Python Classes

- you can use the `class` keyword to create your own classes

- here is a tiny example of a class

Python Classes

tinyclass.py

```python
#!/usr/bin/env python3

import math

class vector:
    def __init__ (self, pair):
        self.pair = pair
    def get_x (self):
        return self.pair[0]
    def get_y (self):
        return self.pair[1]
    def get_pair (self):
        return self.pair
    def get_length (self):
        return math.sqrt(sqr(self.pair[0]) + sqr(self.pair[1]))
```

Python Classes

■
```
def sqr (x):
    return x*x


v = vector ([3, 4])
print ("using the Pythagorean theory the hypotenuse", end=' ')
print ("(or magnitude) of the vector", end=' ')
print (v.get_pair(), "=", v.get_length())
```

Extending last weeks tutorial to use classes

■ create a ball class with a constructor which has an initial position and a velocity vector

Extending last weeks tutorial to use classes

```python3
#!/usr/bin/env python3

import sys, pygame, random

class ball:
    def __init__ (self, initial_position, velocity):
        self.curpos = initial_position
        self.velocity = velocity
        self.ball = pygame.image.load("ball.png").convert ()
        self.ballrect = self.ball.get_rect ()
        self.ballrect = self.ballrect.move (initial_position)


    def update (self, surface):
        self.ballrect = self.ballrect.move (self.velocity)
        if self.ballrect.left < 0 or self.ballrect.right > width:
            self.velocity[0] = -self.velocity[0]
        if self.ballrect.top < 0 or self.ballrect.bottom > height:
            self.velocity[1] = -self.velocity[1]
        surface.blit (self.ball, self.ballrect)
```

Extending last weeks tutorial to use classes

```
width = 1900
height= 1000
black = (0, 0, 0)

pygame.init ()
screen = pygame.display.set_mode([width, height])


bl = []
for i in range (80):
    bl += [ball ([random.randint (0, width-1),
                  random.randint (0, height-1)],
                 [random.randint(-2, 2),
                  random.randint(-2, 2)])]
```

Extending last weeks tutorial to use classes

■
```
while True:
    for event in pygame.event.get ():
        if event.type == pygame.QUIT:
            sys.exit(0)

    screen.fill (black)
    for b in bl:
        b.update (screen)
    pygame.display.flip()
```

PyGame Sprites

■ these notes show how sprites can be created

   ■ without object orientation

   ■ and also with object orientation

■ they re-implement the bouncing ball demo which we covered in earlier weeks

   ■ using sprites rather than images

PyGame Sprites

- sprites are quite complicated
  - in my experience this is due to their close association with object orientation
  - you can use sprites without object orientation
  - for a small number of sprites this is easy enough and the code is smaller than its object orientated counterpart (and much simpler)
  - for any reasonable number of sprites >= 3 then should use OO as the reduction in code probably offsets the OO complication

- use object orientated sprites when implementing Missile Command

When to use sprites in PyGame

- when you have:
    - many instances of an object on the screen at a time
    - some objects that you need to track closely (e.g. collision detection)
        - sprites have a `self.rect` attribute, which can be passed to the function `colliderect` so Pygame will handle all collisions
    - a sprite's `update()` method, with a time argument, makes it easy to deal with a dynamic environment

- you can easily kill sprites if collisions occur (or create sounds)
    - sprites can be thought of as semi autonomous

When not to use sprites

- when your:
  - objects don't share much (if any) code
    - and if you rarely have more than one copy of each object instantiated at a time
  - game entities (images) never move *by themselves* (eg. card decks)

- simple user interfaces are often easier to do with surfaces than with sprites

Creating a simple sprite

```python
#!/usr/bin/python3

import pygame
from pygame.locals import KEYDOWN


width  = 320
height = 240
size   = [width, height]
pygame.init()
screen = pygame.display.set_mode(size)
background = pygame.Surface(screen.get_size())


b = pygame.sprite.Sprite() # create sprite
b.image = pygame.image.load("ball.png").convert() # load ball image
b.rect = b.image.get_rect() # use image extent values
b.rect.topleft = [0, 0] # put the ball in the top left corner
screen.blit(b.image, b.rect)


pygame.display.update()
while pygame.event.poll().type != KEYDOWN:
    pygame.time.delay(100)
```

The sprite in PyGame

- is an object that contains both:
  - an image (a surface)
  - and a location at which to draw that image (a Rect)

- term *sprite* is actually a holdover from older display systems that did such manipulations directly in hardware
  - Commodore 64, Commodore Amiga used this technique in early 1980s to early 1990s
  - other manufactures did exactly the same, Atari etc

The sprite in PyGame

- sprites work well in object-oriented languages like Python
  - you have a standard sprite interface `pygame.sprite.Sprite`, and extend those classes as specific sprites

- see the `BallSprite` class in the example later on

The sprite in PyGame

- sprites have two important instance variables
    - `self.image` and `self.rect`
    - `self.image` is a surface, which is the current image that will be displayed. self.rect is the location at which this image will be displayed when the sprite is drawn to the screen

- sprites also have one important instance method, `self.update.`

Creating a simple sprite using an extra Class

```python
#!/usr/bin/env python3

import pygame
from pygame.locals import KEYDOWN

class BallSprite(pygame.sprite.Sprite):
    image = None

    def __init__(self, location):
        pygame.sprite.Sprite.__init__(self)

        if BallSprite.image is None:
            # This is the first time this class has been
            # instantiated. So, load the image for this and
            # all subsequence instances.
            BallSprite.image = pygame.image.load("ball.png")
        self.image = BallSprite.image

        # Make our top-left corner the passed-in location.
        self.rect = self.image.get_rect()
        self.rect.topleft = location
```

Creating a simple sprite using an extra Class

```
pygame.init()
screen = pygame.display.set_mode([320, 320])
b = BallSprite([0, 0]) # put the ball in the top left corner
screen.blit(b.image, b.rect)
pygame.display.update()
while pygame.event.poll().type != KEYDOWN:
    pygame.time.delay(10)
```

Bouncing ball using sprites and no user defined classes

```python
#!/usr/bin/env python3

import pygame
from pygame.locals import KEYDOWN

width  = 320
height = 240
size   = [width, height]
ydir = 1
xdir = 1
xpos = 0
ypos = 0
pygame.init()
screen = pygame.display.set_mode(size)
background = pygame.Surface(screen.get_size())

b = pygame.sprite.Sprite() # create sprite
b.image = pygame.image.load("ball.png").convert()  # load image
b.rect = b.image.get_rect() # use image extent values
b.rect.topleft = [xpos, ypos] # put the ball in the top left corner
screen.blit(b.image, b.rect)
slow = 0
```

Bouncing ball using sprites and no user defined classes

```
def gravity(y):
    global height
    return int (((height+height/20) * 3) / y)

pygame.display.update()
while pygame.event.poll().type != KEYDOWN:
    pygame.time.delay(gravity(ypos))
    # If we're at the top or bottom of the screen,
    # switch directions.

    if b.rect.bottom>=height:
        ydir = -1
    elif ypos == 0:
        ydir = 1
    if xpos == 0:
        xdir = 1
    elif b.rect.right>=width:
        xdir = -1
```

Bouncing ball using sprites and no user defined classes

```python
    if slow:
        screen.fill([0, 0, 0]) # blank the screen
    else:
        rectlist = [screen.blit(background, b.rect)]

    # Move our position up or down by one pixel
    xpos += xdir
    ypos += ydir
    b.rect.topleft = [xpos, ypos]

    if slow:
        screen.blit(b.image, b.rect)
        pygame.display.update()
    else:
        rectlist += [screen.blit(b.image, b.rect)]
        pygame.display.update(rectlist)
```

Bouncing ball using sprites and no user defined classes

■
```
#!/usr/bin/env python3

import pygame
from pygame.locals import KEYDOWN

class BallSprite(pygame.sprite.Sprite):
    image = None

    def __init__(self, initial_position):
        pygame.sprite.Sprite.__init__(self)
        if BallSprite.image is None:
            BallSprite.image = pygame.image.load("ball.png")
        self.image = BallSprite.image

        self.rect = self.image.get_rect()
        self.rect.topleft = initial_position
        self.going_down = True # Start going downwards
        self.next_update_time = 0 # update() hasn't been called yet.
```

Bouncing ball using sprites and no user defined classes

```
def update(self, current_time, bottom):
    # Update every 10 milliseconds = 1/100th of a second.
    if self.next_update_time < current_time:

        # If we're at the top or bottom of the screen, switch directions.
        if self.rect.bottom == bottom - 1: self.going_down = False
        elif self.rect.top == 0: self.going_down = True

        # Move our position up or down by one pixel
        if self.going_down: self.rect.top += 1
        else: self.rect.top -= 1

        self.next_update_time = current_time + 10
```

Bouncing ball using sprites and no user defined classes

```
pygame.init()
boxes = []
for location in [[0, 0],
                 [60, 60],
                 [120, 120]]:
    boxes.append(BallSprite(location))

screen = pygame.display.set_mode([150, 150])
while pygame.event.poll().type != KEYDOWN:
    screen.fill([0, 0, 0]) # blank the screen.

    time = pygame.time.get_ticks()
    for b in boxes:
        b.update(time, 150)
        screen.blit(b.image, b.rect)
    pygame.display.update()
```

Bouncing ball using sprites and no user defined classes

■ it is worth noting that the OO solution uses processor resources efficiently

The event loop

■
```
while True:
    event = pygame.event.wait()
    if event.type == pygame.QUIT:
        sys.exit(0)
    if event.type == KEYDOWN:
        if event.key == K_ESCAPE:
            sys.exit (0)
```

■ consider the above section of code

■ it waits for an event to occur and then acts upon the event

■ an event will be mouse movement, mouse click, key up/down etc

■ what happens if there is no event present?

The event loop

- various event retrieval mechanisms in Pygame
    - read the documentation for ideas

- how do we handle the problem of no events occuring?

USEREVENTS

- one mechanism is to poll the event queue
    - this is a lazy programming mechanism and cpu intensive

- a better solution is to introduce `USEREVENTs`

USEREVENT example code (snippet)

■

```
def updateAll ():
    if allExplosions != []:
        for e in allExplosions:
            e.update ()
        pygame.display.flip ()
        pygame.time.set_timer (USEREVENT+1, delay)
```

USEREVENT example code (snippet)

■

```python
def wait_for_event ():
    global screen
    while True:
        event = pygame.event.wait ()
        if event.type == pygame.QUIT:
            sys.exit(0)
        if event.type == KEYDOWN and event.key == K_ESCAPE:
            sys.exit (0)
        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            createExplosion (pygame.mouse.get_pos ())
        if event.type == USEREVENT+1:
            updateAll ()
```

Tutorial

- try out the example code given in the lecture

- make sure you completed last weeks tutorial and adapt this code and
  - without using sprites, implement an explosion class
  - which is activated at the cursor position on the screen

- an explosion can be visually generated by drawing expanding circles
  - and then by reversing the size (remembering to blank out the larger ones)

explosions.py

```
#!/usr/bin/env python3

import pygame, sys, time, random
from pygame.locals import *

ramp_one, ramp_two, ramp_three = None, None, None

wood_light = (166, 124, 54)
wood_dark = (76, 47, 0)
blue = (0, 100, 255)
dark_red = (166, 25, 50)
dark_green = (25, 100, 50)
dark_blue = (25, 50, 150)
black = (0, 0, 0)
white = (255, 255, 255)
```

explosions.py

```
width, height = 1024, 768
screen = None

maxRadius = 60
allExplosions = []
delay = 100    # number of milliseconds delay before generating a USEREVENT

class explosion:
    def __init__ (self, pos):
        self._radius = 1
        self._maxRadius = maxRadius
        self._increasing = True
        self._pos = pos
```

explosions.py

```
def update (self):
    if self._increasing:
        pygame.draw.circle (screen, white, self._pos, self._radius, 0)
        self._radius += 1
        if self._radius == self._maxRadius:
            self._increasing = False
    else:
        pygame.draw.circle (screen, black, self._pos, self._radius, 0)
        self._radius -= 1
        if self._radius > 0:
            pygame.draw.circle (screen, white, self._pos, self._radius, 0)
        else:
            globalRemove (self)
```

explosions.py

```
def createExplosion (pos):
    global allExplosions
    allExplosions += [explosion (pos)]
    pygame.time.set_timer (USEREVENT+1, delay)

def globalRemove (e):
    global allExplosions
    allExplosions.remove (e)
```

explosions.py

```python
def updateAll ():
    if allExplosions != []:
        for e in allExplosions:
            e.update ()
        pygame.display.flip ()
        pygame.time.set_timer (USEREVENT+1, delay)
```

explosions.py

```
def wait_for_event ():
    global screen
    while True:
        event = pygame.event.wait ()
        if event.type == pygame.QUIT:
            sys.exit(0)
        if event.type == KEYDOWN and event.key == K_ESCAPE:
            sys.exit (0)
        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            createExplosion (pygame.mouse.get_pos ())
        if event.type == USEREVENT+1:
            updateAll ()
```

explosions.py

```
def main ():
    global screen
    pygame.init ()
    screen = pygame.display.set_mode ([width, height])
    wait_for_event ()


main ()
```

Homework

- firstly get the `explosions.py` to work

- now comment each function

- comment each class and its use

- familiarise yourself with the game missile command

- see if you can extend this code to place six cities and 3 missile silos statically at the bottom of the screen

Homework 2

- write a Python program to print out all permutations of a string

- for example `ab`

-
  ```
  ab
  ba
  ```

- for example `abc`

-

```
abc
acd
bac
bca
cab
cba
```

Homework 2

- your program should be able to take any size string with unique characters and print out all permutations