graphics

Objects:

Window:
An object to control window settings, such as size, background, etc. Also provides functionality for obtaining mouse coordinates.

Shapes:
The included shapes include: Rectangle, Oval, Polygon, Text

API:

Blue parameters are required, pink are optional

Window:

● Window(title, width, height): Sets up a Window of the given title (a string, eg. “My Screen”), width, and height, or creates an object to interact with the current Window if one has already been created and applies the parameters accordingly.

```python
window = Window("My Window", 600, 600)
```

● setBackground(color): Sets the background to the given color. The parameter color can be a string (eg. “red”), or a tuple corresponding to RGB values (eg. (255, 0, 0)).

```python
window = Window("My Window", 600, 600)
window.setBackground("Blue")
```

● checkMouse(): Returns last mouse click or None if mouse has not been clicked since the last call.

```python
window = Window("My Window", 600, 600)
window.checkMouse()
```

● getMouse(): Pauses until the screen is clicked, then returns the coordinates of the click.

```python
window = Window("My Window", 600, 600)
window.getMouse()
```
- **close()**: Closes the window.

  ```python
  window = Window("My Window", 600, 600)
  window.close()
  ```

- **update()**: Issues a window update. If things have been moved/rotated/etc, but the screen has not updated yet, this will reflect those changes.

  ```python
  window = Window("My Window", 600, 600)
  window.update()
  ```

**Shapes**: (these methods work for all shapes)
- **draw(update)**: Draws the given shape to the Window. By default, draw will issue a `Window.update()` command - if `update` is False, then it will not update the window.

  ```python
  square = Rectangle((-50, -50), 100, 100)
  square.draw()
  ```

- **undraw(update)**: Undraws the given shape from the Window. By default, undraw will issue a `Window.update()` command - if `update` is False, then it will not update the window.

  ```python
  square = Rectangle((-50, -50), 100, 100)
  square.draw()
  square.undraw()
  ```

- **move(dx, dy, update)**: Moves the given shape on the window `dx` units to the right (or `|dx|` units to the left, if negative) and `dy` units upward (or `|dy|` units downward, if negative). By default, move will issue a `Window.update()` command - if `update` is False, then it will not update the window.

  ```python
  square = Rectangle((-50, -50), 100, 100)
  square.draw()
  square.move(50, 50)
  ```

- **rotate(dTheta, update)**: Rotates the given shape `dTheta` degrees counterclockwise (or `|dTheta|` degree clockwise if negative) about the
shape's center. By default, rotate will issue a Window.update() command - if update is False, then it will not update the window.

```python
square = Rectangle((-50, -50), 100, 100)
square.draw()
square.rotate(30)
```

- **containsPoint(point)**: Returns True if the given point is contained within the boundary of the shape, otherwise returns False.

```python
square = Rectangle((-50, -50), 100, 100)
square.draw()

# Prints True
print(square.containsPoint((0, 0))

# Prints False
print(square.containsPoint((-100, -100))
```

- **setLabel(text, size, font, color, offset)**: Attaches a Text object to the shape at an offset offset (eg. (3, 7) from the center. The optional parameters size, font, and color can be used to control the font.

```python
square = Rectangle((-50, -50), 100, 100)
square.draw()
square.setLabel("My Label", 12, "Helvetica", "Red", (0, -8))
```

- **getLabelText()**: Returns the string text of the label

```python
square = Rectangle((-50, -50), 100, 100)
square.draw()
square.setLabel("My Label")
print(square.getLabelText())
```

- **removeLabel()**: Removes the label from the shape.

```python
square = Rectangle((-50, -50), 100, 100)
square.draw()
square.setLabel("My Label")
```
square.removeLabel()

- **setFill**(color): Configures the given shape to be filled with the given parameter color.

  square = Rectangle((-50, -50), 100, 100)
square.draw()
square.setFill("Red")

- **setLineColor**(color): Configures the given shape to have lines colored by the given color.

  square = Rectangle((-50, -50), 100, 100)
square.draw()
square.setLineColor("Red")

- **setLineWidth**(width): Configures the given shape to have lines of the given width.

  square = Rectangle((-50, -50), 100, 100)
square.draw()
square.setLineWidth(2)

- **getPoints**(): Returns the list of points defining this shape.

  square = Rectangle((-50, -50), 100, 100)
print(square.getPoints())

- **getCenter**(): Returns the centroid of the shape.

  square = Rectangle((-50, -50), 100, 100)
print(square.getCenter())

**Rectangle**: (in addition to the methods listed for Shapes)

- **Rectangle**(lowerLeftCorner, width, height, precision, lineWidth, lineColor, fillColor): Creates a Rectangle object of the given width and height where lowerLeftCorner is a tuple (eg. (100, 50)) designating the lower left corner of the Rectangle. The optional parameters lineWidth, lineColor, and fillColor control the line
width, line color, and fill color of the Rectangle. The parameter
**precision** controls the number of decimal points used in managing the
shape; by default, it is 0.

```
square = Rectangle((-50, -50), 100, 100)
```

**Oval**: (in addition to the methods listed for Shapes)

- **Oval**(center, width, height, steps, precision, lineWidth, lineColor, fillColor): Creates an Oval object of the
given width and height where center is a tuple (eg. (100, 50))
designating the center of the Oval. The optional parameter **steps**
controls how many points will be used to create the outline of shape; by
default, this value is 500. The remaining optional parameters **lineWidth**, **lineColor**, and **fillColor** control the line width, line color, and fill
color of the Oval. The parameter **precision** controls the number of
decimal points used in managing the shape; by default, it is 0.

```
circle = Oval((0, 0), 100, 100, 500)
```

**Polygon**: (in addition to the methods listed for Shapes)

- **Polygon**(points, precision, lineWidth, lineColor, fillColor): Creates a Polygon object using the given list points (eg.
  [(0, 0), (100, 0), (100, 100), (0, 100)]). Note that the list **points** will be
  iterated through as given, so order matters. The optional parameters
  **lineWidth**, **lineColor**, and **fillColor** control the line width, line
  color, and fill color of the Polygon. The parameter **precision** controls the number of
decimal points used in managing the shape; by default, it is 0.

```
import math
listOfPoints = []
for i in range(5):
    listOfPoints.append((
        50*math.cos(i * 2 * math.pi / 5),
        50*math.sin(i * 2 * math.pi / 5)))
pentagon = Polygon(listOfPoints)
```

**Line**: (in addition to the methods listed for Shapes)
- **Line(point1, point2, precision, lineWidth, lineColor)**: Creates a Line object using the given points point1 and point2. The optional parameters lineWidth and lineColor control the line width, line color, and fill color of the Rectangle. The parameter precision controls the number of decimal points used in managing the shape; by default, it is 0.

```python
line = Line((0, 0), (100, 100))
```

**Text**: (in addition to the methods listed for Shapes)
- **Text(text, pos, size, font, fontColor)**: Creates a Text object using the given string text, centered at the tuple pos, where size, font, and fontColor configure the font of the text. Many functions for Shapes are disabled for Text objects (including rotate).

```python
text = Text("Some text", (0, 0), 14, "Arial", "Blue")
```

**Image**: (in addition to the methods listed for Shapes)
- **Image(lowerLeftCorner, fileName, width, height)**: Creates an Image object anchoring its bottom left to the point given by lowerLeftCorner. Note that either fileName must be provided, or a width and height must be given. If fileName is provided, it attempts to open the file of that name. Otherwise, it creates a blank Image using the dimensions given by width and height. Note that Python natively supports only the following formats: .gif, .pgm, .ppm, and .png. The standard rotate function is disabled for Images. All functions that create changes to the window force updates.

```python
img1 = Image((0, 0), 100, 100)
Image2 = Image((0, 0), "someLocalFile.png")
```

- **getWidth()**: Returns the width of the Image.

```python
img = Image((0, 0), 100, 100)
print(img.getWidth())
```

- **getHeight()**: Returns the height of the Image.

```python
img = Image((0, 0), 100, 100)
```
print(img.getHeight())

- **getPixel** \((x, \ y)\): Returns a tuple \((r, g, b)\) with the RGB color values for pixel \((x, \ y)\) \(r, g, b\) are in range(256). Note that \(x\) and \(y\) are taken with respect to the lower left coordinate.

```python
img = Image((0, 0), 100, 100)
pixel = img.getPixel(50, 50)
```

- **setPixel** \((x, \ y, \text{color})\): Sets the pixel determined by \(x\) and \(y\) to the given \text{color}. Note that \(x\) and \(y\) are taken with respect to the lower left coordinate.

```python
img = Image((0, 0), 100, 100)
img.setPixel(50, 50, (255, 0, 0))
img.setPixel(25, 12, "Pink")
```

- **save** \(\text{fileName}\): Saves this Image to a file with the given \text{fileName}. The extension of \text{fileName} determines its format.

```python
img = Image((0, 0), 100, 100)
img.save("newFile.png")
```