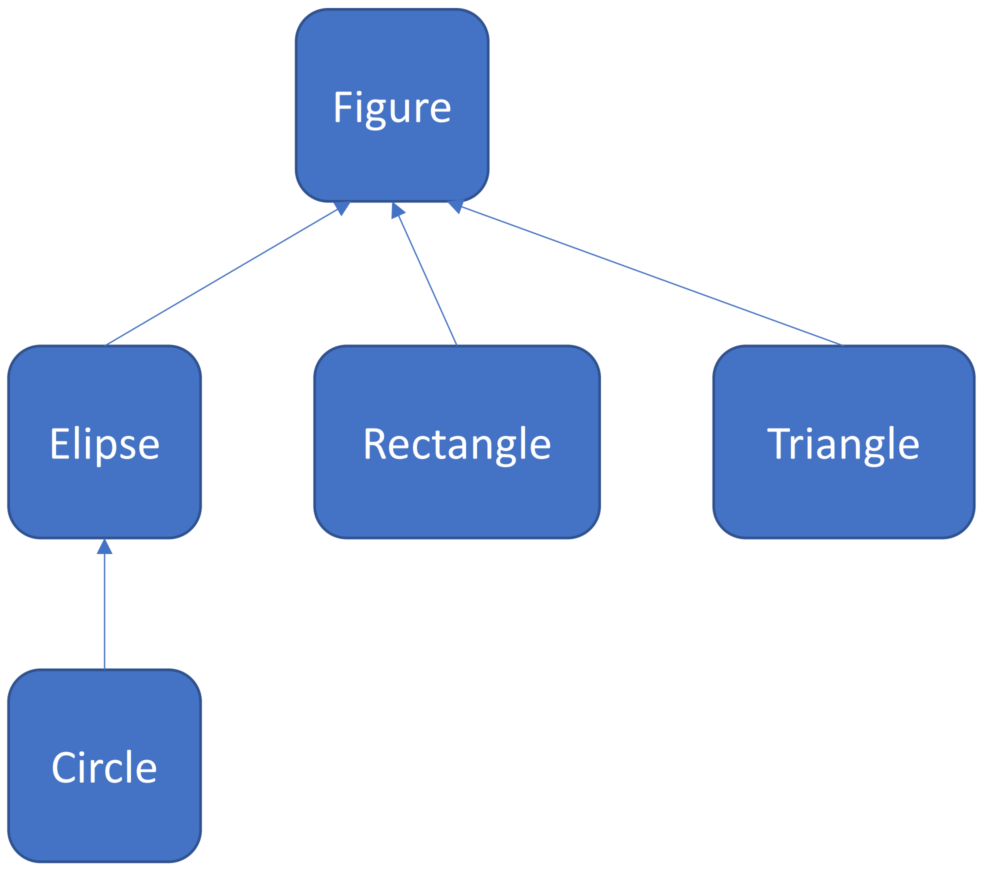
- One class disadvantages
  - Modify the class if new figure is added

- Inheritance
  - Allow to create new abstractization without modifying the existing ones



Inheritance

- Parent class (superclass)
- Child class (subclass)
  - Inherits all data and behaviors of parent class
  - Add more info
  - Add more behavior
  - Override behavior



Inheritance. Parent class

```
class Figure(object):
    def __init__(self, color, x, y, width, height, rotation_angle):
        self.color = color
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.rotation_angle = rotation_angle
    def __repr__(self):
        return "x={}, y={}, w={}, h={}, color={}, rotationAngle={}".format(
                self.x, self.y, self.width, self.height, self.color, self.rotation_angle)

f = Figure('red', 10, 10, 100, 50, 0)
print(f)
```

- Everything is an object
- Class `object` implements basic operations in Python, like binding variables, etc

Inheritance. Child class

```
class Ellipse(Figure):
    def draw(self):
        print("Elipse draw ...")

    def __repr__(self):
        return "Elipse " + super().__repr__()

f = Ellipse('red', 10, 10, 100, 50, 0)
f.draw()
print(f)
```

New functionality

Overrides __repr__

Access hidden implementation from the superclass

- Inherits all attribues of Figure:
  - __init__(), x, y, width, height, color, rotation_angle, __repr__()

- add new functionality with draw()
  - instance of type Elipse can be called with new methods
  - instance of type Figure throws error if called with Ellipse's new method
- __init__ is not missing, uses the Figure version

Which Methods to Use?

- Subclass can have methods with same name as superclass
- For an instance of a class, look for a method name in current class definition
- If not found, look for method name up the hierarchy (in parent, then grandparent, and so on)
- Use first method up the hierarchy that you found with that method name

More Subclasses

Parent class

Call constructor of parent class

```
class Circle(Elipse):
    def __init__(self, color, x, y, radius, filled):
        Ellipse.__init__(self, color, x, y, radius*2, radius*2, 0)
        self.filled = filled
    def __add__(self, other):
        self.width += other + other
        return self
    def __repr__(self):
        return  "Circle x={}, y={}, radius={}, color={}, filled={}".format(
                self.x, self.y, self.width//2,  self.color, self.filled)
f = Circle('red', 10, 10, 100, True)
f += 3
f.draw()
print(f)
```

New attribute

New method

Override parent method

Class Variables

- class variables and their values are shared between all instances of a class
  - `figure_nr` is used to create a unique ID for class instances

Class variable

```
class Figure(object):
    figure_nr =1
    def __init__(self, color, x, y, width, height, rotation_angle):
        self.color = color
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.rotation_angle = rotation_angle
        self.id = Figure.figure_nr
        Figure.figure_nr += 1
```

Class variable

instance variable

OO Programming

- Create your own collections of data
- Organize information
- Division of work
- Access information in a consistent manner
- Add layers of complexity
- Like functions, classes are a mechanism for decomposition and abstraction in programming

Bibliography

- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/lecture-slides-code/
- http://www.cs.toronto.edu/~quellan/courses/summer11/csc108/lectures.shtml