Object-Oriented Programming in Python

1. Object-Oriented Programming
   - definition and an example
   - objects and classes, tkinter

2. Points, Lines, Parabolas
   - points in the plane
   - drawing points
   - representing lines in the plane
   - drawing lines
   - a parabola is defined by its focus and directrix
   - drawing a parabola

MCS 507 Lecture 3
Mathematical, Statistical and Scientific Software
Jan Verschelde, 25 August 2023

Object-Oriented Programming in Python

1. Object-Oriented Programming
   - definition and an example
   - objects and classes, tkinter

2. Points, Lines, Parabolas
   - points in the plane
   - drawing points
   - representing lines in the plane
   - drawing lines
   - a parabola is defined by its focus and directrix
   - drawing a parabola

object-oriented programming

Definition (Grady Booch et al., 2007)

Object-oriented programming is a method of implementation in which

1. programs are organized as cooperative collections of ***objects***,
2. each of which represents an instance of some ***class***,
3. and whose classes are all members of a ***hierarchy*** of classes united via inheritance relationships.

Objects — not algorithms — are the building blocks.

Algorithms are central in procedure-oriented programming.

Definition from page 41 on *Object-Oriented Analysis and Design With Applications* by G. Booch et al., Addison-Wesley, 2007.

an example from plane geometry

A ***point*** in the plane has two coordinates:

1. $x \in \mathbb{R}$, its projection onto the x-axis; and
2. $y \in \mathbb{R}$, its projection onto the y-axis.

A ***line*** in the plane has

1. a basis point, given by (x, y); and
2. a direction, given by an angle $\theta \in [0, 2\pi[$.

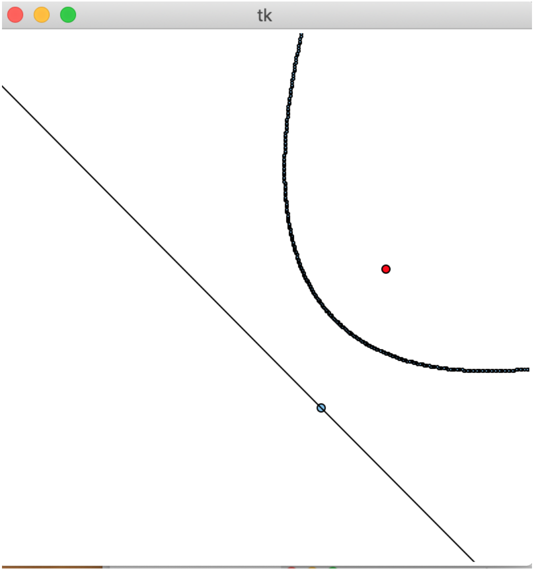
Then any point on the line is $(x + t\cos(\theta), y + t\sin(\theta))$, for $t \in \mathbb{R}$.

A ***parabola*** in the plane is the set of points

1. at equal distance from a point: the *focus*
2. at equal distance from a line: the *directrix*.

Thus we define a parabola by a line and a point, with five real numbers.

with tkinter



Object-Oriented Programming in Python

1. Object-Oriented Programming
   - definition and an example
   - objects and classes, tkinter

2. Points, Lines, Parabolas
   - points in the plane
   - drawing points
   - representing lines in the plane
   - drawing lines
   - a parabola is defined by its focus and directrix
   - drawing a parabola

objects and classes in Python

```
>>> help(range)
```

shows

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
...

>>> r = range(4)
>>> type(r)
<class 'range'>
>>> isinstance(r, range)
True
```

The object `r` is *an instance of* the class `range`.

tkinter, Tk, and Tcl

GUIs for Python programmers

Platform independent Graphical User Interfaces (GUIs),
i.e.: same code runs on Windows, Mac OS X, Linux computers.

The `tkinter` (= `Tk` interface) library provides
an object-oriented interface to the `Tk` GUI toolkit,
the graphical interface development tool for `Tcl`,

`Tk` = Tool Kit, `Tcl` = Tool Command Language.

Object-Oriented Programming in Python

storing coordinates of points

```
class Point(object):
    """
    Stores a point in the plane.
    """
    def __init__(self, x=0, y=0):
        """
        Defines the coordinates.
        """
        self.xpt = x
        self.ypt = y
```

- The `__init__()` defines the *constructor* method, for objects of the class Point.
- The coordinates are the *data attributes* of the class, by default, if unspecified, both coordinates are zero.
- The parameter `self` of `__init__()` is the instance itself, the variable to which the constructor method is applied.

to print points, define the string representation

For a point `p`, `print(p)` shows the string representation.

```
def __str__(self):
    """
    Returns the string representation.
    """
    return '(' + str(self.xpt) \
               + ', ' \
               + str(self.ypt) + ')'
```

The definition looks recursive, as we call `str()` on the coordinates.

If the coordinate is an integer (or a double), then
the string representation method on integers (or on doubles) is called.

the representation of a point

The class definition is stored in the file `class_point.py`.

```
>>> from class_point import Point
>>> p = Point(y=2, x=1)
>>> print(p)
(1, 2)
>>> p
(1, 2)
```

The effect of the last statement is determined by the representation.

```
    def __repr__(self):
        """
        Returns the representation.
        """
        return self.__str__() # same as string rep
```

a test program

In the file with the class definition:

```
def main():
    """
    Instantiates two points.
    """
    print('instantiating two points ...')
    first = Point()
    print('p =', first)
    second = Point(3, 4)
    print('q =', second)

if __name__ == "__main__":
    main()
```

use `class_point.py` as a script or a module

The last two lines of `class_point.py` are

```
if __name__ == "__main__":
    main()
```

which allows

- to run the script `class_point.py`, at the prompt `$`:

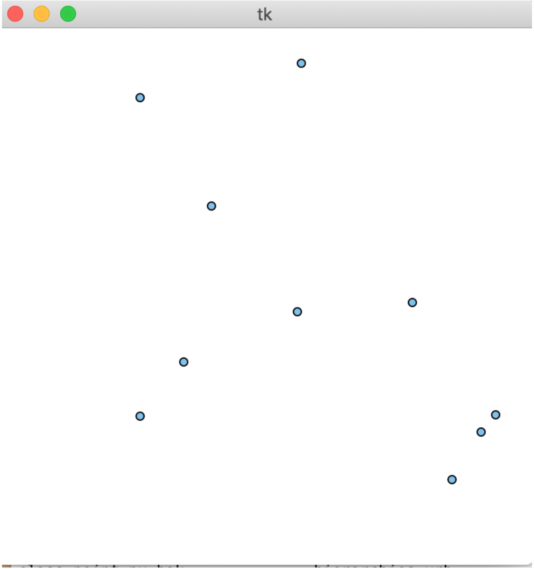
  ```
  $ python class_point.py
  ```

  which will execute the function `main()`.
- or in an interactive Python shell do

  ```
  >>> from class_point import Point
  >>> p = Point(3, 4)
  >>> print p
  (3, 4)
  ```

Object-Oriented Programming in Python

drawing ten random points on canvas



extending the class `Point`

To draw a point, we will extend the class `Point` with

- a data attribute: a canvas; and
- a method: to draw the point on canvas.

Our new script `class_showpoint.py` starts with

```
from tkinter import Tk, Canvas
from random import randint
from class_point import Point
```

inheriting from `Point`

```
class ShowPoint(Point):
    """
    Extends the class Point
    with a draw method on a Tkinter Canvas.
    """
    def __init__(self, cnv, x=0, y=0):
        """
        Defines the coordinates (x, y)
        and stores the canvas cnv.
        """
        Point.__init__(self, x, y)
        self.canvas = cnv
```

- The class `ShowPoint` *inherits* from the class `Point`.
- The coordinates `x` and `y` of the constructor are passed to the constructor of the class `Point`.
- The added data attributed is called `canvas`.

the method `draw()`

An instance of the class `ShowPoint` has three data attributes:

1. `xpt`: the *x*-coordinate of the point;
2. `ypt`: the *y*-coordinate of the point; and
3. `canvas`: an instance of a tkinter Canvas.

```
def draw(self):
    """
    Draws the point on canvas.
    """
    (xpt, ypt) = (self.xpt, self.ypt)
    self.canvas.create_oval(xpt-3, ypt-3, \
        xpt+3, ypt+3, fill='SkyBlue2')
```

The `draw`() method defines the appearance of the point on a canvas, using the `create_oval()` method.

the main program in the module `class_showpoint`

```
def main():
    """
    Shows 10 random points on canvas.
    """
    top = Tk() # gets a window
    dim = 400  # dimension of the window
    cnv = Canvas(top, width=dim, height=dim)
    cnv.pack() # default positioning in window
    points = []
    for _ in range(10):
        xrd = randint(6, dim-6)
        yrd = randint(6, dim-6)
        points.append(ShowPoint(cnv, xrd, yrd))
    for point in points:
        point.draw()
    top.mainloop() # launches the main event loop
```

Object-Oriented Programming in Python

1. Object-Oriented Programming
   - definition and an example
   - objects and classes, tkinter

2. Points, Lines, Parabolas
   - points in the plane
   - drawing points
   - representing lines in the plane
   - drawing lines
   - a parabola is defined by its focus and directrix
   - drawing a parabola

the class `Line` inherits from `Point`

For $t \in \mathbb{R}$, a point on a line is $(x + t\cos(\theta), y + t\sin(\theta))$, where

- x and y are the coordinates of the basis point; and
- the angle θ is the direction of the line.

```
from math import cos, sin
from class_point import Point

class Line(Point):
    """
    A line is a base point and
    a direction angle.
    """
    def __init__(self, x=0, y=0, a=0):
        """
        Defines base point and angle.
        """
        Point.__init__(self, x, y)
        self.angle = a
```

implications of inheritance

```
>>> from class_point import Point
>>> from class_line import Line
>>> L = Line(3, 4, 1.23)
>>> isinstance(L, Line)
True
>>> isinstance(L, Point)
True
```

Because of the inheritance, the line `L` is also an instance of the class `Point`.

We view a line as an extension of a point.

the string representation

We *override* the string representation for `Line`,
using the string representation defined in `Point`:

```
def __str__(self):
    """
    Returns the string representation.
    """
    strp = Point.__str__(self)
    stra = ', angle = ' + str(self.angle)
    return strp + stra
```

As we did in the definition of `Point`,
the representation of a line is set to its string representation.

a line as a function

Instances of the class `Line` become *callable* with

```
def __call__(self, argt):
    """
    Returns a new point on the line.
    """
    rxt = self.xpt + argt*cos(self.angle)
    ryt = self.ypt + argt*sin(self.angle)
    return Point(rxt, ryt)
```

The class definition is stored in `class_line.py`.

```
>>> from class_line import Line
>>> from math import pi
>>> L = Line(1,2,pi/3)
>>> L
(1, 2), angle = 1.0471975511965976
>>> L(4)
(3.0000000000000004, 5.464101615137754)
```

testing the class

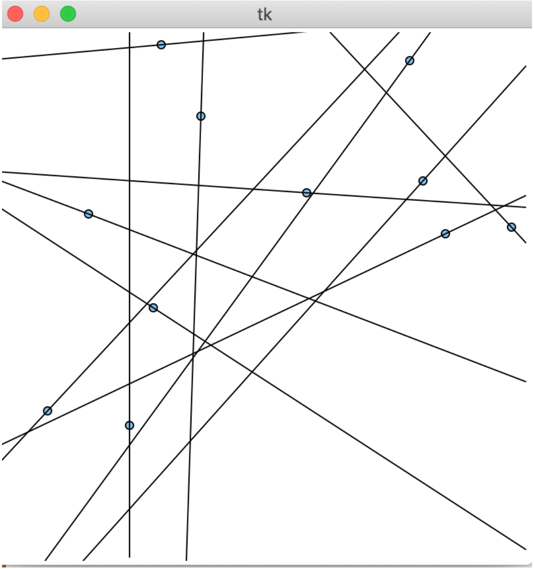
```
def main():
    """
    Defining a line.
    """
    print('defining a line ...')
    first = Line()
    print('the default line :', first)
    print('some point on it :', first(5))
    from math import pi
    second = Line(3, 4, pi/2)
    print('a vertical line :', second)
    print('   that passes through', second(0))
    apt = second(4)
    print('another point on it is', apt)

if __name__ == "__main__":
    main()
```

Object-Oriented Programming in Python

1. Object-Oriented Programming
   - definition and an example
   - objects and classes, tkinter

2. Points, Lines, Parabolas
   - points in the plane
   - drawing points
   - representing lines in the plane
   - drawing lines
   - a parabola is defined by its focus and directrix
   - drawing a parabola

eleven lines



extending the `Line` class

```
from class_line import Line
from class_showpoint import ShowPoint

class ShowLine(Line, ShowPoint):
    """
    Extends the Line class with a draw method.
    """
    def __init__(self, c, d, x=0, y=0, a=0):
        """
        Defines the line through the point
        (x, y), with angle a and stores the
        canvas c of dimension d.
        """
        Line.__init__(self, x, y, a)
        ShowPoint.__init__(self, c, x, y)
        self.dimension = d
```

multiple inheritance

- In the class definition:

  ```
  class ShowLine(Line, ShowPoint):
  ```

  The class `ShowLine` inherits from `Line` and `ShowPoint`.
- In the constructor:

  ```
          Line.__init__(self, x, y, a)
          ShowPoint.__init__(self, c, x, y)
          self.dimension = d
  ```

  we instantiate the superclasses `Line` and `ShowPoint`
  and add one extra data attribute, the dimension of the canvas.

Inheriting from `ShowPoint`,
the drawing of the base point of the line is already defined.

computing the range of the line

The extension of the class stores the canvas
and the dimension of the canvas so we may
compute the part of the line that fits on canvas.

The base point is (x, y) and we want to compute the point
$x + t\cos(\theta) = d$, where d is the right edge of the canvas.

$$\Rightarrow t = \frac{d - x}{\cos(\theta)}$$

One special case: a vertical line.

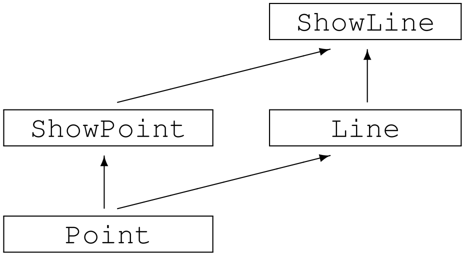
drawing a line

```
def draw(self):
    """
    Draws the line on canvas.
    """
    ShowPoint.draw(self)
    cnv = self.canvas
    (xpt, ypt) = (self.xpt, self.ypt)
    dmx = self.dimension - xpt
    csa = cos(self.angle)
    if csa + 1.0 != 1.0:
        pt1 = Line.__call__(self, dmx/csa)
        cnv.create_line(xpt, ypt, pt1.xpt, pt1.ypt)
        pt2 = Line.__call__(self, -xpt/csa)
        cnv.create_line(pt2.xpt, pt2.ypt, xpt, ypt)
    else: # vertical line
        cnv.create_line(xpt, 0, xpt, self.dimension)
```

the test program

```
def main():
    """
    Shows 11 lines on canvas.
    """
    top = Tk()
    dim = 400
    cnv = Canvas(top, width=dim, height=dim)
    cnv.pack()
    lines = [ShowLine(cnv, dim, 100, 300, pi/2)]
    for _ in range(10):
        (xrd, yrd) = (randint(6, dim-6), randint(6, dim-6))
        rnd = uniform(0, 2*pi)
        lines.append(ShowLine(cnv, dim, xrd, yrd, rnd))
    for line in lines:
        line.draw()
    top.mainloop()
```

bottom up programming



Object-Oriented Programming in Python

1. Object-Oriented Programming
   - definition and an example
   - objects and classes, tkinter

2. Points, Lines, Parabolas
   - points in the plane
   - drawing points
   - representing lines in the plane
   - drawing lines
   - a parabola is defined by its focus and directrix
   - drawing a parabola

inheriting from `Point` and `Line`

A parabola is the set of points

- at equal distance from a point: the *focus*
- at equal distance from a line: the *directrix*.

```
from class_point import Point
from class_line import Line

class Parabola(Line):
    """
    A parabola is defined by a line,
    its directrix and a point, its focus.
    """
    def __init__(self, px, py, agl, fx, fy):
        """
        Defines the line at (px, py)
        and angle agl and focus (fx, fy).
        """
        Line.__init__(self, px, py, agl)
        self.focus = Point(fx, fy)
```

is instance versus *has a*

```
class Parabola(Line):
    def __init__(self, px, py, agl, fx, fy):
        Line.__init__(self, px, py, agl)
        self.focus = Point(fx, fy)
```

An instance of a Parabola

- is an instance of a Line,
- has an instance of a Point as data attribute.

the string representation

```
def __str__(self):
    """
    Returns the string representation.
    """
    strfocus = str(self.focus)
    line = Line.__str__(self)
    result = 'focus : ' + strfocus
    result += ', directrix : ' + line
    return result
```

computing points on a parabola

We compute points on the parabola, running along the directrix $(c_x, c_y) = (x + t\cos(\theta), y + t\sin(\theta))$.

Let the focus have coordinates (f_x, f_y).

The point (α, β) on the parabola at t satisfies

$$(\alpha - c_x)^2 + (\beta - c_y)^2 = (\alpha - f_x)^2 + (\beta - f_y)^2$$

and the line spanned by (α, β) and (c_x, c_y) is orthogonal to the line spanned by (x, y) and (c_x, c_y).

This leads to a linear system in α and β.

deriving the linear system in α and β

Expanding both sides of

$$(\alpha - c_x)^2 + (\beta - c_y)^2 = (\alpha - f_x)^2 + (\beta - f_y)^2$$

leads to

$$\begin{array}{rcl}(\alpha - c_x)^2 + (\beta - c_y)^2 & = & \alpha^2 - 2\alpha c_x + c_x^2 + \beta^2 - 2\beta c_y + c_y^2 \\ (\alpha - f_x)^2 + (\beta - f_y)^2 & = & \alpha^2 - 2\alpha f_x + f_x^2 + \beta^2 - 2\beta f_y + f_y^2.\end{array}$$

Subtracting the equations eliminates α^2 and β^2:

$$2(f_x - c_x)\alpha + 2(f_y - c_y)\beta = f_x^2 + f_y^2 - c_x^2 - c_y^2.$$

The other linear equation: $(\alpha - c_x, \beta - c_y) \perp (x - c_x, y - c_y)$,
with $(c_x, c_y) = (x + t\cos(\theta), y + t\sin(\theta))$.

the linear system

Perpendicularity $\perp$ is calculated with a dot product:

$$\begin{aligned} & (\alpha - c_x, \beta - c_y) \perp (x - c_x, y - c_y) \\ \Leftrightarrow \quad & (\alpha - c_x)(x - c_x) + (\beta - c_y)(y - c_y) = 0 \\ \Leftrightarrow \quad & (x - c_x)\alpha + (y - c_y)\beta = c_x(x - c_x) + c_y(y - c_y). \end{aligned}$$

Recall that $2(f_x - c_x)\alpha + 2(f_y - c_y)\beta = f_x^2 + f_y^2 - c_x^2 - c_y^2$.

In matrix form:

$$\left[\begin{array}{cc} 2(f_x - c_x) & 2(f_y - c_y) \\ x - c_x & y - c_y \end{array} \right] \left[\begin{array}{c} \alpha \\ \beta \end{array} \right] = \left[\begin{array}{c} f_x^2 + f_y^2 - c_x^2 - c_y^2 \\ c_x(x - c_x) + c_y(y - c_y) \end{array} \right].$$

With Cramer's rule we find explicit formulas for α and β.

making a parabola callable

```
def __call__(self, t):
    """
    Returns a point on the parabola
    as far from the focus and the point
    obtained by evaluating the directrix.
    """
    line = Line.__call__(self, t)
    (fcx, fcy) = (self.focus.xpt, self.focus.ypt)
    disc = 2*(fcx - line.xpt)*(line.ypt - self.ypt) \
         - 2*(fcy - line.ypt)*(line.xpt - self.xpt)
    rh1 = fcx**2 + fcy**2 - line.xpt**2 - line.ypt**2
```

The discriminant `disc` is the determinant of

$$\left[\begin{array}{cc} 2(f_x - c_x) & 2(f_y - c_y) \\ x - c_x & y - c_y \end{array}\right].$$

definition continued

```
if disc + 1.0 != 1.0:
    rh2 = line.xpt*(line.xpt - self.xpt) \
        + line.ypt*(line.ypt - self.ypt)
    sdx = (rh1*(line.ypt - self.ypt) \
        - 2*rh2*(fcy - line.ypt))/float(disc)
    sdy = (2*(fcx - line.xpt)*rh2 \
        - (line.xpt - self.xpt)*rh1)/float(disc)
else:
    from math import exp, pi
    line = Line.__call__(self, exp(pi))
    disc = 2*(fcx - line.xpt)*(line.ypt - self.ypt) \
         - 2*(fcy - line.ypt)*(line.xpt - self.xpt)
    rh1 = fcx**2 + fcy**2 - line.xpt**2 - line.ypt**2
    rh2 = line.xpt*(line.xpt - self.xpt) \
        + line.ypt*(line.ypt - self.ypt)
    sdx = (rh1*(line.ypt - self.ypt) \
        - 2*rh2*(fcy - line.ypt))/float(disc)
    sdy = (2*(fcx - line.xpt)*rh2 \
        - (line.xpt - self.xpt)*rh1)/float(disc)
return Point(sdx, sdy)
```

the test function

```
def main():
    """
    Instantiates a parabola and evaluates.
    """
    print('instantiating a parabola ...')
    prb = Parabola(3, 4, -1.23, 10, 0)
    print(prb)
    point = prb(4)
    print('a point on the parabola :', point)

if __name__ == "__main__":
    main()
```

Object-Oriented Programming in Python

1. Object-Oriented Programming
   - definition and an example
   - objects and classes, tkinter

2. Points, Lines, Parabolas
   - points in the plane
   - drawing points
   - representing lines in the plane
   - drawing lines
   - a parabola is defined by its focus and directrix
   - drawing a parabola

inheriting from `Parabola` and `Showline`

```
from Tkinter import Tk, Canvas
from math import pi
from random import randint, uniform
from class_parabola import Parabola
from class_showline import ShowLine

class ShowParabola(Parabola, ShowLine):
    """
    Extends the Parabola class with a draw method.
    """
    def __init__(self, c, d, x, y, a, fx, fy):
        """
        Defines the parabola with directrix the line
        at base point (x, y), angle a, and the focus
        with coordinates (fx, fy).
        Stores the canvas c of dimension d.
        """
        Parabola.__init__(self, x, y, a, fx, fy)
        ShowLine.__init__(self, c, d, x, y, a)
```

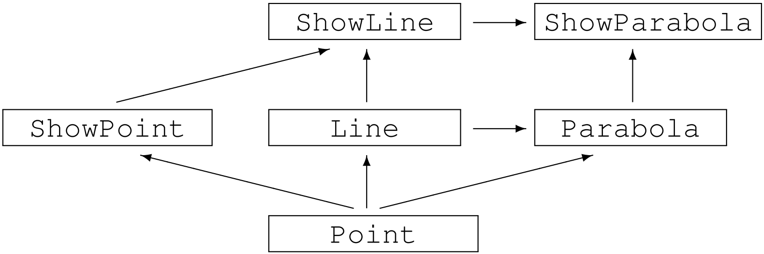
drawing the parabola

```
def draw_focus(self):
    """
    Draws the focus on canvas.
    """
    (fxp, fyp) = (self.focus.xpt, self.focus.ypt)
    self.canvas.create_oval(fxp-3, \
        fyp-3, fxp+3, fyp+3, fill='red')

def draw(self):
    """
    Draws the parabola on canvas.
    """
    ShowLine.draw(self)
    self.draw_focus()
    for tpt in xrange(-1000, 1000):
        prb = Parabola.__call__(self, tpt)
        self.canvas.create_oval(prb.xpt-1, \
            prb.ypt-1, prb.xpt+1, prb.ypt+1, \
            fill='SkyBlue2')
```

the main program

```
def main():
    """
    Draws one parabola.
    """
    top = Tk()
    dim = 400
    cnv = Canvas(top, width=dim, height=dim)
    cnv.pack()
    (xbs, ybs) = (randint(6, dim-6), randint(6, dim-6))
    agl = uniform(0, 2*pi)
    (fpx, fpy) = (randint(6, dim-6), randint(6, dim-6))
    pbl = ShowParabola(cnv, dim, xbs, ybs, agl, fpx, fpy)
    pbl.draw()
    top.mainloop()
```

bottom up programming



Summary + Exercises

Object-oriented programming applies a *bottom up* order of developing programs, opposed to a top down order.

Exercises:

1. A line in the plane is defined algebraically by the equation $ax + by = c$, for $a, b, c \in \mathbb{R}$.
   - Inherit from `Line` to define a class `AlgebraicLine`.
   - The string representation of an object of `AlgebraicLine` shows an equation. Define a method `slope()` in the class `AlgebraicLine`, which returns the slope of the line.
2. A circle with center p and radius r is the set of all points at distance r from the point p.
   - Define a class `Circle`, inheriting from the class `Point`.
   - Make objects of this class callable.
   - Define a method `area`, which returns the area of the circle.
   - Inheriting from `Circle`, define `ShowCircle` which extends this class with a tkinter Canvas object and with a method to draw the circle on canvas.