Contents:

1 Introduction to Pygame
  1.1 Import the module ............................................. 3
  1.2 Show the event loop ........................................... 4
  1.3 Quit the event loop properly ................................. 4
  1.4 Define colors .................................................. 5
  1.5 Switch the background color ................................. 6
  1.6 Import pygame.locals ......................................... 7
  1.7 Use a dictionary to decode keys ............................. 7
  1.8 Change the window caption .................................. 8
  1.9 Explore a simple ball game ................................. 9

2 Drawing graphics primitives .................................. 11
  2.1 Draw solid and outlined rectangles ......................... 11
  2.2 Draw solid and outlined ellipses ......................... 12
  2.3 Detect the mouse ............................................ 13
  2.4 Draw a rectangle with the mouse ......................... 14
  2.5 Draw multiple shapes ...................................... 15
  2.6 Draw a polygon line with the mouse .................... 17

3 Working with images ............................................ 21
  3.1 Load an image ............................................... 21
  3.2 Move the image with the mouse ......................... 22
  3.3 Rotate and Scale the image ............................ 23
  3.4 Reset the image to the original ....................... 24
  3.5 Flip the image ............................................ 24
  3.6 Detect edges with the Laplacian ..................... 24
  3.7 Transform the image with the mouse ................ 25

4 Work with text .................................................... 31
  4.1 Initialize a font .......................................... 31
  4.2 Render the text ........................................... 32
  4.3 Edit text with the keyboard ............................ 34
  4.4 Add a blinking cursor ................................ 35

5 Making apps with Pygame ...................................... 37
  5.1 Create the App class .................................... 37
  5.2 Add the Text class ...................................... 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>Shortcut keys</td>
<td>40</td>
</tr>
<tr>
<td>5.4</td>
<td>Fullscreen, resizable and noframe mode</td>
<td>41</td>
</tr>
<tr>
<td>5.5</td>
<td>Add the Scene class</td>
<td>42</td>
</tr>
<tr>
<td>5.6</td>
<td>Scenes with background images</td>
<td>44</td>
</tr>
<tr>
<td>5.7</td>
<td>Automatic node placement</td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td>Create a graphical user interface (GUI)</td>
<td>49</td>
</tr>
<tr>
<td>6.1</td>
<td>Text attributes</td>
<td>49</td>
</tr>
<tr>
<td>6.2</td>
<td>Horizontal and vertical alignment</td>
<td>51</td>
</tr>
<tr>
<td>6.3</td>
<td>Text attributes</td>
<td>52</td>
</tr>
<tr>
<td>6.4</td>
<td>Editable text</td>
<td>53</td>
</tr>
<tr>
<td>6.5</td>
<td>Buttons</td>
<td>55</td>
</tr>
<tr>
<td>6.6</td>
<td>ListBox</td>
<td>56</td>
</tr>
<tr>
<td>6.7</td>
<td>Detecting double-clicks</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>Playing sound</td>
<td>59</td>
</tr>
<tr>
<td>7.1</td>
<td>Making sounds</td>
<td>59</td>
</tr>
<tr>
<td>8</td>
<td>Board Games</td>
<td>61</td>
</tr>
<tr>
<td>8.1</td>
<td>Selecting cells with the mouse</td>
<td>62</td>
</tr>
<tr>
<td>8.2</td>
<td>Adding background color</td>
<td>62</td>
</tr>
<tr>
<td>8.3</td>
<td>Create a checkerboard pattern</td>
<td>63</td>
</tr>
<tr>
<td>9</td>
<td>About Sphinx</td>
<td>65</td>
</tr>
<tr>
<td>9.1</td>
<td>Getting started</td>
<td>65</td>
</tr>
<tr>
<td>9.2</td>
<td>reStructuredText</td>
<td>66</td>
</tr>
<tr>
<td>9.3</td>
<td>Include from a file</td>
<td>67</td>
</tr>
<tr>
<td>9.4</td>
<td>Directives</td>
<td>68</td>
</tr>
<tr>
<td>9.5</td>
<td>Math formulas</td>
<td>70</td>
</tr>
<tr>
<td>9.6</td>
<td>The app module</td>
<td>70</td>
</tr>
<tr>
<td>9.7</td>
<td>Glossary</td>
<td>75</td>
</tr>
<tr>
<td>10</td>
<td>Indices and tables</td>
<td>77</td>
</tr>
</tbody>
</table>

Python Module Index | 79 |
Index | 81 |
This tutorial explains how to make interactive applications and games using Pygame. The first part is a general introduction to Pygame without defining classes and objects. The second part introduces classes and objects and teaches an object-oriented programming approach to making apps.
Pygame is a multimedia library for Python for making:

- games
- multimedia applications

It is a wrapper around the SDL (Simple DirectMedia Layer) library. In this section we introduce the basics of pygame functions without defining classes and objects.

1.1 Import the module

In order to use the methods defined in the pygame package, the pygame module must first be imported:

```python
import pygame
```

The import statement writes the pygame version and the following text to the console:

```
pygame 1.9.5
Hello from the pygame community. https://www.pygame.org/contribute.html
```

The pygame import statement is always placed at the beginning of the program. The effect is to import pygame classes, methods and attributes into the current name space. Now this new methods can be called via `pygame.method_name()`.

For example we can now initialize the pygame submodules with the following command:

```python
pygame.init()
```

Then we set the screen size with the function `display.set_mode()`. This function returns a `Surface` object which we assign to the variable `screen`. This variable will be one of the most important and most used variables. It is the one variable which represents what we see in the application:

```python
screen = pygame.display.set_mode((640, 240))
```
You can now run this program and test it. At this moment it does very little. It opens a window and closes it immediately.

### 1.2 Show the event loop

One of the essential parts of any interactive application is the event loop. Reacting to events allows the user to interact with the application. Events are the things that can happen in a program, such as a

- mouse click,
- mouse movement,
- keyboard press,
- joystick action.

The following is an infinite loop which prints all events to the console:

```python
while True:
    for event in pygame.event.get():
        print(event)
```

Try to move the mouse, click a mouse button, or type something on the keyboard. Every action you do produces an event which will be sent and printed on the console. This will look something like this:

```plaintext
<Event(4-MouseMotion {'pos': (173, 192), 'rel': (173, 192), 'buttons': (0, 0, 0), 'window': None})>
<Event(2-KeyDown {'unicode': 'a', 'key': 97, 'mod': 0, 'scancode': 0, 'window': None})>
<Event(3-KeyUp {'key': 97, 'mod': 0, 'scancode': 0, 'window': None})>
<Event(12-Quit {}>)
```

As we are in an infinite loop, it is impossible to quit this program from within the application. In order to quit the program, make the console the active window and type `ctrl-C`. This will write the following message to the console:

```
^CTraceback (most recent call last):
File "/Users/raphael/GitHub/pygame-tutorial/docs/tutorial1/intro1.py", line 7, in <module>
    for event in pygame.event.get():
KeyboardInterrupt
```

### 1.3 Quit the event loop properly

In order to quit the application properly, from within the application, by using the window close button (QUIT event), we modify the event loop. First we introduce the boolean variable `running` and set it to `True`. Within the event loop we check for the QUIT event. If it occurs, we set `running` to `False`:

```python
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```
Once the event loop, we call the `pygame.quit()` function to end the application correctly.

### 1.4 Define colors

Colors are defined as tuples of the base colors red, green and blue. This is called the **RGB model**. Each base color is represented as a number between 0 (minimum) and 255 (maximum) which occupies 1 byte in memory. An RGB color is thus represented as a 3-byte value. Mixing two or more colors results in new colors. A total of 16 million different colors can be represented this way.

Let’s define the base colors as tuples. Since these are constants, we are going to use capitals. At the beginning of the program we add:
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)

Further we define the colors obtained by mixing two or more of the base colors:

YELLOW = (255, 255, 0)
CYAN = (0, 255, 255)
MAGENTA = (255, 0, 255)
GRAY = (127, 127, 127)
WHITE = (255, 255, 255)

Inside the event loop, at its end we add the following:

```python
screen.fill(YELLOW)
pygame.display.update()
```

The method `fill(color)` fills the whole screen with the specified color. At this point nothing will be displayed. In order to show anything, the function `pygame.display.update()` must be called.

### 1.5 Switch the background color

At the beginning of the program we add a new variable `background` and initialize it to gray:

```python
background = GRAY
```

Within the event loop we are looking now for `KEYDOWN` events. If found, we check if the R or G keys have been pressed and change the background color to red (R) and green (G). This is the code added in the event loop:

```python
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_r:
        background = RED
    elif event.key == pygame.K_g:
        background = GREEN
```

(continues on next page)
elif event.key == pygame.K_g:
    background = GREEN

In the drawing section we use now the variable `background` representing the background color:

    screen.fill(background)
    pygame.display.update()

Test the program. Pressing the R and G keys allows you to switch the background color.

## 1.6 Import pygame.locals

The `pygame.locals` module contains some 280 constants used and defined by pygame. Placing this statement at the beginning of your program imports them all:

```python
import pygame
from pygame.locals import *
```

We find the key modifiers (alt, ctrl, cmd, etc.)

- `KMOD_ALT`, `KMOD_CAPS`, `KMOD_CTRL`, `KMOD_LALT`,
- `KMOD_LCTRL`, `KMOD_LMETA`, `KMOD_LSHIFT`, `KMOD_META`,
- `KMOD_MODE`, `KMOD_NONE`, `KMOD_NUM`, `KMOD_RALT`, `KMOD_RCTRL`,
- `KMOD_RMETA`, `KMOD_RSHIFT`, `KMOD_SHIFT`,

the number keys:

- `K_0`, `K_1`, `K_2`, `K_3`, `K_4`, `K_5`, `K_6`, `K_7`, `K_8`, `K_9`,

the special character keys:

- `K_AMPERSAND`, `K_ASTERISK`, `K_AT`, `K_BACKQUOTE`,
- `K_BACKSLASH`, `K_BACKSPACE`, `K_BREAK`,

the function keys:

- `K_F1`, `K_F2`, `K_F3`, `K_F4`, `K_F5`, `K_F6`, `K_F7`, `K_F8`,
- `K_F9`, `K_F10`, `K_F11`, `K_F12`, `K_F13`, `K_F14`, `K_F15`

the letter keys of the alphabet:

- `K_a`, `K_b`, `K_c`, `K_d`, `K_e`, `K_f`, `K_g`, `K_h`, `K_i`, `K_j`, `K_k`, `K_l`, `K_m`,
- `K_n`, `K_o`, `K_p`, `K_q`, `K_r`, `K_s`, `K_t`, `K_u`, `K_v`, `K_w`, `K_x`, `K_y`, `K_z`,

Instead of writing `pygame.KEYDOWN` we can now just write `KEYDOWN`.

## 1.7 Use a dictionary to decode keys

The easiest way to decode many keys, is to use a dictionary. Instead of defining many if-else cases, we just create a dictionary with the keyboard key entries. In this example we want to associate 8 different keys with 8 different background colors. At the beginning of the program we define this key-color dictionary:
key_dict = {K_k:BLACK, K_r:RED, K_g:GREEN, K_b:BLUE, K_y:YELLOW, K_c:CYAN, K_m:MAGENTA, K_w:WHITE}

print(key_dict)

Printing the dictionary to the console gives this result:

{(107: (0, 0, 0), 114: (255, 0, 0), 103: (0, 255, 0), 98: (0, 0, 255), 121: (255, 255, 0), 99: (0, 255, 255), 109: (255, 0, 255), 119: (255, 255, 255))

The keys are presented here with their ASCII code. For example the ASCII code for k is 107. Colors are represented as tuples. The color black is represented as (0, 0, 0).

The event loop now becomes very simple. First we check if the event type is a KEYDOWN event. If yes, we check if the event key is in the dictionary. If yes, we look up the color which is associated with that key and set the background color to it:

```python
if event.type == KEYDOWN:
    if event.key in key_dict:
        background = key_dict[event.key]
```

Try to press the 8 specified keys to change the background color.

### 1.8 Change the window caption

The function `pygame.display.set_caption(title)` allows to change the caption (title) of the application window. We can add this to the event loop:

```python
if event.key in key_dict:
    background = key_dict[event.key]
    caption = 'background color = ' + str(background)
    pygame.display.set_caption(caption)
```

This will display the RGB value of the current background color in the window caption.
1.9 Explore a simple ball game

To show what Pygame can do, here is a simple program that does a bouncing ball animation:

```python
import pygame
from pygame.locals import *

width = 640
height = 320
speed = [2, 2]
GREEN = (150, 255, 150)
running = True

pygame.init()
screen = pygame.display.set_mode((width, height))
b = pygame.image.load("ball.gif")
b = b.get_rect()

while running:
    for event in pygame.event.get:
        if event.type == QUIT:
            running = False

    ball = ball.move(speed)
    if ballrect.left < 0 or ballrect.right > width:
        speed[0] = -speed[0]
    if ballrect.top < 0 or ballrect.bottom > height:

    screen.fill(GREEN)
screen.blit(ball, ballrect)
    pygame.display.flip()

pygame.quit()
```
Try to understand what the program does. Then try to modify it's parameters.
The `pygame.draw` module allows to draw simple shapes to a surface. This can be the screen surface or any Surface object such as an image or drawing:

- rectangle
- polygon
- circle
- ellipse

The functions have in common that they:

- take a `Surface` object as first argument
- take a color as second argument
- take a width parameter as last argument
- return a `Rect` object which bounds the changed area

The following format:

```python
rect(Surface, color, Rect, width) -> Rect
polygon(Surface, color, pointlist, width) -> Rect
circle(Surface, color, center, radius, width) -> Rect
```

Most of the functions take a width argument. If the width is 0, the shape is filled.

### 2.1 Draw solid and outlined rectangles

The following draws first the background color and then adds three overlapping solid rectangles and next to it three outlined overlapping rectangles with increasing line width:
Try to modify the parameters and play with the drawing function.

### 2.2 Draw solid and outlined ellipses

The following code draws first the background color and then adds three overlapping solid ellipses and next to it three outlined overlapping ellipses with increasing line width:

```python
code
screen.fill(background)
pygame.draw.ellipse(screen, RED, (50, 20, 160, 100))
pygame.draw.ellipse(screen, GREEN, (100, 60, 160, 100))
pygame.draw.ellipse(screen, BLUE, (150, 100, 160, 100))

pygame.draw.ellipse(screen, RED, (350, 20, 160, 100), 1)
pygame.draw.ellipse(screen, GREEN, (400, 60, 160, 100), 4)
pygame.draw.ellipse(screen, BLUE, (450, 100, 160, 100), 8)

pygame.display.update()
```
2.3 Detect the mouse

Pressing the mouse buttons produces MOUSEBUTTONDOWN and MOUSEBUTTONUP events. The following code in the event loop detects them and writes the event to the console:

```python
for event in pygame.event.get():
    if event.type == QUIT:
        running = False
    elif event.type == MOUSEBUTTONDOWN:
        print(event)
    elif event.type == MOUSEBUTTONUP:
        print(event)
```

Pressing the mouse buttons produces this kind of events:

```python
<Event(5-MouseButtonDown {'pos': (123, 88), 'button': 1, 'window': None})>
<Event(6-MouseButtonUp {'pos': (402, 128), 'button': 1, 'window': None})>
<Event(5-MouseButtonDown {'pos': (402, 128), 'button': 3, 'window': None})>
<Event(6-MouseButtonUp {'pos': (189, 62), 'button': 3, 'window': None})>
```

Just moving the mouse produces a MOUSEMOTION event. The following code detects them and writes the event to the console:

```python
elif event.type == MOUSEMOTION:
    print(event)
```

Moving the mouse produces this kind of event:

```python
<Event(4-MouseMotion {'pos': (537, 195), 'rel': (-1, 0), 'buttons': (0, 0, 0), 'window': None})>
<Event(4-MouseMotion {'pos': (527, 189), 'rel': (-10, -6), 'buttons': (0, 0, 0), 'window': None})>
<Event(4-MouseMotion {'pos': (508, 180), 'rel': (-19, -9), 'buttons': (0, 0, 0), 'window': None})>
```
2.4 Draw a rectangle with the mouse

We can use this three events to draw a rectangle on the screen. We define the rectangle by its diagonal start and end point. We also need a flag which indicates if the mouse button is down and if we are drawing:

```python
start = (0, 0)
size = (0, 0)
drawing = False
```

When the mouse button is pressed, we set start and end to the current mouse position and indicate with the flag that the drawing mode has started:

```python
elif event.type == MOUSEBUTTONDOWN:
    start = event.pos
    size = 0, 0
    drawing = True
```

When the mouse button is released, we set the end point and indicate with the flag that the drawing mode has ended:

```python
elif event.type == MOUSEBUTTONUP:
    end = event.pos
    size = end[0] - start[0], end[1] - start[1]
    drawing = False
```

When the mouse is moving we have also have to check if we are in drawing mode. If yes, we set the end position to the current mouse position:

```python
elif event.type == MOUSEMOTION and drawing:
    end = event.pos
    size = end[0] - start[0], end[1] - start[1]
```

Finally we draw the rectangle to the screen. First we fill in the background color. Then we calculate the size of the rectangle. Finally we draw it, and at the very last we update the screen:

```python
screen.fill(GRAY)
pygame.draw.rect(screen, RED, (start, size), 2)
pygame.display.update()
```
2.5 Draw multiple shapes

To draw multiple shapes, we need to place them into a list. Besides variables for `start`, `end` and `drawing` we add a rectangle list:

```python
start = (0, 0)
size = (0, 0)
drawing = False
rect_list = []
```

When drawing of an object (rectangle, circle, etc.) is done, as indicated by a MOUSEBUTTONDOWN event, we create a rectangle and append it to the rectangle list:

```python
elif event.type == MOUSEBUTTONDOWN:
    end = event.pos
    size = end[0]-start[0], end[1]-start[1]
    rect = pygame.Rect(start, size)
    rect_list.append(rect)
    drawing = False
```

In the drawing code, we first fill the background color, then iterate through the rectangle list to draw the objects (red, thickness=3), and finally we draw the current rectangle which is in the process of being drawn (blue, thickness=1):

```python
screen.fill(GRAY)
for rect in rect_list:
    pygame.draw.rect(screen, RED, rect, 3)
    pygame.draw.rect(screen, BLUE, (start, size), 1)
pygame.display.update()
```
Here is the complete file:

"""Place multiple rectangles with the mouse."""

```python
import pygame
from pygame.locals import *

RED = (255, 0, 0)
BLUE = (0, 0, 255)
GRAY = (127, 127, 127)

pygame.init()
screen = pygame.display.set_mode((640, 240))

start = (0, 0)
size = (0, 0)
drawing = False
rect_list = []

running = True

while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False

        elif event.type == MOUSEBUTTONDOWN:
            start = event.pos
            size = 0, 0
            drawing = True

        elif event.type == MOUSEBUTTONUP:
            end = event.pos
            size = end[0]-start[0], end[1]-start[1]
            rect = pygame.Rect(start, size)
            rect_list.append(rect)
```

(continues on next page)
2.6 Draw a polygon line with the mouse

To draw a polygon line we need to add the points to a list of points. First we define an empty point list and a drawing flag:

```python
drawing = False
points = []
```

At the MOUSEBUTTONDOWN event we add the current point to the list and set the `drawing` flag to True:

```python
def main():
    drawing = False
    points = []

    # Main game loop
    while True:
        # Handle events
        for event in pygame.event.get():
            if event.type == MOUSEBUTTONDOWN:
                points.append(event.pos)
                drawing = True
            elif event.type == KEYDOWN:
                if event.key == K_ESCAPE:
                    if len(points) > 0:
                        points.pop()
            # Update display
        screen.fill(GRAY)
        # Draw lines
        if len(points) > 1:
            rect = pygame.draw.lines(screen, RED, True, points, 3)
            pygame.draw.rect(screen, GREEN, rect, 1)
        pygame.display.update()

    # Quit game
    pygame.quit()
```

At the MOUSEBUTTONUP event we deactivate the `drawing` flag:

```python
elif event.type == MOUSEBUTTONUP:
    drawing = False
```

At the MOUSEMOTION event we move the last point in the polygon list if the drawing flag is set:

```python
elif event.type == MOUSEMOTION and drawing:
    points[-1] = event.pos
```

If there are more than 2 points in the point list we draw a polygon line. Each `pygame.draw` function returns a `Rect` of the bounding rectangle. We display this bounding rectangle in green:

```python
screen.fill(GRAY)
if len(points) > 1:
    rect = pygame.draw.lines(screen, RED, True, points, 3)
    pygame.draw.rect(screen, GREEN, rect, 1)
pygame.display.update()
```

Pressing the ESCAPE key will remove the last point in the list:

```python
elif event.type == KEYDOWN:
    if event.key == K_ESCAPE:
        if len(points) > 0:
            points.pop()
```

2.6. Draw a polygon line with the mouse
Here is the complete file:

```python
import pygame
from pygame.locals import *

RED = (255, 0, 0)
GREEN = (0, 255, 0)
GRAY = (150, 150, 150)

pygame.init()
screen = pygame.display.set_mode((640, 240))

drawing = False
points = []
running = True

while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False

        elif event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                if len(points) > 0:
                    points.pop()

        elif event.type == MOUSEBUTTONDOWN:
            points.append(event.pos)
            drawing = True

        elif event.type == MOUSEBUTTONUP:
            drawing = False

        elif event.type == MOUSEMOTION and drawing:

(continues on next page)```
points[-1] = event.pos

screen.fill(GRAY)
if len(points)>1:
    rect = pygame.draw.lines(screen, RED, True, points, 3)
    pygame.draw.rect(screen, GREEN, rect, 1)
    pygame.display.update()

pygame.quit()
CHAPTER 3

Working with images

3.1 Load an image

The `pygame.image` module provides methods for loading and saving images. The method `load()` loads an image from the file system and returns a Surface object. The method `convert()` optimizes the image format and makes drawing faster:

```python
img = pygame.image.load('bird.png')
img.convert()
```

Download the image `bird.png` to the same folder where your program resides:

`bird.png`

The method `get_rect()` returns a Rect object from an image. At this point only the size is set and position is placed at (0, 0). We set the center of the Rect to the center of the screen:

```python
rect = img.get_rect()
rect.center = w//2, h//2
```

To recapitulate, we are working with 3 objects:
- `screen` is the Surface object representing the application window
- `img` is the Surface object of the image to display
- `rect` is the Rect object which is the bounding rectangle of the image

To display the image we fill the screen with a background color (GRAY). Then we blit the image, draw a red rectangle around it and finally update the screen:

```python
screen.fill(GRAY)
screen.blit(img, rect)
pygame.draw.rect(screen, RED, rect, 1)
pygame.display.update()
```
3.2 Move the image with the mouse

At the beginning of the programm we set a boolean variable `moving` to False. Only when the mouse button is pressed, and when the mouse position is within the image (collidepoint) we set it to True:

```python
elif event.type == MOUSEBUTTONDOWN:
    if rect.collidepoint(event.pos):
        moving = True
```

When the mouse button is released, we set it to False again:

```python
elif event.type == MOUSEBUTTONUP:
    moving = False
```

When the mouse moves, and the flag `moving` is True, then we move the image by the amount of relative movement (event.rel):

```python
elif event.type == MOUSEMOTION and moving:
    rect.move_ip(event.rel)
```

This is the whole code:

```python
"""Move an image with the mouse."""

import pygame
from pygame.locals import *

RED = (255, 0, 0)
GRAY = (150, 150, 150)

pygame.init()
w, h = 640, 240
screen = pygame.display.set_mode((w, h))
running = True
```

```python
img = pygame.image.load('bird.png')
img.convert()
rect = img.get_rect()
rect.center = w/2, h/2
moving = False

while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False
        elif event.type == MOUSEBUTTONDOWN:
            if rect.collidepoint(event.pos):
                moving = True
        elif event.type == MOUSEBUTTONUP:
            moving = False
        elif event.type == MOUSEMOTION and moving:
            rect.move_ip(event.rel)

    screen.fill(GRAY)
    screen.blit(img, rect)
    pygame.draw.rect(screen, RED, rect, 1)
    pygame.display.update()

pygame.quit()
```

### 3.3 Rotate and Scale the image

The `pygame.transform` module provides methods for **scaling**, **rotating** and **flipping** images. As we are going to modify the image `img` we keep the original image in a variable called `img0`:

```python
img0 = pygame.image.load(path)
img0.convert()
```

In order to show the image rectangle, we add a green border to the original image:

```python
rect0 = img0.get_rect()
pygame.draw.rect(img0, GREEN, rect0, 1)
```

Then we place the image in the center of the screen:

```python
center = w//2, h//2
img = img0
rect = img.get_rect()
rect.center = center
```

First we define the global variables `scale` and `angle`:

```python
angle = 0
scale = 1
```

We use the R key to increment rotation by 10 degrees and (decrement if the SHIFT key is pressed). The function
rotozoom() allows to combine rotation and scaling. We always transform the original image (img0). Repeated rotation or scaling of an image would degrade its quality:

```python
if event.type == KEYDOWN:
    if event.key == K_r:
        if event.mod & KMOD_SHIFT:
            angle -= 10
        else:
            angle += 10
        img = pygame.transform.rotozoom(img0, angle, scale)
```

We use the S key to increment the scale by 10% (decrease if the SHIFT key is pressed):

```python
elif event.key == K_s:
    if event.mod & KMOD_SHIFT:
        scale /= 1.1
    else:
        scale *= 1.1
    img = pygame.transform.rotozoom(img0, angle, scale)
```

As the image is transformed the bounding rectangle changes size. It must be recalculated and placed at the center again:

```python
rect = img.get_rect()
rect.center = center
```

### 3.4 Reset the image to the original

We use the O key to reset the image to its original state:

```python
elif event.key == K_o:
    img = img0
    angle = 0
    scale = 1
```

### 3.5 Flip the image

We use the H key to flip the image horizontally:

```python
elif event.key == K_h:
    img = pygame.transform.flip(img, True, False)
```

and the V key to flip the image vertically:

```python
elif event.key == K_v:
    img = pygame.transform.flip(img, False, True)
```

### 3.6 Detect edges with the Laplacian

The function `laplacien(img)` allows to detect the outline of the image:
elif event.key == K_l:
    img = pygame.transform.laplacian(img)

The function `scale2x(img)` doubles the size of a pixel:

elif event.key == K_2:
    img = pygame.transform.scale2x(img)

3.7 Transform the image with the mouse

In this section we show how to use the mouse to scale and rotate an image. At the beginning we import the `math` module:
At the beginning we store the initial mouse position:

```python
mouse = pygame.mouse.get_pos()
```

When the mouse moves we update the mouse position `mouse` and calculate the x, y coordinates from the center of the image. We also calculate the center-mouse distance d.

```python
elif event.type == MOUSEMOTION:
    mouse = event.pos
    x = mouse[0] - center[0]
    y = mouse[1] - center[1]
    d = math.sqrt(x ** 2 + y ** 2)
```

The `atan2(y, x)` math function allows to find the rotation angle. We need to transform radians in degrees. From the distance mouse-center we calculate the scale argument:

```python
angle = math.degrees(-math.atan2(y, x))
scale = abs(5 * d / w)
img = pygame.transform.rotozoom(img0, angle, scale)
rect = img.get_rect()
rect.center = center
```

To finally draw the transformed image we first fill the whole screen background (GRAY), blit the transformed image, surround it with a red rectangle.

In order to give visual feedback for the mouse action when transforming an image, we

- draw a green line between the center of the image and the mouse position,
- place two circles on the center and on the mouse position:

```python
screen.fill(GRAY)
screen.blit(img, rect)
pygame.draw.rect(screen, RED, rect, 1)
pygame.draw.line(screen, GREEN, center, mouse, 1)
pygame.draw.circle(screen, RED, center, 6, 1)
pygame.draw.circle(screen, RED, mouse, 6, 1)
pymgame.display.update()
```
Here is the full code.

```python
'''Rotate, scale and flip an image.'''

import pygame
import math, sys, os
from pygame.locals import *

RED = (255, 0, 0)
GREEN = (0, 255, 0)
GRAY = (150, 150, 150)

pygame.init()
w, h = 640, 240
screen = pygame.display.set_mode((w, h))
running = True

module = sys.modules['__main__']
path, name = os.path.split(module.__file__)
path = os.path.join(path, 'bird.png')

img0 = pygame.image.load(path)
img0.convert()
rect0 = img0.get_rect()
pygame.draw.rect(img0, GREEN, rect0, 1)

center = w//2, h//2
img = img0
rect = img.get_rect()
rect.center = center

angle = 0
scale = 1
```

3.7. Transform the image with the mouse
mouse = pygame.mouse.get_pos()

while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False

        if event.type == KEYDOWN:
            if event.key == K_r:
                if event.mod & KMOD_SHIFT:
                    angle -= 10
                else:
                    angle += 10
                img = pygame.transform.rotozoom(img0, angle, scale)
            elif event.key == K_s:
                if event.mod & KMOD_SHIFT:
                    scale /= 1.1
                else:
                    scale *= 1.1
                img = pygame.transform.rotozoom(img0, angle, scale)
            elif event.key == K_o:
                img = img0
                angle = 0
                scale = 1
            elif event.key == K_h:
                img = pygame.transform.flip(img, True, False)
            elif event.key == K_v:
                img = pygame.transform.flip(img, False, True)
            elif event.key == K_l:
                img = pygame.transform.laplacian(img)
            elif event.key == K_2:
                img = pygame.transform.scale2x(img)

        rect = img.get_rect()
        rect.center = center

    elif event.type == MOUSEMOTION:
        mouse = event.pos
        x = mouse[0] - center[0]
        y = mouse[1] - center[1]
        d = math.sqrt(x ** 2 + y ** 2)
        angle = math.degrees(-math.atan2(y, x))
        scale = abs(5 * d / w)
        img = pygame.transform.rotozoom(img0, angle, scale)
        rect = img.get_rect()
        rect.center = center

    screen.fill(GRAY)
    screen.blit(img, rect)
    pygame.draw.rect(screen, RED, rect, 1)
pygame.draw.line(screen, GREEN, center, mouse, 1)
pygame.draw.circle(screen, RED, center, 6, 1)
pygame.draw.circle(screen, RED, mouse, 6, 1)
pygame.display.update()

pygame.quit()
Work with text

In pygame, text cannot be written directly to the screen. The first step is to create a Font object with a given font size. The second step is to render the text into an image with a given color. The third step is to blit the image to the screen. These are the steps:

```python
font = pygame.font.SysFont(None, 24)
img = font.render('hello', True, BLUE)
screen.blit(img, (20, 20))
```

Once the font is created, its size cannot be changed. A Font object is used to create a Surface object from a string. Pygame does not provide a direct way to write text onto a Surface object. The method `render()` must be used to create a Surface object from the text, which then can be blit to the screen. The method `render()` can only render single lines. A newline character is not rendered.

### 4.1 Initialize a font

Initializing the font can take a few seconds. On a MacBook Air the creation of a system Font object:

```python
t0 = time.time()
font = pygame.font.SysFont(None, 48)
print('time needed for Font creation :', time.time()-t0)
```

took more then 8 seconds:

```
time needed for Font creation : 8.230187892913818
```

The function `get_fonts()` returns a list of all installed fonts. The following code checks what fonts are on your system and how many, and prints them to the console:

```python
fonts = pygame.font.get_fonts()
print(len(fonts))
for f in fonts:
    print(f)
```
4.2 Render the text

The font object can render a given text into an image. In the example below, we place a blue bounding rectangle around that text image:

```python
img = font.render(sysfont, True, RED)
rect = img.get_rect()
pygame.draw.rect(img, BLUE, rect, 1)
```

We then create two more fonts, *Chalkduster* and *Didot* at a size of 72 points. We render a text with both fonts:

```python
font1 = pygame.font.SysFont('chalkduster.ttf', 72)
img1 = font1.render('chalkduster.ttf', True, BLUE)

font2 = pygame.font.SysFont('didot.ttc', 72)
img2 = font2.render('didot.ttc', True, GREEN)
```

Finally the text images are blit to the screen like regular images:

```python
screen.fill(background)
screen.blit(img, (20, 20))
screen.blit(img1, (20, 50))
screen.blit(img2, (20, 120))
pygame.display.update()
```

This is the result:
"""Draw text to the screen."""
import pygame
from pygame.locals import *
import time

BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
GRAY = (200, 200, 200)

pygame.init()
screen = pygame.display.set_mode((640, 240))

sysfont = pygame.font.get_default_font()
print('system font :', sysfont)
t0 = time.time()
font = pygame.font.SysFont(None, 48)
print('time needed for Font creation :', time.time()-t0)

img = font.render(sysfont, True, RED)
rect = img.get_rect()
pygame.draw.rect(img, BLUE, rect, 1)

font1 = pygame.font.SysFont('chalkduster.ttf', 72)
img1 = font1.render('chalkduster.ttf', True, BLUE)

font2 = pygame.font.SysFont('didot.ttc', 72)
img2 = font2.render('didot.ttc', True, GREEN)

fonts = pygame.font.get_fonts()
print(len(fonts))
for i in range(7):
  
4.2. Render the text
print(fonts[i])

running = True
background = GRAY
while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False

    screen.fill(background)
    screen.blit(img, (20, 20))
    screen.blit(img1, (20, 50))
    screen.blit(img2, (20, 120))
    pygame.display.update()

pygame.quit()

4.3 Edit text with the keyboard

The keyboard event can be used to edit a text. First we create a text which we save in a string variable `text` and which we render to an image:

```python
text = 'this text is editable'
font = pygame.font.SysFont(None, 48)
img = font.render(text, True, RED)
```

Then we define the bounding rectangle and furthermore a cursor rectangle which is juxtaposed to the text bounding rectangle:

```python
rect = img.get_rect()
rect.topleft = (20, 20)
cursor = Rect(rect.topleft, (3, rect.height))
```

Inside the event loop we watch out for KEYDOWN events. If the key press is a BACKSPACE and the length of the string is larger than 0, then we remove the last character, else we append the new character to the text variable:

```python
if event.type == KEYDOWN:
    if event.key == K_BACKSPACE:
        if len(text)>0:
            text = text[:-1]
    else:
        text += event.unicode
```

Then we render the modified text, update the bounding rectangle, and place the cursor box at the end of the updated bounding rectangle:

```python
img = font.render(text, True, RED)
rect.size = img.get_size()
cursor.topleft = rect.topright
```
4.4 Add a blinking cursor

In order to make the cursor more visible, we let it blink every 0.5 seconds. We do this using the time.time() floating point value:

```python
screen.fill(background)
screen.blit(img, rect)
if time.time() % 1 > 0.5:
    pygame.draw.rect(screen, RED, cursor)
pygame.display.update()
```

This is the result:

![Pygame window with blinking cursor](image)

Here is the full code.

```
"""Edit text with the keyboard."""
import pygame
from pygame.locals import *
import time

BLACK = (0, 0, 0)
RED = (255, 0, 0)
GRAY = (200, 200, 200)

pygame.init()
screen = pygame.display.set_mode((640, 240))
text = 'this text is editable'
font = pygame.font.SysFont(None, 48)
img = font.render(text, True, RED)
rect = img.get_rect()
rect.topleft = (20, 20)
cursor = Rect(rect.topright, (3, rect.height))

running = True
background = GRAY

(continues on next page)"""
while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False

        if event.type == KEYDOWN:
            if event.key == K_BACKSPACE:
                if len(text) > 0:
                    text = text[:-1]
            else:
                text += event.unicode
            img = font.render(text, True, RED)
            rect.size = img.get_size()
            cursor.topleft = rect.topright

        screen.fill(background)
        screen.blit(img, rect)
        if time.time() % 1 > 0.5:
            pygame.draw.rect(screen, RED, cursor)
            pygame.display.update()

pygame.quit()
In this section we are going to create applications and games with Pygame. From here on we will be using an object-oriented programming (OOP) approach.

Pygame only allows to create one single window. Different from other applications, those based on Pygame cannot have multiple windows. If for example dialog window is needed, it must be displayed within the main window.

Within an application we provide multiples scenes (environments, rooms, or levels). Each scene contains different objects such as:

- text
- sprites (images)
- GUI elements (buttons, menus)
- shapes (rectangles, circles)

### 5.1 Create the App class

The basis for a game or application is the App class. The first thing to do is to import the pygame module, as well as a series of useful constants:

```python
import pygame
from pygame.locals import *
```

Then we create define the App class which initializes Pygame and opens a the app window:

```python
class App:
    """Create a single-window app with multiple scenes.""

    def __init__(self):
        """Initialize pygame and the application.""
        pygame.init()
        flags = RESIZABLE
```

(continues on next page)
Further we have to define the main event loop:

```python
def run(self):
    """Run the main event loop.""
    while App.running:
        for event in pygame.event.get():
            if event.type == QUIT:
                App.running = False
        pygame.quit()
```

At the end of the module we run a demo, if the program is run directly and not imported as a module:

```python
if __name__ == '__main__':
    App().run()
```

## 5.2 Add the Text class

Now we add some text to the screen. We create a Text class from which we can instantiate text objects:

```python
class Text:
    """Create a text object.""
    def __init__(self, text, pos, **options):
        self.text = text
        self.pos = pos
        self.fontname = None
        self.fontsize = 72
        self.fontcolor = Color('black')
        self.set_font()
        self.render()
```

The `Font` object needs to be created initially and everytime the font name or the font size changes:

```python
def set_font(self):
    """Set the font from its name and size.""
    self.font = pygame.font.Font(self.fontname, self.fontsize)
```

The text needs to be rendered into a surface object, an image. This needs to be done only once, or whenever the text changes:

```python
def render(self):
    """Render the text into an image.""
    self.img = self.font.render(self.text, True, self.fontcolor)
    self.rect = self.img.get_rect()
    self.rect.topleft = self.pos
```

Drawing the text means blitting it to the application screen:
```python
def draw(self):
    """Draw the text image to the screen.""
    App.screen.blit(self.img, self.rect)
```

This is the result:

![Pygame App](image)

Here is the complete code:

```python
import pygame
from pygame.locals import *

class Text:
    """Create a text object.""
    def __init__(self, text, pos, **options):
        self.text = text
        self.pos = pos

        self.fontname = None
        self.fontsize = 72
        self.fontcolor = Color('black')
        self.set_font()
        self.render()

    def set_font(self):
        """Set the Font object from name and size.""
        self.font = pygame.font.Font(self.fontname, self.fontsize)

    def render(self):
        """Render the text into an image.""
        self.img = self.font.render(self.text, True, self.fontcolor)
        self.rect = self.img.get_rect()
        self.rect.topleft = self.pos

    def draw(self):
        """Draw the text image to the screen.""
```

(continues on next page)
class App:
    """Create a single-window app with multiple scenes."""

    def __init__(self):
        """Initialize pygame and the application."""
        pygame.init()
        flags = RESIZABLE
        App.screen = pygame.display.set_mode((640, 240), flags)
        App.t = Text('Pygame App', pos=(20, 20))

        App.running = True

    def run(self):
        """Run the main event loop."""
        while App.running:
            for event in pygame.event.get():
                if event.type == QUIT:
                    App.running = False

            App.screen.fill(Color('gray'))
            App.t.draw()
            pygame.display.update()

        pygame.quit()

if __name__ == '__main__':
    App().run()

5.3 Shortcut keys

Key presses (called shortcuts) can be used to interact with the application and run commands. We can add the following code inside the event loop to intercept the S key and print a message:

```python
if event.type == KEYDOWN:
    if event.key == K_s:
        print('Key press S')
```

If the application has many shortcuts, the keys alone may not be enough and modifier keys (cmd, ctrl, alt, shift) can be used to increase the number of combinations. The easiest way to represent these shortcuts is under the form of a dictionary, where the key/mod tuples are associated with a command strings. The dictionary has this shape:

```python
self.shortcuts = {
    (K_x, KMOD_LMETA): 'print("cmd+X")',
    (K_x, KMOD_LALT): 'print("alt+X")',
    (K_x, KMOD_LCTRL): 'print("ctrl+X")',
    (K_x, KMOD_LMETA + KMOD_LALT): 'print("cmd+alt+X")',
    (K_x, KMOD_LMETA + KMOD_LALT + KMOD_LSHIFT): 'print("cmd+alt+shift+X")',
}
```

Inside the event loop we detect keydown events and call the key handler:
if event.type == KEYDOWN:
    self.do_shortcut(event)

The `do_shortcut()` method looks up the shortcut and executes the command string:

def do_shortcut(self, event):
    
        Find the key/mod combination in the dictionary and execute the cmd.
        
    k = event.key
    m = event.mod
    if (k, m) in self.shortcuts:
        exec(self.shortcuts[k, m])

This is the result on the console when pressing different key+modifier combinations:

```plaintext
cmd+X
alt+X
ctrl+X
cmd+shift+X
cmd+alt+X
cmd+alt+shift+X
```

### 5.4 Fullscreen, resizable and noframe mode

Pygame allows a window to be displayed in 3 different modes:

- fullscreen mode
- resizable (a resize edge is displayed)
- noframe mode (without a window title bar)

Inside the `App` class `__init__()` method we first define the screen size and the display mode flags, and then create the screen surface:

```python
self.flags = RESIZABLE
self.rect = Rect(0, 0, 640, 240)
App.screen = pygame.display.set_mode(self.rect.size, self.flags)
```

In order to toggle (turn on and off) the three display modes we add these entries to the `shortcuts` dictionary:

```python
(K_f, KMOD_LMETA): 'self.toggle_fullscreen()',
(K_r, KMOD_LMETA): 'self.toggle_resizable()',
(K_g, KMOD_LMETA): 'self.toggle_frame()',
```

Inside the `App` class we define three methods to toggle the corresponding mode flag, by using the bit-wise XOR operator (^=):

```python
def toggle_fullscreen(self):
    
        Toggle between full screen and windowed screen.
        
    self.flags ^= FULLSCREEN
    pygame.display.set_mode((0, 0), self.flags)

def toggle_resizable(self):
    
        Toggle between resizable and fixed-size window.
        
    self.flags ^= RESIZABLE
    pygame.display.set_mode(self.rect.size, self.flags)
```

(continues on next page)
def toggle_frame(self):
    """Toggle between frame and noframe window."""
    self.flags ^= NOFRAME
    pygame.display.set_mode(self.rect.size, self.flags)

5.5 Add the Scene class

Most applications or games have different scenes, such as an introduction screen, an intro, and different game levels. So we are going to define the Scene class:

class Scene:
    """Create a new scene (room, level, view)."""
    id = 0
    bg = Color('gray')

When creating a new scene, we append the scene to the applications scene list and make this scene the current scene:

def __init__(self, *args, **kwargs):
    # Append the new scene and make it the current scene
    App.scenes.append(self)
    App.scene = self

Then we set a scene id, which is kept as class attribute of the Scene class. Then we set the nodes list to the empty list and set the background color:

# Set the instance id and increment the class id
self.id = Scene.id
Scene.id += 1
self.nodes = []
self.bg = Scene.bg

The scene object knows how to draw itself. It first fills the background with the background color, then draws each nodes and finally flips the display to update the screen:

def draw(self):
    """Draw all objects in the scene."""
    App.screen.fill(self.bg)
    for node in self.nodes:
        node.draw()
    pygame.display.flip()

The string representation of the scene is Scene followed by its ID number:

def __str__(self):
    return 'Scene {}'.format(self.id)

This is an image of scene 0 with two text objects and a default gray background color. The second text object has been selected.
Here is the complete code:

```
from app import *
```

(continues on next page)
5.6 Scenes with background images

We can add a background image to a scene:

```python
self.file = Scene.options['file']
if self.file != '':
    self.img = pygame.image.load(self.file)
    size = App.screen.get_size()
    self.img = pygame.transform.smoothscale(self.img, size)
    self.enter()
```

This is an image of scene 0 with a forest background image and a white Text object.

![Forest scene](image)

This is an image of scene 1 with a lake background image and a black Text object.
This is an image of scene 2 with a sunset background image and a white Text object.

Here is the complete code:

```python
"""Display different scene background images."""
from app import *

class Demo(App):
    def __init__(self):
        super().__init__()
        Scene(img_folder='..//background', file='forest.jpg', caption='Forest')
        Text('Forest scene', fontcolor=Color('white'))
        Scene(file='lake.jpg', caption='Lake')
        Text('Lake scene')
        Scene(file='sunset.jpg', caption='Sunset')
        Text('Sunset scene', fontcolor=Color('white'))
        Scene(file='', bg=Color('lightgreen'), caption='Green background')
        Text('Colored background scene')

if __name__ == '__main__':
    Demo().run()
```

5.6. Scenes with background images
5.7 Automatic node placement

Nodes are containers for GUI elements. It is convenient if they can be placed automatically inside a scene.

- **pos** the current position
- **size** the current size
- **dir** the current direction: vertical (1, 0), horizontal (0, 1), diagonal (1, 1)
- **gap** the spacing

The default placement direction is vertical. Nodes placed in a scene stack up vertically. At any time the node position, node size, node gap or node direction can be changed:

```python
Scene(caption='Nodes - vertical placement')
Node()
Node()
Node()
Node(pos=(200, 20))
Node()
Node()
```

Here we change the node placement direction to horizontal, dir=(0, 1). At any time we can change the node position or gap. We can place the initial node position at (0, 0) and change the gap to (0, 0):

```python
Scene(caption='Nodes - horizontal placement')
Node(dir=(1, 0), pos=(0, 0), gap=(0, 0))
Node()
Node()
Node(pos=(0, 100))
Node()
Node()
```
The placement can also be diagonal by choosing the direction vector \( \text{dir} = (1, 1) \):

```python
Scene(caption='Nodes - diagonale placement')
Node(dir=(1, 1), gap=(0, 0))
Node()
```

Here is the complete code:

```python
from app import *

class Demo(App):
    def __init__(self):
        super().__init__()

        Scene(caption='Nodes - vertical placement')
        Node()
        Node()
        Node()
        Node(pos=(200, 20), size=(200, 50))
        Node()
        Node()

        Scene(caption='Nodes - horizontal placement')
```

(continues on next page)
Node(dir=(1, 0), gap=(0, 0))
Node()
Node()

Node(pos=(20, 100)
Node()
Node()

Scene(caption='Nodes - diagonal placement')
Node(dir=(1, 1), gap=(0, 0))
Node()
Node()

if __name__ == '__main__':
    Demo().run()
The graphical user interface (GUI) consists of all the elements the user can interact with (read, click, drag, resize, select, input):

- text
- button
- checkbutton
- radiobutton
- menu (pop-up, pull-down)
- listbox
- slider

### 6.1 Text attributes

We store all pygame text attributes as class variables:

```python
class Text(Node):
    """Create a text object which knows how to draw itself."""
    fontname = None
    fontsize = 36
    fontcolor = Color('black')
    background = None
    italic = False
    bold = False
    underline = False
```

After initializing the Node, we update the instance variables from the Text class variables:
super().__init__(**options)
self.__dict__.update(Text.options)

The font size and the three styles (bold, italic, underline) are set at font creation:

def set_font(self):
    """Set the font and its properties."""
    self.font = pygame.font.Font(self.fontname, self.fontsize)
    self.font.set_bold(self.bold)
    self.font.set_italic(self.italic)
    self.font.set_underline(self.underline)

The font color and the background color are set when rendering the text:

def render(self):
    """Render the text into an image."""
    self.img = self.font.render(self.text, True, self.fontcolor, self.background)
    self.rect.size = self.img.get_size()

Here is a code example:

"""Display text with different size, color and font."""
from app import *
class Demo(App):
    def __init__(self):
        super().__init__()
        Scene(caption='Text')
        Text('Default text')
        Text('fontsize = 24', fontsize=24)
        Text('fontcolor = RED', fontcolor=Color('red'))
        Text('48 pts, blue', fontsize=48, fontcolor=Color('blue'))
        Text('fontbg = yellow', fontcolor=Color('yellow'))
        Text('italic', pos=(400, 20), italic=True)
        Text('bold', bold=True)
        Text('underline', underline=True, font_bg=None)

if __name__ == '__main__':
    Demo().run()

Which produces this result:
6.2 Horizontal and vertical alignment

For a given box size, text can be aligned horizontally to the left, center, or right. The following code aligns the text image with these three positions:

```python
w, h = self.rect.size
w0, h0 = self.text_img.get_size()

if self.h_align == 0:
    x = 0
elif self.h_align == 1:
    x = (w-w0)//2
else:
    x = w-w0

self.img0.blit(self.text_img, (x, y))
self.img = self.img0.copy()
```

In the vertical direction the text image can be aligned at the top, middle or bottom:

```python
if self.v_align == 0:
    y = 0
elif self.v_align == 1:
    y = (h-h0)//2
else:
    y = h-h0

self.img0.blit(self.text_img, (x, y))
self.img = self.img0.copy()
```

The image `img0` is the original, used for scaling. The `img` is the one used for drawing.

Here is a code example:

```python
"""Horizontal and vertical text alignment.""
from app import *

class Demo(App):
    def __init__(self):
        super().__init__()

        Scene(caption='Text Alignment', bg=Color('pink'))
```

(continues on next page)
6.3 Text attributes

A Text object has various attributes which are remembered.

Here is a code example:

```python
from app import *

class Demo(App):
    def __init__(self):
        super().__init__()

        Scene(caption='Text', bg=Color('pink'))
        Text(size=(100, 40))
        Text(bg=Color('yellow'), h_align=1)
        Text(fontcolor=Color('red'))
        Text(fontbg=Color('green'), cmd='print(self.text)')

        Text(pos=(200, 20))
        Text(italic=True, v_align=1)
        Text(underline=True, fontsize=24)
        Text(bold=True)
```

(continues on next page)
If \texttt{__name__} == \texttt{"__main__"}:
    \texttt{Demo().run()}

It produces the following result:

6.4 Editable text

The class \texttt{TextEdit} provides editable text with a movable cursor. The cursor is represented as a small rectangle which is rendered under the text. A selection is represented as a large rectangle under the selected letters.

6.4.1 Create the cursor

The class attribute \texttt{TextEdit.cursor} defines the cursor color and width:

\begin{verbatim}
cursor = Color('red'), 2  # cursor color and width
\end{verbatim}

Inside the constructor, the cursor is placed at the end of the text. A cursor image is created and filled with the cursor color. The cursor rectangle is initially placed at the end of the text:
6.4.2 Get the character index

The cursor is represented as an integer index in the range [0 .. n] where n is the length of the text. Each letter has a different width. The list `self.char_positions` remembers the x position of each letter:

```python
def set_char_positions(self):
    """Get a list of all character positions."""
    self.char_positions = [0]
    for i in range(len(self.text)):
        w, h = self.font.size(self.text[:i+1])
        self.char_positions.append(w)
```

When we click with the mouse anywhere in the text, we need to know the character index:

```python
def get_char_index(self, position):
    """Return the character index for a given position."""
    for i, pos in enumerate(self.char_positions):
        if position <= pos:
            return i
    # if not found return the highest index
    return i
```

6.4.3 Move the cursor

The arrow keys allow to move the cursor to the left or to the right. The argument `d` is 1 or -1 and indicates the direction of movement. The cursor movement is limited to the interval [0 .. n]:

```python
def move_cursor(self, d):
    """Move the cursor by d characters, and limit to text length."""
    mod = pygame.key.get_mods()
    n = len(self.text)
    i = min(max(0, self.cursor+d), n)
```

Pressing the CMD key, the cursor goes all the way to the beginning or the end of the line:

```python
if mod & KMOD_META:
    if d == 1:
        i = n
    else:
        i = 0
```

Pressing the ALT key, the cursor goes to the end of the word:

```python
if mod & KMOD_ALT:
    while (0 < i < n) and self.text[i] != ' ':
        i += d
```
Pressing the SHIFT key prevents cursor2 from moving, thus setting a selection:

```python
if not mod & KMOD_SHIFT:
    self.cursor2 = i
self.cursor = i
```

### 6.4.4 Copy, cut and insert text

The two cursors can be inverted. The following method returns the two cursors (selection indices) in the right order:

```python
def get_selection_indices(self):
    '''Get ordered tuple of selection indicies.'''
    i = self.cursor
    i2 = self.cursor2

    if i < i2:
        return i, i2
    else:
        return i2, i
```

To copy text we save the selection in a Scene variable `text`:

```python
def copy_text(self):
    '''Copy text to Scene.text buffer.'''
    i, i2 = self.get_selection_indices()
    text = self.text[i:i2]
    App.scene.text = text
```

To cut text we copy the text and replace the selection with an empty string:

```python
def cut_text(self):
    '''Cut text and place copy in Scene.text buffer.'''
    self.copy_text()
    self.insert_text('')
```

To insert text we replace the current selection with the new text:

```python
def insert_text(self, text):
    '''Insert text at the cursor position or replace selection.'''
    i, i2 = self.get_selection_indices()
    text1 = self.text[:i]
    text2 = self.text[i2:]
    self.text = text1 + text + text2
    self.cursor = i + len(text)
    self.cursor2 = self.cursor
```

### 6.5 Buttons

The button class displays a text and executes a command upon a mouse-click
6.6 ListBox

The ListBox class displays a list of items. One item can be selected with a mouse-click or with the UP/DOWN arrow keys. Pressing the RETURN key executes the command.

6.7 Detecting double-clicks

In order to detect double-clicks or multiple clicks we need to use a timer event. The reason for using a timer is that we cannot know at the time of a mouse click if there are more clicks to follow. We only know for sure after a short timeout period. So we define a new event as the first USEREVENT:

```python
DBL_CLICK_TIMER = pygame.USEREVENT
DBL_CLICK_TIMEOUT = 250
```

Inside the `Scene.do_event()` we look for a MOUSEBUTTONDOWN event and we set a timer and increment the clicks:

```python
if event.type == MOUSEBUTTONDOWN:
    pygame.time.set_timer(DBL_CLICK_TIMER, DBL_CLICK_TIMEOUT)
    self.clicks += 1
```
Once the timeout occurs, we

- reset (disable) the timer
- print the number of clicks and
- reset the click count to zero:

```python
elif event.type == DBL_CLICK_TIMER:
    pygame.time.set_time(DBL_CLICK_TIMER, 0)
    print(self.clicks, 'clicks in', self.focus)
    self.clicks = 0
```

The text printed to the console looks like this:

- 2 clicks in Text0
- 4 clicks in Text0
- 3 clicks in Ellipse1
- 1 clicks in Rectangle2
- 2 clicks in None
7.1 Making sounds

The `pygame.mixer` module allows to play compressed OGG files or uncompressed WAV files.

This checks the initialization parameters and prints the number of channels available. It opens a sound object and plays it:

```python
print('init =', pygame.mixer.get_init())
print('channels =', pygame.mixer.get_num_channels())
App.snd = pygame.mixer.Sound('5_app/rpgaudio.ogg')
App.snd.play()
print('length =', App.snd.get_length())
```

Writes this to the console:

```
init = (22050, -16, 2)
channels = 8
length = 28.437868118286133
```

Here is a code example:

```python
"""Play a sound."""
from app import *

class Demo(App):
    def __init__(self):
        super().__init__()

        print('init =', pygame.mixer.get_init())
        print('channels =', pygame.mixer.get_num_channels())
        App.snd = pygame.mixer.Sound('5_app/rpgaudio.ogg')
        App.snd.play()
        print('length =', App.snd.get_length())
```

(continues on next page)
Scene(caption='Sound mixer')
Button('Stop', cmd='pygame.mixer.stop()')
Button('Pause', cmd='pygame.mixer.pause()')
Button('Unpause', cmd='pygame.mixer.unpause()')
Button('Fadeout', cmd='pygame.mixer.fadeout(5000)')
Button('Play', cmd='App.snd.play()')
Button('Volume 0.1', cmd='App.snd.set_volume(0.1)', pos=(200, 20))
Button('Volume 0.3', cmd='App.snd.set_volume(0.3)')
Button('Volume 1.0', cmd='App.snd.set_volume(1.0)')

if __name__ == '__main__':
    Demo().run()

Which produces the following result.
In this section we create the framework for board games. These games are based on a nxm grid. Each cell can have

- text
- color
- image
8.1 Selecting cells with the mouse

Board
click to select
cmd+click multiple
arrow to move

8.2 Adding background color

Color
Add random colors
### 8.3 Create a checkerboard pattern

#### Checker

Create a pattern

```
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
```
Sphinx is a tool for making documentation. It was originally created for the Python documentation, but is now used for many other software projects.

Sphinx uses reStructuredText as its markup language. It can produce HTML, LaTeX, ePub and PDF documents.

Source: https://www.sphinx-doc.org

## 9.1 Getting started

After installation, you can get started quickly with the tool `sphinx-quickstart`. Just enter:

```
sphinx-quickstart
```

Answer each customization question with yes or no. Be sure to say yes to the `autodoc` extension. The `sphinx-quickstart` creates a directory with several documents:

- `conf.py` file, the default configuration file
- `index.rst` file, the master document

The `conf.py` file lets you configure all aspects of Sphinx. The `index.rst` is the entry page for your documentation. It contains the `toctree` directive which determines the files to include. For this project it looks like this:

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:

   1_intro/intro
   2_draw/draw
   3_image/image
   ...
```

To build the HTML pages just run:
9.2 reStructuredText

reStructuredText (.rst) is the default markup language used with Sphinx. It is important to know that:

• paragraphs are separated by one or more blank lines
• indentation is significant

9.2.1 Inline styles

Inside text you can use:

• one asterisk for *italics*
• two asterisks for **bold**
• backquotes for `code`

9.2.2 Lists

This code:

```
* This is a bulleted list.
* It has two items, the second item uses two lines.
#. This is a numbered list.
#. It has two items too.
```

produces this result:

• This is a bulleted list.
• It has two items, the second item uses two lines.
  1. This is a numbered list.
  2. It has two items too.

9.2.3 Hyperlinks

This code:

```
`Source <https://www.sphinx-doc.org>`_
```

produces Source
9.2.4 Admonitions

**Danger:** Be careful with this code!

**Tip:** Be careful with this code!

**Warning:** Be careful with this code!

9.2.5 Footnotes

This is a footnote\(^1\) inside a text, this is another one\(^2\).

9.2.6 Horizontal list

To add a horizontal list add this code:

```
.. hlist::
   :columns: 3

* happy
...
```

- happy
- short
- intelligent
- thankful
- displayed
- horizontal

9.2.7 Download

To add a download link add this code:

```
:download:`requirements.txt<requirements.txt>`.
```

requirements.txt.

9.3 Include from a file

It is possible to include a Python object (class, method) from a file. For example you can include a `class` definition with:

---
\(^1\) Text of the first footnote
\(^2\) Text of the second footnote
resulting in

```python
class Rectangle(Node):
    """Draw a rectangle on the screen."""
    options = { 'fg': Color('green'),
               'bg': Color('black'),
               'thickness': 2}
    
def __init__(self, **options):
        super().__init__(**options)
        self.set_options(Rectangle, options)
        self.render()

    def render(self):
        self.img0 = pygame.Surface(self.rect.size, flags=SRCALPHA)
        if self.fg != None:
            pygame.draw.rect(self.img0, self.fg, Rect(0, 0, *self.rect.size), 0)
        pygame.draw.rect(self.img0, self.bg, Rect(0, 0, *self.rect.size), self.
                        -->thickness)
        self.img = self.img0.copy()
```

Or you can include just a `method` definition with:

```python
.. literalinclude:: 5_app/app.py
    :pyobject: Rectangle.render
```

resulting in

```python
def render(self):
    self.img0 = pygame.Surface(self.rect.size, flags=SRCALPHA)
    if self.fg != None:
        pygame.draw.rect(self.img0, self.fg, Rect(0, 0, *self.rect.size), 0)
    pygame.draw.rect(self.img0, self.bg, Rect(0, 0, *self.rect.size), self.
                    -->thickness)
    self.img = self.img0.copy()
```

### 9.4 Directives

A directive consists of

- name
- arguments
- options
- content

The structure is this:
9.4.1 Function

This directive defines a function:

```plaintext
.. function:: spam(eggs)
   ham(eggs)

   Spam ham the are made with a certain number of eggs.
```

```python
spam(eggs)
ham(eggs)
```

Spam and ham the are made with a certain number of eggs.

To cross-reference you can use:

- `method_name()` with `:meth:`method_name`
- `class_name` with `:class:`class_name`
- `function_name()` with `:func:`function_name`

For example with `:func:`spam` one can reference the above functions `spam()` or `ham()` inside a sentence..

9.4.2 Data

To describe global data and constants in a module use this code:

```plaintext
.. data:: number=1000

   Describe data.
```

```python
number=1000
```

Describe data.

9.4.3 Class

```python
class App

   Describe class without parameters.

   run()

   Describe the method.
```

```python
class App(parameters)

   Describe class with parameters.

   objects

   Global class attribute.
```
9.4.4 Functions with arguments

send_message(sender, recipient, message_body[, priority=1])

Send a message to a recipient

Parameters

• sender (str) – The person sending the message
• recipient (str) – The recipient of the message
• message_body (str) – The body of the message
• priority (integer or None) – The priority of the message, can be a number 1-5

Returns the message id

Return type int

Raises

• ValueError – if the message_body exceeds 160 characters
• TypeError – if the message_body is not a basestring

9.5 Math formulas

Since Pythagoras, we know that \( a^2 + b^2 = c^2 \).

\[ e^{i\pi} + 1 = 0 \] (9.1)

Euler’s identity, equation (9.1), was elected one of the most beautiful mathematical formulas.

9.6 The app module

This code:

```plaintext
.. automodule:: app
   :members:
   :member-order: bysource
```

Prints the whole app documentation and lists members by source order.

App - there is only one App object - an app has multiple scenes (App.scenes) - an app has one current scene (App.scene) - an app has one window to draw in (App.screen)

Scene

• a scene has multiple nodes (App.scene.nodes)
• nodes are ordered: the last in the list is displayed last
• the node which is clicked becomes active
• the active node becomes the top node
• the active node has focus (App.scene.focus)
• TAB and shift-TAB select the next node
Node (object)

- nodes have default position and size (pos, size)
- nodes are automatically placed at creation (dir, gap)
- nodes inherit options (color, size, . . . ) from the previous object

A Node object has the following properties

- clickable: mouse-click has effect
- movable: can be moved (mouse, arrow-keys)
- visible: is drawn
- has focus

Debug

- print events to console (cmd+E)
- display node label (cmd+L)
- display outline (cmd+O)

```python
class app.App(size=(640, 240), shortcuts={})

Create a single-window app with multiple scenes having multiple objects.

run()

Run the main event loop.

next_scene(d=1)

Switch to the next scene.

do_shortcut(event)

Find the key/mod combination in the dictionary and execute the cmd.

capture()

Save a screen capture to the directory of the calling class, under the class name in PNG format.

toggle_fullscreen()

Toggle between full screen and windowed screen.

toggle_resizable()

Toggle between resizable and fixed-size window.

toggle_frame()

Toggle between frame and noframe window.
```

class app.Scene(caption='Pygame', remember=True, **options)

Create a new scene and initialize the node options.

load_img(file)

Load the background image.

enter()

Enter a scene.

update()

Update the nodes in a scene.

set_status(txt)

Set status text and render it.

render_status()

Render the status text.

9.6. The app module
**draw**

Draw all objects in the scene.

**do_event** *(event)*

Handle the events of the scene.

**next_focus** *(d=1)*

Advance focus to next node.

**cut**

Cuts the selected objects and places them in App.selection.

**copy**

Copies the selected objects and places them in App.selection.

**paste**

Pastes the objects from App.selection.

**debug**

Print all scene/node options.

```python
class app.Node(**options)**

Create a node object with automatic position and inherited size.

**set_options** *(cls, options)*

Set instance options from class options.

**create_img**

Create the image surface, and the original img0.

**color_img**

Add background color to the image.

**set_background** *(img)*

Set background color or transparency.

**load_img**

Load the image file.

**calculate_pos** *(options)*

Calculate the next node position.

**render_label**

Create and render the node label.

**do_event** *(event)*

React to events happening for focus node.

**draw**

Draw the node and optionally the outline, label and focus.

```python
class app.TextObj(text='Text', **options)**

Create a text surface image.

**set_font**

Set the font and its properties.

**render_text**

Render the text into an image.

```python
class app.Text(text='Text', **options)**

Create a text object horizontal and vertical alignment.

```python
class app.TextLines(text, **options)**

Create a text object horizontal and vertical alignment.
```python
class app.EditableTextObj (text=’EditableText’, cmd=’’, **options)
    Create keyboard and mouse-editable text with cursor and selection.

    set_char_positions()  
        Make a list of all character positions.

    get_char_index(position)  
        Return the character index for a given position.

    move_cursor(d)  
        Move the cursor by d characters, and limit to text length.

    get_selection()  
        Get ordered tuple of selection indices.

    copy_text()  
        Copy text to Scene.text buffer.

    cut_text()  
        Cut text and place copy in Scene.text buffer.

    insert_text(text)  
        Insert text at the cursor position or replace selection.

    select_word()  
        Select word at current position.

    select_all()  
        Select the whole text.

    do_event(event)  
        Move cursor, handle selection, add/backspace text, copy/paste.

    render()  
        Render cursor, selection and text to an image.

class app.EditableText (text=’CursorText’, **options)
    Create an editable text node.

    do_event(event)  
        React to events happening for focus node.

    draw()  
        Draw the node and optionally the outline, label and focus.

    double_click()  
        Select the current word.

class app.Button (text=’Button’, cmd=’’, **options)
    Create a button object with command.

    do_event(event)  
        React to events happening for focus node.

class app.Toggle
    Add toggle button behavior.

class app.Checkbox (**options)
class app.Radiobutton (**options)
class app.ListBox (items, i=0, **options)
    Show a list of text items.
```

9.6. The app module
**set_list**(*items*)
Set items and selection list.

**scroll**(*d*)
Scroll listbox up and down.

**move_cursor**(*d*)
Move the active cell up or down.

**do_event**(*event*)
React to events happening for focus node.

**class** *app.ListMenu*(*items, **options*)
Display a drop-down menu.

**class** *app.SliderObj(**options*)
Define a slider object.

**class** *app.Slider(**options*)

  **do_event**(*event*)
  React to events happening for focus node.

**class** *app.Spinbox(**options*)
Input a number.

  **do_event**(*event*)
  React to events happening for focus node.

**class** *app.Rectangle(**options*)
Draw a rectangle on the screen.

**class** *app.Ellipse(**options*)
Draw an ellipse on the screen.

**class** *app.Board(**options*)
Draw a mxn board grid with m lines and n columns. m, n number of cells (row, col) i, j index of cell (row, col) dx, dy size of cell Num numeric matrix Num0 initial numeric matrix Col color matrix

  **set_Num**(*s*)
  Load Num table from a string.

  **render_colors**()
  Render the background colors.

  **render_grid**()
  Render the grid lines.

  **render_num**()
  Draw number.

  **render_tile**()
  Draw number.

  **render**()
  Render the whole board.

  **get_index**(*x, y*)
  Get index (i, j) from mouse position (x, y).

  **do_event**(*event*)
  React to events.
class app.Sudoku(**options)
    Create a sudoku game board.
    
    render_grid()
    Override the grid lines.

class app.Chess(**options)
    Create a sudoku game board.

class app.Go(**options)
    
    render()
    Render the Go board and add extra dots on certain intersections.

class app.Puzzle(div=(3, 3), **options)
    Take an image and create a puzzle.

9.7 Glossary

reStructuredText  reStructuredText is light-weight markup language.
CHAPTER 10

Indices and tables

- genindex
- modindex
- search
Python Module Index

a
  app, 70
Index

A
App (built-in class), 69
App (class in app), 71
app (module), 70

B
Board (class in app), 74
Button (class in app), 73

C
calculate_pos () (app.Node method), 72
capture() (app.App method), 71
Checkbox (class in app), 73
Chess (class in app), 75
color_img () (app.Node method), 72
copy () (app.Scene method), 72
copy_text () (app.EditableTextObj method), 73
create_img() (app.Node method), 72
cut () (app.Scene method), 72
cut_text() (app.EditableTextObj method), 73

D
debug() (app.Scene method), 72
do_event () (app.Board method), 74
do_event () (app.Button method), 73
do_event () (appEditableText method), 73
do_event () (appEditableText method), 73
do_event () (app.ListBox method), 74
do_event () (app.Node method), 72
do_event () (app.Scene method), 72
do_event () (app.Slider method), 74
do_event () (app.Spinbox method), 74
doShortcut () (app.App method), 71
double_click() (app.EditableText method), 73
draw() (app.EditableText method), 73
draw() (app.Node method), 72
draw() (app.Scene method), 71

E
EditableText (class in app), 73
EditableTextObj (class in app), 72
Ellipse (class in app), 74
enter() (app.Scene method), 71

G
get_char_index() (appEditableTextObj method), 73
get_index() (app.Board method), 74
get_selection() (appEditableTextObj method), 73
Go (class in app), 75

H
ham() (built-in function), 69

I
insert_text() (appEditableTextObj method), 73

L
ListBox (class in app), 73
ListMenu (class in app), 74
load_img() (app.Node method), 72
load_img() (app.Scene method), 71

M
move_cursor() (appEditableTextObj method), 73
move_cursor () (app.ListBox method), 74

N
next_focus() (app.Scene method), 72
next_scene () (app.App method), 71
Node (class in app), 72

O
objects (App attribute), 69

P
paste() (app.Scene method), 72
Puzzle (class in app), 75
R
Radiobutton (class in app), 73
Rectangle (class in app), 74
render() (app.Board method), 74
render() (app.EditableTextObj method), 73
render() (app.Go method), 75
render_colors() (app.Board method), 74
render_grid() (app.Board method), 74
render_grid() (app.Sudoku method), 75
render_label() (app.Node method), 72
render_num() (app.Board method), 74
render_status() (app.Scene method), 71
render_text() (app.TextObj method), 72
reStructuredText, 75
run() (App method), 69
run() (app.App method), 71

S
Scene (class in app), 71
scroll() (app.ListBox method), 74
select_all() (app.EditableTextObj method), 73
select_word() (app.EditableTextObj method), 73
send_message() (built-in function), 70
set_background() (app.Node method), 72
set_char_positions() (app.EditableTextObj method), 73
set_font() (app.TextObj method), 72
set_list() (app.ListBox method), 73
set_Num() (app.Board method), 74
set_options() (app.Node method), 72
set_status() (app.Scene method), 71
Slider (class in app), 74
SliderObj (class in app), 74
spam() (built-in function), 69
Spinbox (class in app), 74
Sudoku (class in app), 74

T
Text (class in app), 72
TextLines (class in app), 72
TextObj (class in app), 72
Toggle (class in app), 73
toggle_frame() (app.App method), 71
toggleFullscreen() (app.App method), 71
toggle_resizable() (app.App method), 71

U
update() (app.Scene method), 71