CS 11 Exercise 09
Introduction to API Programming and Event Handling

University of the Philippines Diliman

November 2018

1 Instructions

- For this exercise, you have to write three different computer programs. The other problems will be uploaded soon.
- Present your output to your instructor to get a grade on this exercise.
- Submit your solutions on or before Friday, December 07 at 5:30pm.

2 Preliminaries

2.1 Programming with an API

An API, or an *application programming interface*, is a set of routines, protocols, and tools for building software applications. APIs are often used by programmers who work on applications that communicate with other applications such as graphics drivers, operating systems, databases, networks, etc. An API, as the name implies, provides an *interface* that the programmer can use to make their programs communicate to other applications. These interfaces are in the form of software: functions, objects, variables, and so on.

Programming with an API provides a lot of benefits:

1. APIs provide an abstraction to the programmer, letting the programmer spend more time on formulating solutions faster and safer rather than minding the details of implementation.
2. APIs prevent the programmer from programming directly on critical software such as operating systems and databases.
3. APIs prevent the programmer from reinventing the wheel and instead focus on the problem at hand.

and the list goes on.

For this exercise, your goal is to create programs that will use the `Pyglet` API.

2.2 Pyglet

To get a quick overview on Pyglet you can check their website: https://pyglet.readthedocs.io/en/pyglet-1.3-maintenance/

2.2.1 Getting Pyglet

Pyglet 1.3.2 is already installed in the laboratory computers. To install pyglet on your own machine, you can follow the installation instructions here.

2.3 Hello World!

Copy the following code snippet in your editor and save it as `helloworld.py`. This will be your first Pyglet program.

```
import pyglet

window = pyglet.window.Window()
label = pyglet.text.Label('Hello, world',
                font_name='Times New Roman',
                font_size=36,
                x=window.width // 2, y=window.height // 2,
                anchor_x='center', anchor_y='center')

@window.event
def on_draw():
    window.clear()
    label.draw()

pyglet.app.run()
```

2.4 Running the script

To run the program, one approach is by opening the terminal and typing

```
python3 helloworld.py
```

You can also run the program by just double clicking on `helloworld.py`. On linux, you can do the following:

1. Add a "hashbang" at the start of your program.

```
#! /usr/bin/python3
import pyglet
```

2. Enter the following on the terminal:

```
chmod a+x helloworld.py
```

When run, you should see something that looks like this.

2.5 Events and Event Programming

One of the most popular ways to program graphical interfaces is through event programming. In event programming, program behavior is determined by events

such as mouse clicks, keyboard presses, moving the window, etc. Pyglet provides several facilities to programmers for handling events.

In our hello world program, we start by importing the `pyglet` module. We then create a window by calling `pyglet.window.Window()`. The `window` object contains attributes such as `window.width` and `window.height`. We will then put some text in the window through a `pyglet.text.Label` object.

Once the window is drawn (displayed), we will instruct Python to clear the window and draw the label through the `on_draw` function. We call the `on_draw` function an event handler. Event handlers are pieces of code that are executed when certain events happen. On `on_draw`, the event is the window is being redrawn. Drawing the window happens at the start of the program, and is repeated every time another event occurs.

To complete the process of handling the `on_draw` event, we have to attach it to an `EventDispatcher`, which are objects that produce events. In our `helloworld` program, the `EventDispatcher` is the `window`, and we will handle one of the its events through an event handler (`on_draw` function). To attach our event handler to the event dispatcher, we will decorate `on_draw` with the `window.event decorator`. All event dispatchers in Pyglet have a decorator for attaching events.

Finally, we will run the application by calling `pyglet.app.run()`. This function starts the event loop, where the program waits for events. The `on_draw` function is called in the event loop. Our program will only terminate if all the windows are closed, or when `pyglet.app.exit()` is called, or when Python gets terminated.

2.5.1 Displaying Windows and Text

As a graphics module, Pyglet supports windowing. To create windows on Pyglet use the `pyglet.window.Window()` object. We display text on our windows by creating labels through `pyglet.text.Label` and drawing that label. By defatult, the dimensions of the window is 640 by 480, but we can modify it by passing the width and height arguments.

```
import pyglet

window = pyglet.window.Window(width=700, height=700)
label = pyglet.text.Label('Sample Label',
                          font_name='Times New Roman',
                          font_size=48,
                          x=window.width//2, y=window.height//2,
```

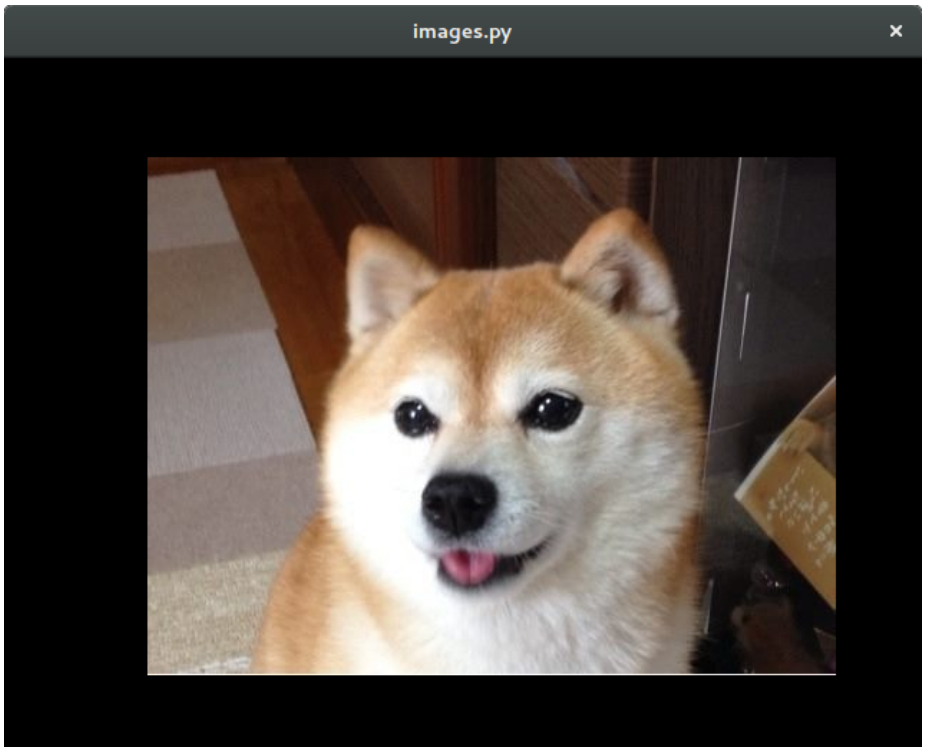
```
                        anchor_x='center', anchor_y='center')

@window.event
def on_draw():
    window.clear()
    label.draw()

pyglet.app.run()
```

2.5.2 Images and Sprites

There are two ways to display images. One is by loading images. On the following code snippet, the image is loaded using `pyglet.image.load`. The image is then displayed using the `blit` method. the `blit` method accepts the x and y coordinates of the lower-left corner of the image, where $(0, 0)$ is the lower-left corner of the window.

```
import pyglet

window = pyglet.window.Window()
doge = pyglet.image.load('doge.jpg')

@window.event
def on_draw():
    window.clear()
    doge.blit(100,50)

pyglet.app.run()
```

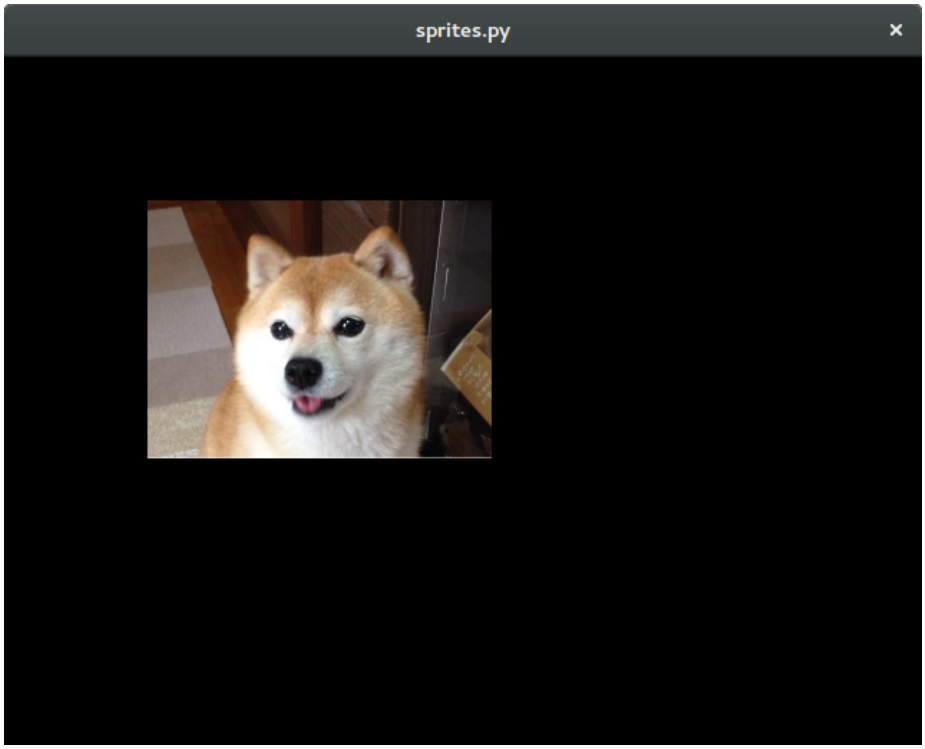
When run, you should see something that looks like this.



Another way to display images is through sprites. This approach is more flexible than the previous one. On sprites, you can easily modify the scale and the position of the image, which will be useful when you want your images to move.

```
import pyglet

window = pyglet.window.Window()
doge = pyglet.image.load('doge.jpg')

dogesprite = pyglet.sprite.Sprite(doge)
dogesprite.scale = 0.5
dogesprite.position = (100,200)

@window.event
def on_draw():
    window.clear()
    dogesprite.draw()

pyglet.app.run()
```

When run, you should see something that looks like this.



2.5.3 Moving images

To animate a moving image, we have to update its position periodically. To do this, we have to create an `update` function and schedule it using `pyglet.clock.schedule_interval`.

```
import pyglet

window = pyglet.window.Window()
doge = pyglet.image.load('doge.jpg')

dogesprite = pyglet.sprite.Sprite(doge)
dogesprite.scale = 0.5
dogesprite.position = (0,0)

def update(dt):
    dogesprite.x += 100 * dt
    dogesprite.y += 100 * dt
```

```
@window.event
def on_draw():
    window.clear()
    dogesprite.draw()

pyglet.clock.schedule_interval(update, 1/60)
pyglet.app.run()
```

2.5.4 Mouse and Keyboard Events

The following program show how mouse and keyboard events are handled:

```
import pyglet
from pyglet.window import key
from pyglet.window import mouse

window = pyglet.window.Window(width = 900)
key_label = pyglet.text.Label('',
                          font_name='Times New Roman',
                          font_size=48,
                          x=window.width//2, y=window.height//2,
                          anchor_x='center', anchor_y='center')

mouse_label = pyglet.text.Label('',
                          font_name='Times New Roman',
                          font_size=32,
                          x=window.width//2, y=0,
                          anchor_x='center', anchor_y='bottom')

mouse_hover_label = pyglet.text.Label('',
                          font_name='Times New Roman',
                          font_size=32,
                          x=window.width//2, y=window.height,
                          anchor_x='center', anchor_y='top')


@window.event
def on_key_press(symbol, modifiers):
    if symbol == key.A:
        key_label.text = 'The "A" key was pressed.'
    elif symbol == key.LEFT:
        key_label.text = 'The left arrow key was pressed.'
```

```
    elif symbol == key.ENTER:
        key_label.text = 'The enter key was pressed.'

@window.event
def on_mouse_press(x, y, button, modifiers):
    if button == mouse.LEFT:
        mouse_label.text = 'The left mouse button was pressed at ({},{}).'.format(x,y)

@window.event
def on_mouse_motion(x, y, button, modifiers):
    mouse_hover_label.text = '({},{})'.format(x,y)

@window.event
def on_draw():
    window.clear()
    key_label.draw()
    mouse_label.draw()
    mouse_hover_label.draw()

window.push_handlers(pyglet.window.event.WindowEventLogger())
pyglet.app.run()
```

On the program, we added three more event handlers. The handlers `on_key_press`, `on_mouse_press`, `on_mouse_motion`, handles key presses, mouse presses, and mouse motion respectively.

2.5.5 Sounds

You can also play sounds using Pyglet. Please check the documentation for details.

2.5.6 Other events and event dispatchers

To capture all the events dispatched by your window, you can add attach `WindowEventLogger` to your window. This event handler logs all the events dispatched by the window.

```
window.push_handlers(pyglet.window.event.WindowEventLogger())
```

Handling these events are similar to handling the draw event, where we have to create an event handler and attach it to the event dispatcher. A brief description of all the window events is found in the documentation.

2.6 References

Much of the work done when programming with APIs is reading documentation. A comprehensive documentation of Pyglet is located at here

2.6.1 Additional Materials

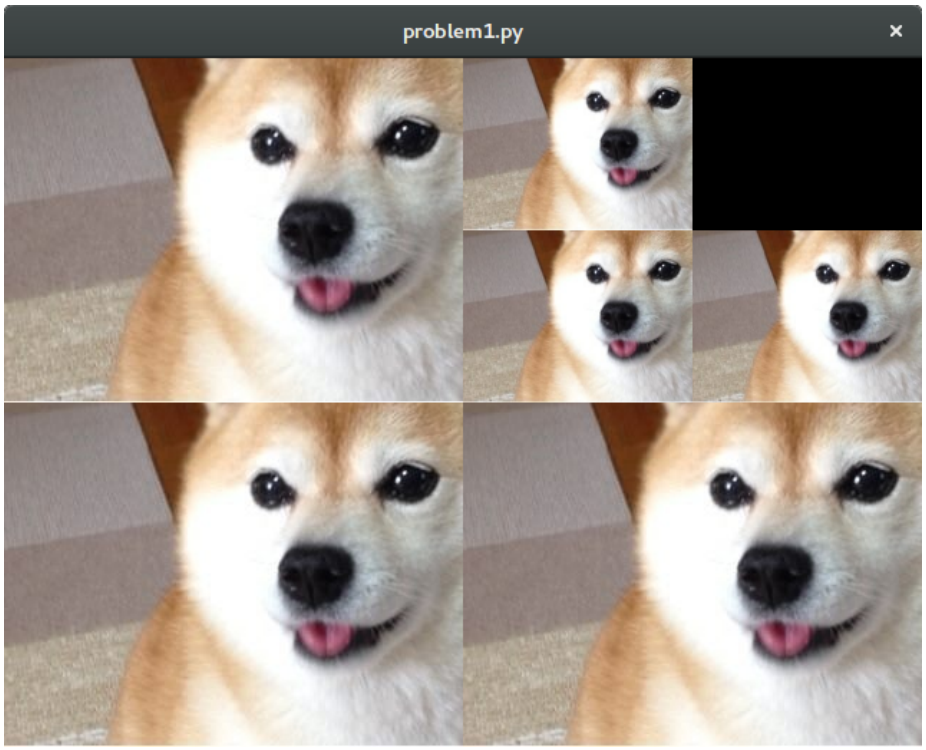
You can find additional source files in UVLe for examples. For Problem 2.7.2, `batch.py` will show you an easier way to draw several sprites at once. For more examples of animated sprites, you can check the files at the `dragonfly` folder. You can also try the examples found in the documentation.

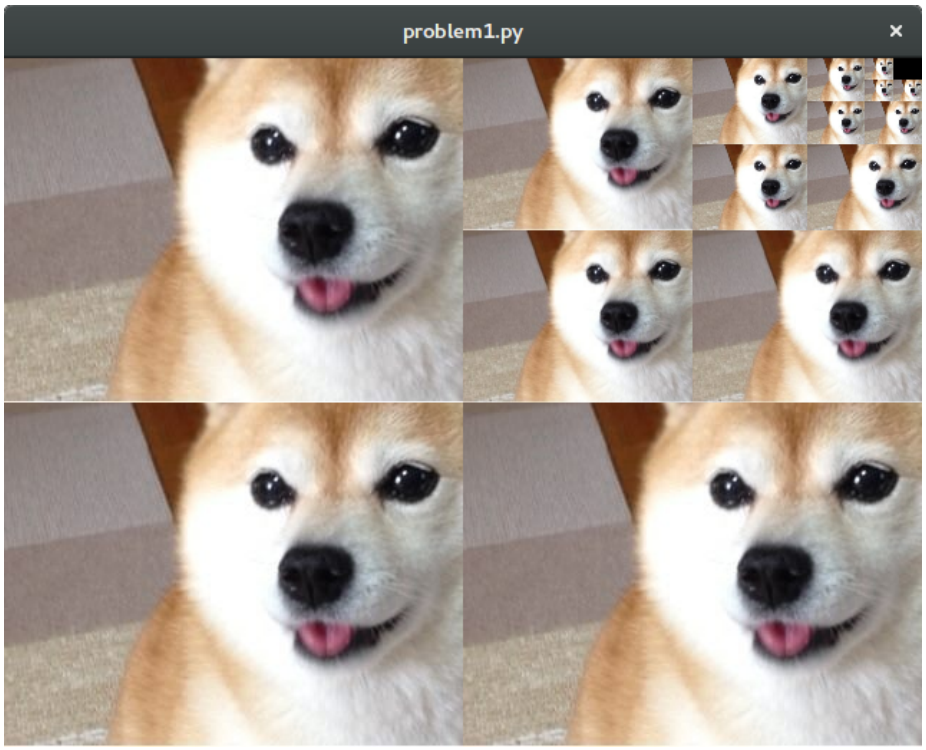
2.7 Problems

The following problems will require you to load images on Pyglet. Some images are available along with the sample code, but you can use any image on these problems.

2.7.1 Image Tiling (5 points)

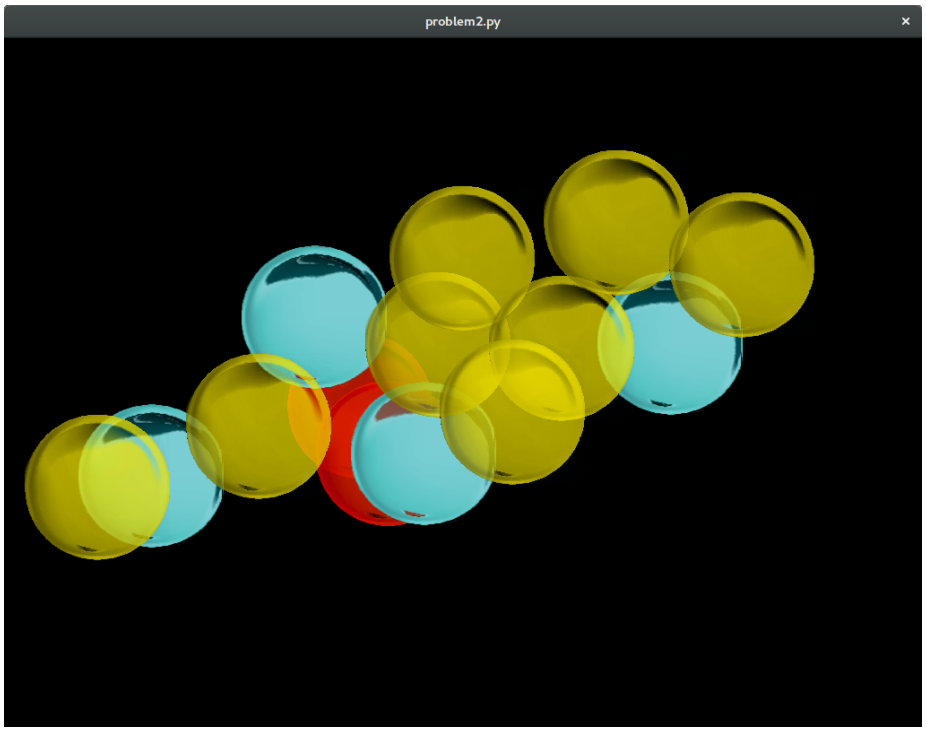
Create a Pyglet program that tiles an image in three quadrants. The pattern must repeat on every upper-right quadrant, and the number of repetitions must be based on command-line argument $n >= 1$. You can find the command-line arguments from the `sys.argv` list of the `sys` module. The images below show the tiling for $n = 2$ and $n = 5$.


problem1.py



2.7.2 Colored Bubbles (5 points)

Create a Pyglet program that generates a bubble every time a user clicks. The bubble must be generated right where the user clicks in the window, and the color of the bubble must be randomly selected between red, blue and yellow. Upon generation, the bubble move in a random direction, and it must bounce back when it hits the boundary of the window. A sample snapshot of the application is shown below:



Hint: Each bubble has their own sprite, position, and direction. For all bubbles to move, each bubble must be updated periodically. You might consider using classes to represent a bubble. With this approach, the position, movement direction, and sprite will become attributes of the bubble. Since each bubble must be updated, you might also consider putting the update function as a bubble method.