Outline

1. Object Oriented Programming
   - encapsulation, inheritance, polymorphism
   - objects in turtle
2. Two Examples
   - polynomials in one variable
   - points and circles

MCS 260 Lecture 28
Introduction to Computer Science
Jan Verschelde, 14 July 2023

encapsulation and inheritance
polymorphism and wrapping

1. Object Oriented Programming
   - encapsulation, inheritance, polymorphism
   - objects in turtle
2. Two Examples
   - polynomials in one variable
   - points and circles

Evolution of Computer Languages

We defined OOP as a new programming paradigm.
Learning a language is to learn idioms,
i.e.: how to express yourself properly.

1. procedure-oriented languages: C, FORTRAN 77
   Algorithms are central in program development.

   We use flowcharts or pseudo code to design programs.

2. object-oriented languages: Ada, C++, Smalltalk
   Objects belonging to ***classes*** organized in a ***hierarchy***
   are the main building blocks for programs.

   To design we use the Unified Modeling Language.

3. framework languages: Java, Python

   A ***framework*** is a collection of classes
   that provides a set of services for a given domain.

encapsulation, inheritance, and polymorphism

Definition (encapsulation)

Encapsulation of data represented by objects happens
by limiting the manipulation of the data to the methods of the class.

Definition (inheritance)

Inheritance is the derivation of a new class (child)
by inheriting attributes from other classes (parents).

Definition (polymorphism)

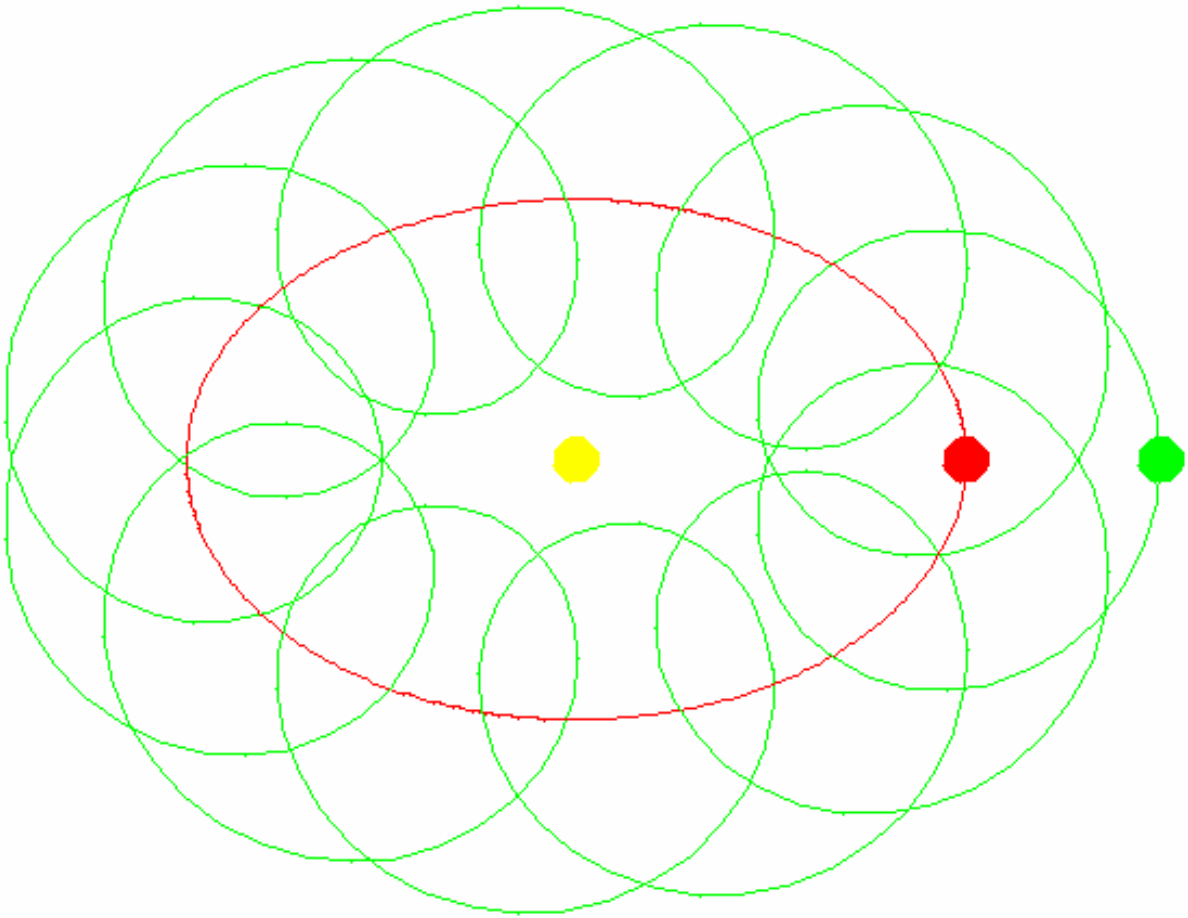
Polymorphism is realized via overloading or overriding
existing methods so they apply to objects of different classes.

encapsulation and inheritance
polymorphism and wrapping

1. Object Oriented Programming
   - encapsulation, inheritance, polymorphism
   - objects in turtle
2. Two Examples
   - polynomials in one variable
   - points and circles

Moon circles orbiting Earth

running `orbiting.py`



The sun is yellow, the earth is red, and the moon is green.

objects in turtle

Our sun-earth-moon simulation involves 3 turtles

The module `turtle` exports `Turtle` class:

```
>>> from turtle import Turtle
>>> sun = Turtle(shape='circle')
>>> sun.color('yellow')
>>> sun.position()
(0.00,0.00)
```

1. `sun` is an instance of the `Turtle` class, instantiated with value `'circle'` of `shape` attribute.
2. The method `color` changed the color of the object.
3. The method `position` returns the current position.

top down versus bottom up design

Our design of the sun-earth-moon simulation is ***top down***:

1. There is a main program that *controls* the action.
2. For each day, we tell the earth to move.

An object-oriented design would be ***bottom up***:

1. The objects earth and moon have `move()` methods.
2. Instead of calling `earth.move()`, and `moon.move()`,
   - one thread of execution would be assigned to `earth`, and
   - another thread of execution would be assigned to `moon`,

   so that both object move on their own pace.

The main program in a bottom up design launches threads.

encapsulation and inheritance
polymorphism and wrapping

1. Object Oriented Programming
   - encapsulation, inheritance, polymorphism
   - objects in turtle

2. Two Examples
   - polynomials in one variable
   - points and circles

Encapsulation

data hiding

Information hiding is important in modular design.

In object oriented programming,
we can hide the representation of an object.

Hidden data attributes are called *private*,
opposed to *public* (the default).

In Python, starting a name with _ _
makes a data attribute private.

Polynomials in One Variable

the need for data hiding

Problem: design a class to manipulate polynomials.

Representing $2x^8 - 3x^2 + 7$:

1. as coefficient vector $c = [7, 0, -3, 0, 0, 0, 0, 0, 2]$,
   $c[i]$ is coefficient of x^i;
2. as list of tuples $L = [(2, 8), (-3, 2), (7, 0)]$
   $(c, i) \in L$ represents cx^i.

Both representations have advantages and disadvantages.

Solution offered by *encapsulation*:

1. make representation of the object polynomial private;
2. access to the data only via specific methods.
3. resolve the problem of normalization, i.e.: `p == 0`?

Polymorphism

Recall Python's dynamic typing:
$\rightarrow$ during run time Python determines the type of a variable to know which methods may be applied.

- The string representation of a polynomial is defined by overloading `__str__`.
- By overloading `__call__`, an object is *callable*.

  Example: For a polynomial `p`, we can write `p(3)`, to evaluate the polynomial `p` at the input `3`.

encapsulation and inheritance
polymorphism and wrapping

1. Object Oriented Programming
   - encapsulation, inheritance, polymorphism
   - objects in turtle

2. Two Examples
   - polynomials in one variable
   - points and circles

Inheritance

base classes and derived classes

We can create new classes from existing classes.
These new classes are *derived* from *base* classes.

The derived class *inherits* the attributes of the base class and usually contains additional attributes.
Inheritance is a powerful mechanism to reuse software.
To control complexity, we add extra features later.

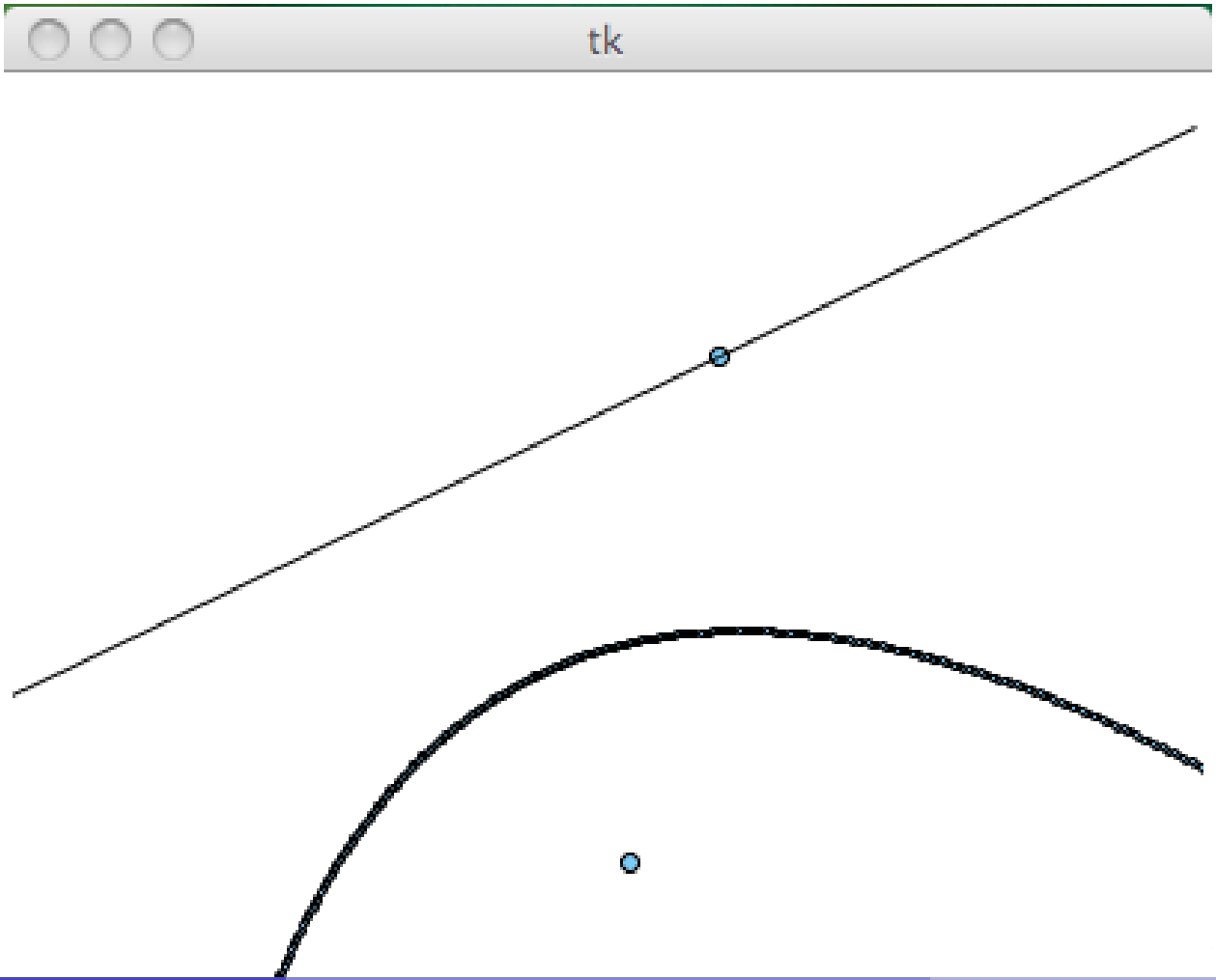
We distinguish between single and multiple inheritance:

- single: a derived class inherits from only *one* class;
- multiple: derivation from *multiple* different classes.

Multiple inheritance may lead to name clashes,
in case the parents have methods with same name.

an example of multiple inheritance

- A line inherits from a point,
  as a line is a point and a direction.
- A parabola has a focus and a directrix,
  so a parabola inherits from a point and a line.



Points and Circles

a first example of inheritance

We represent a point in the plane by the values for its coordinates, usually called x and y.

The class `Point` has attributes `x` and `y`
and a string representation.

A circle is determined by a center and radius.

The class `Circle` will inherit from the class `Point` to represent its center. The radius of the circle is an additional object data attribute. The function `area` is an additional functional attribute.

The class `Circle` will use the string representation of `Point` for its center and extend the `__str()__` function for its radius.

Exercises

1. An additional invariant to the representation of a polynomial in one variable is that its monomials are ordered along descending degree. Describe how you would implement this in the class `Poly`.
2. Derive a class `Triangle` from the class `Point`.
   The constructor should take three points as argument.
   Write an `area` function.
3. Write Python code to define a class Queue (FIFO).
   It should export `enqueue` and `dequeue`.
4. Give code so the robot draws a pentagon.
5. Extend the turn method of the robot class with the rotation angle.
   By default, the angle is $\pi/2$.

More Exercises

6. Change the `orbiting.py` program to simulate the sun-earth-moon orbits with a class `Planet`. Specify what data attributes are relevant to display the simulation. Add a functional attribute `Circulate` to the class `Planet`. Write a main program to run the simulation with multiple planets.
7. Extend the program of the previous exercise with a class `Satellite`. The class `Satellite` inherits from the class `Planet`. Write a main program to run the sun-earth-moon simulation with at least two moons.