

John Romero Programming Proverbs

- 3. “Keep your code absolutely simple. Keep looking at your functions and figure out how you can simplify further.”
- John Romero, “The Early Days of Id Software - John Romero @ WeAreDevelopers Conference 2017”

Python scopes

- a declaration occurs when an object is created
- by default usage is confined to the current scope

Python scopes

myfunc.py

```
#!/usr/bin/python

s = "hello world" # s is declared as a string
i = 42            # i is an int

def func (j):
    i = 1         # another variable, i, local
    print j * 2  # to func

func (2)
print i
```

notice how the scope changes with indentation

```
$ python myfunc.py
4
42
```

Accessing global variables from a function

myfunc2.py

```
#!/usr/bin/python

s = "hello world" # s is declared as a string
i = 42            # i is an int

def func (j):
    global i      # reference global scope i
    i = 1         # assign to global variable
    print j * 2

func (2)
print i
```

■ a common error is to forget the `global` keyword in a function

Python dictionaries

- recall from our first python lecture that python has a dictionary type

```
#!/usr/bin/python  
  
d = {'spam':2, 'ham':1, 'eggs':10}  
  
print d['eggs']
```

```
$ python py13.py  
10
```

Python dictionaries

- dictionaries can be altered
 - by adding new entries
 - by changing values
 - by removing entries

- we can test for the presence of an entry within the dictionary

Python dictionaries

```
#!/usr/bin/python

d = {'spam':2, 'ham':1, 'eggs':10}
print d['eggs']

d['eggs']=12
d['bacon']=1
print d['bacon']
print d
```

```
$ python py14.py
10
1
{'spam': 2, 'ham': 1, 'bacon': 1, 'eggs': 12}
```

Dictionary initialisation

- we can create an empty dictionary by:

```
d = {}
```

- we can add to a dictionary by:

```
d['foobar'] = 4
```


Dictionary modification

- we can remove an entry via:

```
del d['eggs']
```

Dictionary modification

- we can test whether an item exists in the dictionary

```
#!/usr/bin/python

d = {'spam':2, 'ham':1, 'eggs':10}

if d.has_key('eggs'):
    print 'we have some eggs'
else:
    print 'we do not have any eggs'

if d.has_key('flour'):
    print 'we have some flour'
else:
    print 'we do not have any flour'
```

PyGame

- is a set of Python modules which sits on top of the excellent SDL library
- allows you to create fully featured games and multimedia programs in the python language
- PyGame is highly portable and runs on nearly every platform and operating system

PyGame

- it has the ability to initialise screen resolution or window size
- sprites, sound, collisions, keyboard, mouse, joystick events
- images, text can all be rendered to the 2D screen (or surface)
- fast image copies are achieved using the SDL (Simple DirectMedia Layer)

Is Python suitable for gaming?

- 30 frames per second is often quoted as the minimum necessary for smooth graphics
 - this means you must compute the next frame in under $1/30$ of a second ≈ 30 milliseconds

- for 2D games PyGame is a very good solution
 - rapid prototyping
 - you can convert any slow pieces of Python into C if necessary and call these from Python
 - classically 90% of the time is spent in 10% of the code

Is Python suitable for gaming?

- **abridged** (<http://www.pygame.org/docs/tut/intro/intro.html>)

- games are often split into two components
 - game engine
 - game logic

- most of the time the game engine must run really fast whereas the game logic requires less processing power
 - game engine - written in C exploiting the SDL and presented as PyGame modules
 - game logic - your python code

Is Python suitable for gaming?

- scripting is used in games such as Quake3 and Unreal (although not Python..)
- in 2001 Rebel Act Studios released "Severance: Blade of Darkness"
- this used their own custom 3D engine
 - the rest of the game is written with Python
- game was a bloody action 3rd person perspective fighter
 - third party addons for this game are available and they are Python source files

Bouncing ball

```
#!/usr/bin/python

import sys, pygame

width = 320
height= 240
speed = [2, 2]
black = (0, 0, 0)

pygame.init ()
screen = pygame.display.set_mode([width, height])
ball = pygame.image.load("ball.png").convert ()
ballrect = ball.get_rect()
```


Bouncing ball

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

    ballrect = ballrect.move (speed)
    if ballrect.left < 0 or ballrect.right > width:
        speed[0] = -speed[0]
    if ballrect.top < 0 or ballrect.bottom > height:
        speed[1] = -speed[1]

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

- the ball can be taken from here: [ball.png](#) <ball.png>

Commentary

- **Reference** (<http://www.pygame.org/docs/tut/intro/intro.html>)
- `import pygame` imports the package with all the available PyGame modules
`pygame.init()` initialises each of these modules
- `pygame.display.set_mode()` creates the graphic window

Commentary

- PyGame represents images as Surface objects
 - `display.set_mode()` creates a new Surface object that represents the actual displayed graphics
 - any modification to this Surface becomes visible on the monitor
- `pygame.image.load().convert()` loads in the ball image
 - we convert it to a fast internal format

Commentary

- the program is initialised and ready to run so we enter an infinite loop in which we check for user input, move the ball, and then draw the ball
- `ballrect = ballrect.move(speed)` updates the ball position
- the two `if` statements check to see whether the ball has touched the edge of the window
 - the `then` section bounces the ball back
- `screen.fill(black)` erases everything on the screen
- `screen.blit(ball,ballrect)` redraw the ball at the new position

Commentary

- `pygame.display.flip()` now make the screen visible

Detail

- notice that PyGame uses double buffering
 - we can update the `screen` piece at a time and the user sees nothing
 - only when we call `pygame.display.flip()` does the screen become visible
 - ensures speed and near instant updates
- it also allows us to use high level techniques such as `screen.fill(black)` without much of a performance penalty

Tutorial

- try out the bouncing ball example and modify the example so that the ball travels at a slower and faster speed
- try adding a second ball..
- now make the ball travel slower at the top of the screen and faster at the bottom (adding gravity)
- read [the 13 rules on PyGame usage](http://www.pygame.org/docs/tut/newbieguide.html) (`http://www.pygame.org/docs/tut/newbieguide.html`)

PyGame documentation

- PyGame is well documented online, (<http://www.pygame.org/docs>)