PsychoPy
Psychology software in Python

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Introduction to Psychopy

孙彦良
心理学院2-514
yanliangsun@126.com

Chapter1 About Psychopy

Overview

PsychoPy is an open source software package, written in Python, for the generation of experiments for neuroscience and experimental psychology

Features

- simple install process
- precise timing
- huge variety of stimuli generated in real-time
- platform independent-run the same script on win, os or linux
- coder / builder
- input from keyboard, mouse, microphone or button boxes

Install Psychopy

for windows

- http://www.psychopy.org/index.html

  StandalonePsychoPy-1.83.04-win32.exe

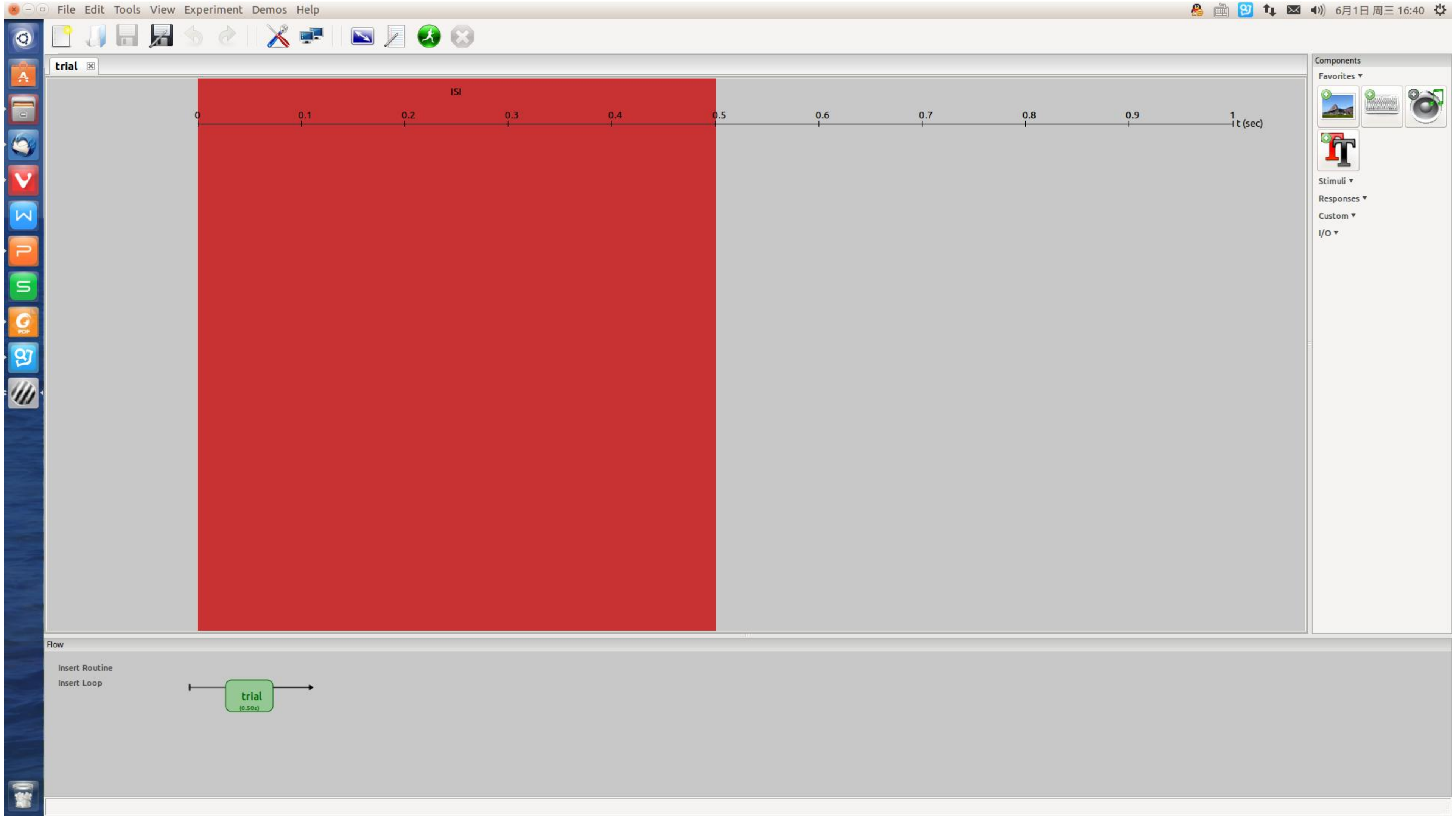
for mac os x

- sudo port install py25-psychopy

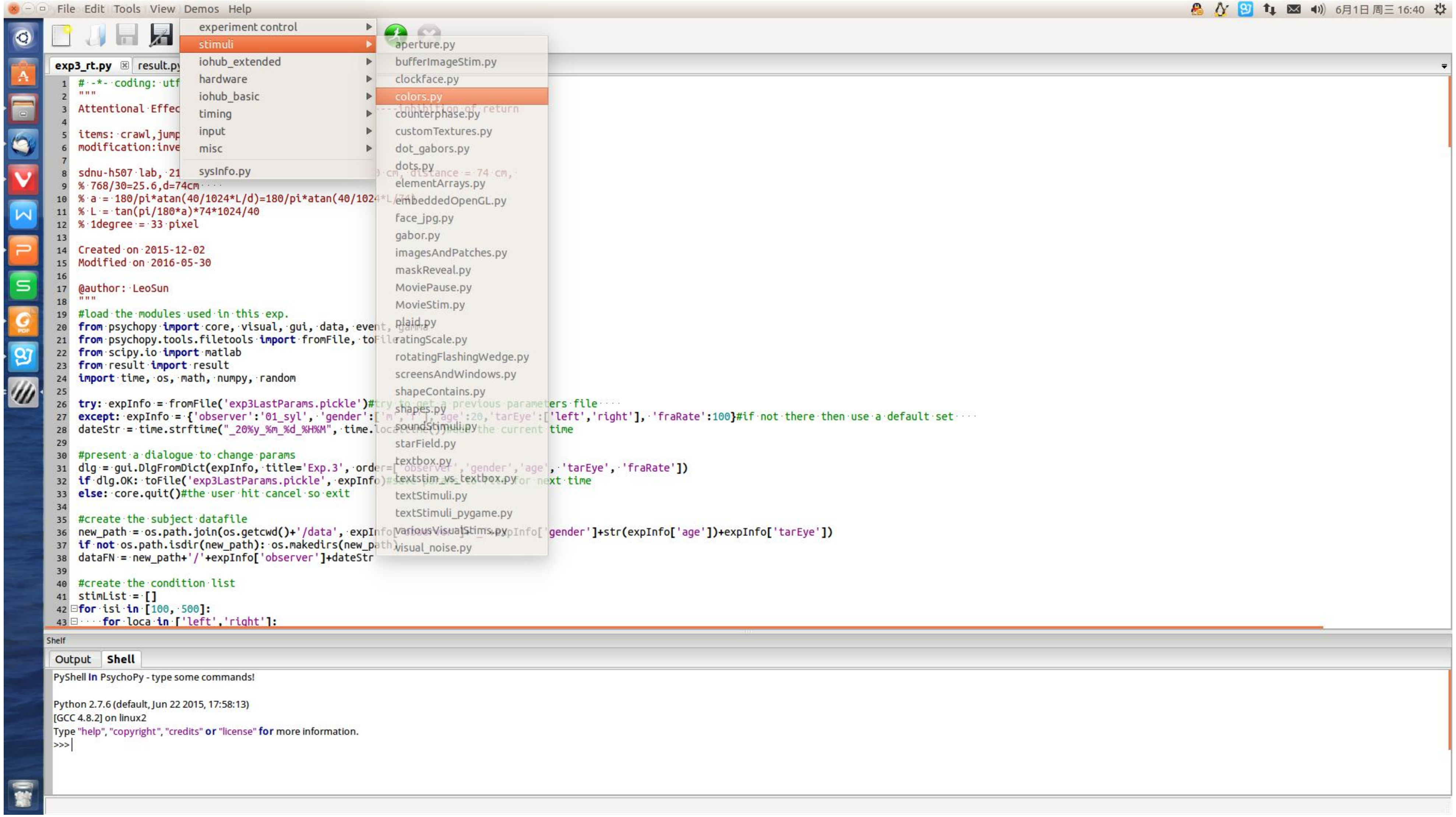
for linux-ubuntu

- sudo apt-get install psychopy

The First Glimpse-Builder



The First Glimpse-Coder



```
# -*- coding: utf
"""
Attentional Effec

items: crawl,jump
modification:inve

sdnu-h507 lab, 21
% 768/30=25.6,d=74cm
% a = 180/pi*atan(40/1024*L/d)=180/pi*atan(40/1024*L/(74
% L = tan(pi/180*a)*74*1024/40
% 1degree = 33 pixel

Created on 2015-12-02
Modified on 2016-05-30

@author: LeoSun
"""
#load the modules used in this exp.
from psychopy import core, visual, gui, data, event, gamma
from psychopy.tools.filetools import fromFile, toFile
from scipy.io import matlab
from result import result
import time, os, math, numpy, random

try: expInfo = fromFile('exp3LastParams.pickle')#try to get a previous parameters file
except: expInfo = {'observer':'01_syl', 'gender':['m','f'],'age':20,'tarEye':['left','right'], 'fraRate':100}#if not there then use a default set
dateStr = time.strftime("_20%y_%m_%d_%H%M", time.localtime())#add the current time

#present a dialogue to change params
dlg = gui.DlgFromDict(expInfo, title='Exp.3', order=['observer','gender','age', 'tarEye', 'fraRate'])
if dlg.OK: toFile('exp3LastParams.pickle', expInfo)#save params to file for next time
else: core.quit()#the user hit cancel so exit

#create the subject datafile
new_path = os.path.join(os.getcwd()+'/data', expInfo['observer']+'_'+expInfo['gender']+str(expInfo['age'])+expInfo['tarEye'])
if not os.path.isdir(new_path): os.makedirs(new_path)
dataFN = new_path+'/'+expInfo['observer']+dateStr

#create the condition list
stimList = []
for isi in [100, 500]:
    for loca in ['left','right']:
```

```
PyShell In PsychoPy - type some commands!

Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Chapter2 Fundamentals of Python

Python Interactive Shell

```
PyShell in PsychoPy - type some commands!
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print 'Hello World!'
Hello World!
>>> print 5 + 3
8
>>> print 'cat' + 'dog'
catdog
```

```
>>> print 'Hello ' * 8
Hello Hello Hello Hello Hello Hello Hello Hello
```

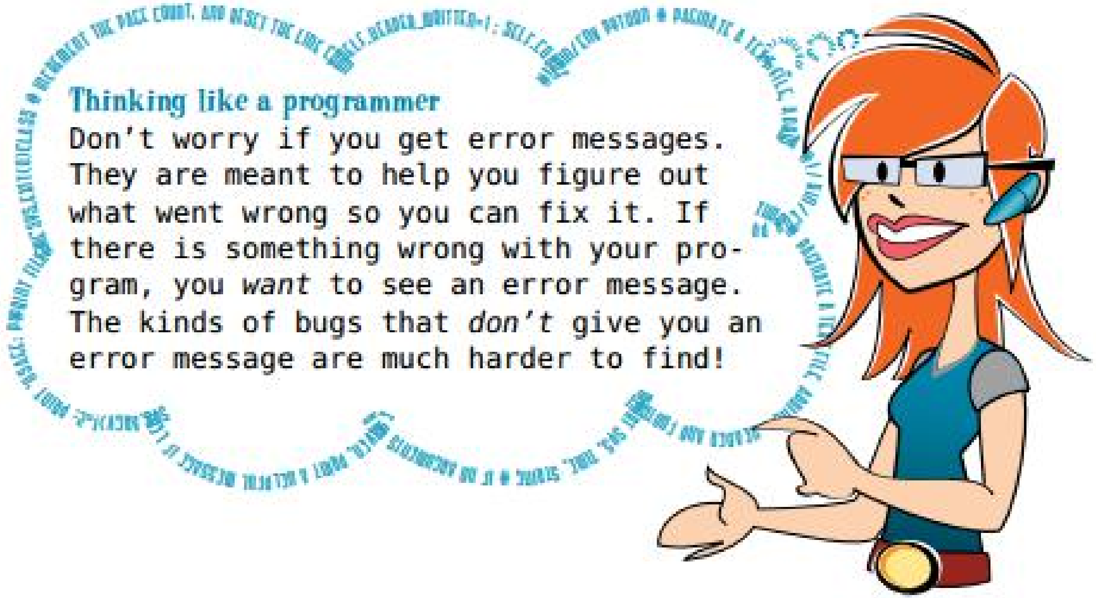
Learning from Error Messages

Syntax errors

```
>>>print Bye for now!'
File "<input>", line 1
    print Bye for now!'
          ^
SyntaxError: invalid syntax
```

Runtime errors

```
>>>print "Bye for now!" + 5
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```



Thinking like a programmer

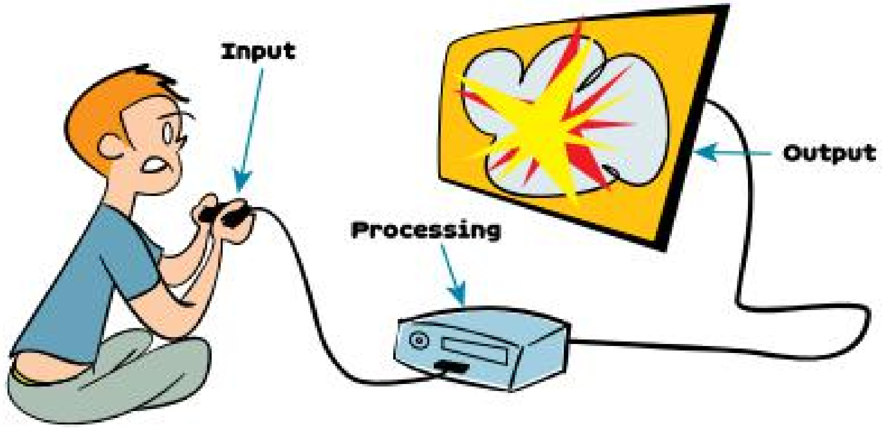
Don't worry if you get error messages. They are meant to help you figure out what went wrong so you can fix it. If there is something wrong with your program, you *want* to see an error message. The kinds of bugs that *don't* give you an error message are much harder to find!

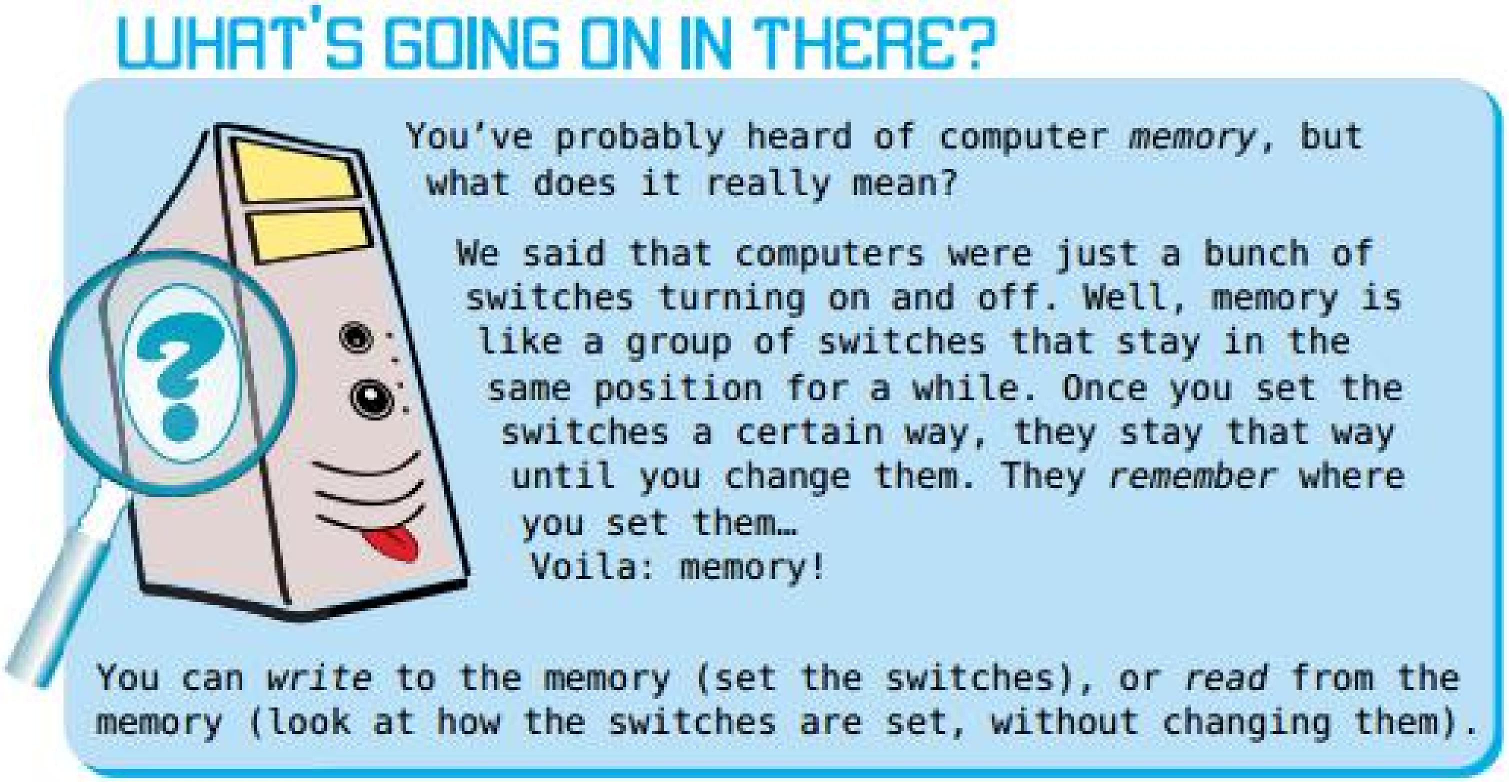
Learning from help()

```
>>>help('dir')
```

```
Help on built-in function dir in module __builtin__:

dir(...)
    dir([object]) -> list of strings

    If called without an argument, return the names in the current scope.
    Else, return an alphabetized list of names comprising (some of) the attributes
    of the given object, and of attributes reachable from it.
    If the object supplies a method named __dir__, it will be used; otherwise
    the default dir() logic is used and returns:
      for a module object: the module's attributes.
      for a class object:  its attributes, and recursively the attributes
        of its bases.
      for any other object: its attributes, its class's attributes, and
        recursively the attributes of its class's base classes.
```

Memory and Variables





WHAT'S GOING ON IN THERE?

You've probably heard of computer *memory*, but what does it really mean?

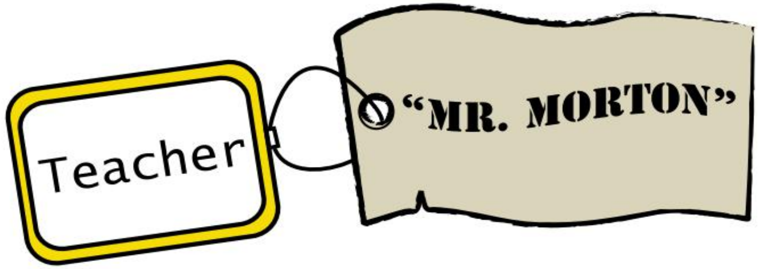
We said that computers were just a bunch of switches turning on and off. Well, memory is like a group of switches that stay in the same position for a while. Once you set the switches a certain way, they stay that way until you change them. They *remember* where you set them…
Voila: memory!

You can *write* to the memory (set the switches), or *read* from the memory (look at how the switches are set, without changing them).

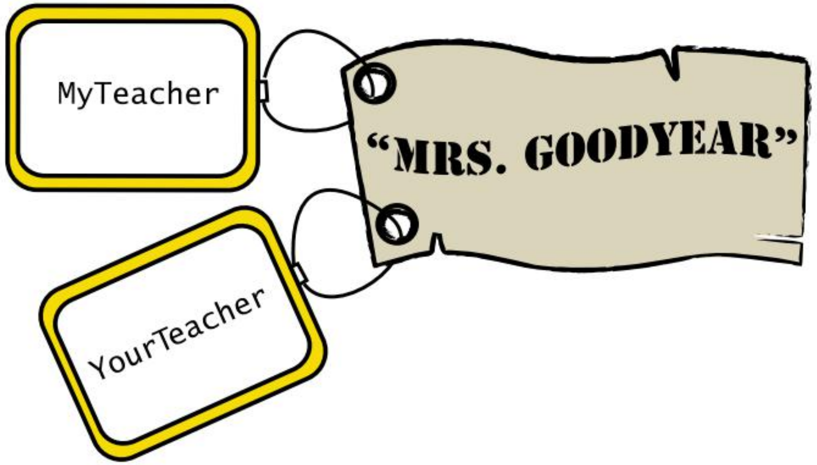
How do we tell Python where in the memory to put something?
How do we find it again?

Names / Variables

```
>>> Teacher = 'Mr. Morton'
>>> print Teacher
Mr. Morton
```

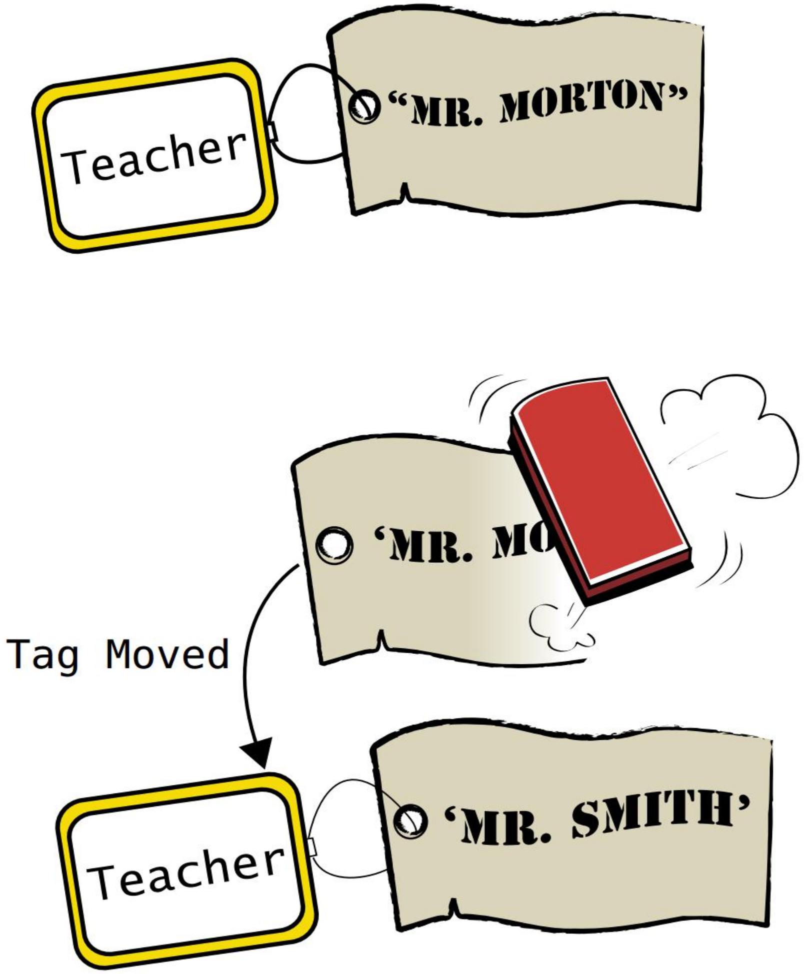


```
>>> MyTeacher = 'Mrs. Goodyear'
>>> YourTeacher = MyTeacher
>>> MyTeacher
'Mrs. Goodyear'
>>> YourTeacher
'Mrs. Goodyear'
```



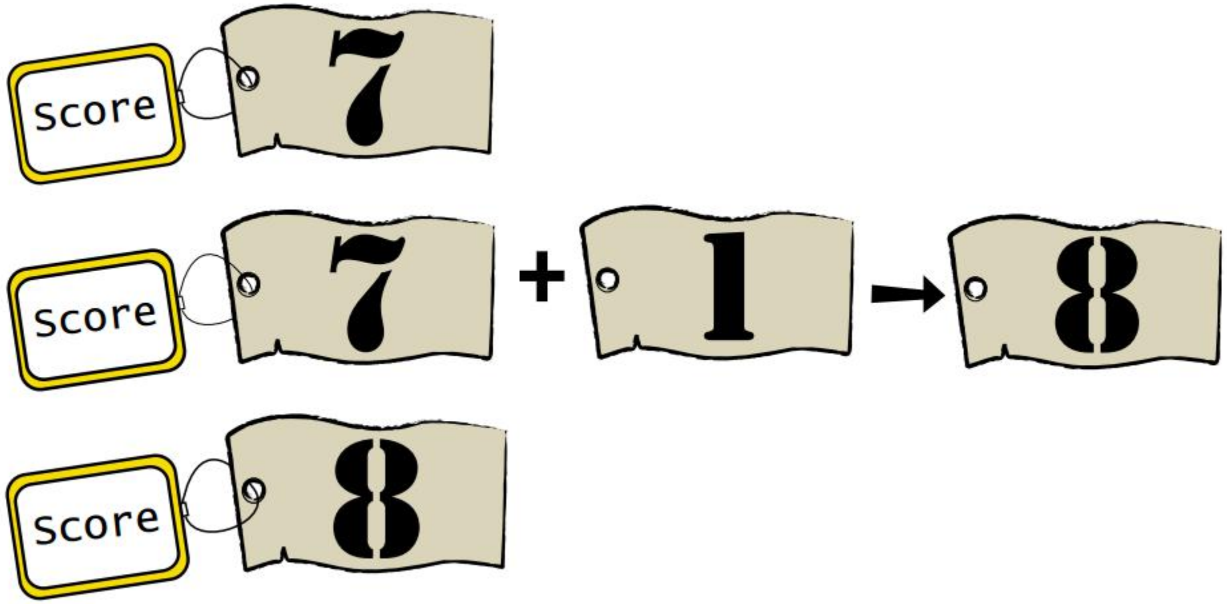
Names / Variables

```
>>> Teacher = 'Mr. Morton'
>>> Teacher
'Mr. Morton'

>>> Teacher = 'Mr. Smith'
>>> Teacher
'Mr. Smith'
```



The New Me

```
>>> Score = 7
>>> Score = Score + 1
>>> Score
8

>>> Score += 1
>>> Score
9
```



Variable Naming Rules

They must begin with either a letter or an underscore character (_)

They are case-sensitive, which means that uppercase and lowercase matter

A variable name can't start with a number

A variable name can't have any spaces in it

~ ` ! @ # $ % ^ & * ( ) ; - : " ' < > , . ? / { } [ ] + = /

my_answer	my-answer
answer23	23answer
YourAnswer	Your Answer

Numbers and Strings

```
>>> teacher = 'Mr. Morton'
>>> teacher = "Mr. Morton"
>>>long_string = '''Sing a song of sixpence, a pocket full of rye,
Four and twenty blackbirds baked in a pie.
When the pie was opened the birds began to sing.
Wasn't that a dainty dish to set before the king?'''
```

```
>>> first = 5
>>> second = 3
>>> first + second
8
```

```
>>> first = '5'
>>> second = '3'
>>> first + second
'53'
```

Basic Math

```
>>>8 + 5
13
>>>8 - 5
3
>>>8 * 5
40
>>>8 / 5
1
>>>8.0 / 5
1.6
```

```
>>> 8 % 5
3
>>> 8 ** 5
32768

1.752e-13
```

Changing Types of Data

float()

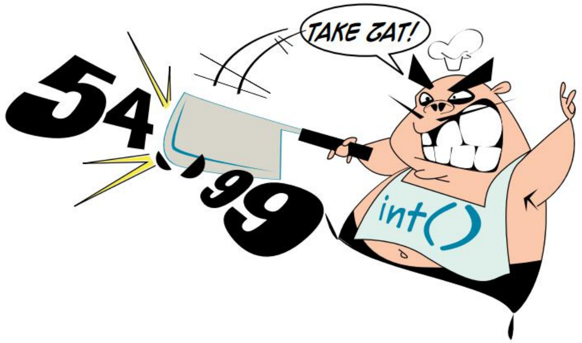
```
>>> a = 24
>>> b = float(a)
>>> b
24.0
```

```
>>> a = '76.3'
>>> b = float(a)
>>> b
76.3
```

int()

```
>>> c = 38.0
>>> d = int(c)
>>> d
38
```

```
>>> e = 54.99
>>> f = int(e)
>>> f
54
```



str()

```
>>> a = 38.0
>>> b = str(c)
>>> b
'38.0'
```

type()

```
>>> type(b)
<type 'str'>
```

Print Formatting and Strings-New lines

```
print 'Hi'
print 'There'

print 'Hi',
print 'There'

print 'Hi' + 'There'
print 'Hi ' + 'There'
```

```
print 'Hi'
print
print 'There'

print 'Hello \nWorld'
```

Print Formatting and Strings-tabs

```
>>>print 'ABC\tXYZ'
>>>print 'ABCDE\tXYZ'
>>>print 'ABCDEF\tXYZ'
>>>print 'ABCDEFG\tXYZ'
>>>print 'ABCDEFGHI\tXYZ'

>>>print 'hi\\there'
```

Inserting variables in strings

```
name = 'Warren Sande'
print 'My name is', name, 'and I wrote this book.'

name = 'Warren Sande'
print 'My name is %s and I wrote this book' % name

age = 13
print 'I am %i years old.' % age

average = 75.6
print 'The average on our math test was %f%%.' % average
```

Number formatting-%

```
>>>dec_number = 12.3456
>>>print 'It is %.2f degrees today.' % dec_number

>>>number = 12.67
>>>print '%i' % number
>>>print '%+f' % number
>>>number = 12.3456789
>>>print '%.3e' % number
>>>print '%g' % number
```

String Formatting-format()

```
math = 75.4
science = 82.1
print 'I got %.1f in math, %.1f in science' % (math, science)
print 'I got {0:.1f} in math, {1:.1f} in science'.format(math, science)
```

Splitting & Joining strings

```
>>>name_string = 'Sam,Brad,Alex,Cameron,Toby,Gwen,Jenn'
>>>names = name_string.split(',')
>>>parts = name_string.split('Cameron,')

>>> word_list = ['My', 'name', 'is', 'Warren']
>>> long_string = ' '.join(word_list)
```

Searching for strings

```
>>>name = 'Frankenstein'
>>>name.startswith('Frank')
>>>name.endswith('stein')

>>>addr1 = '657 Maple Lane'
>>> if 'Maple' in addr1:
            position = addr1.index('Maple')
            print "found 'Maple' at index", position
```

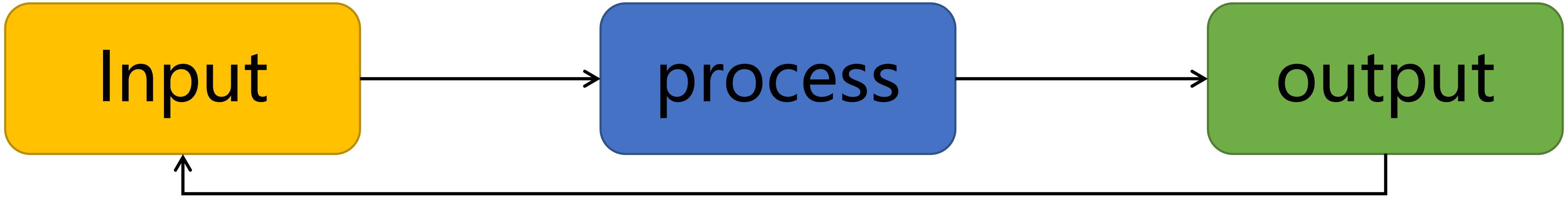
Removing part of a string

```
>>>name = 'Warren Sande'
>>>short_name = name.strip('de')

>>>name = 'Warren Sande        '
>>>short_name = name.strip()
```

Changing case

```
>>> string1 = 'HI, LEO'
>>> string2 = string1.lower()

>>> string1 = 'hi, leo'
>>> string2 = string1.upper()
```

Input



Input

process

output

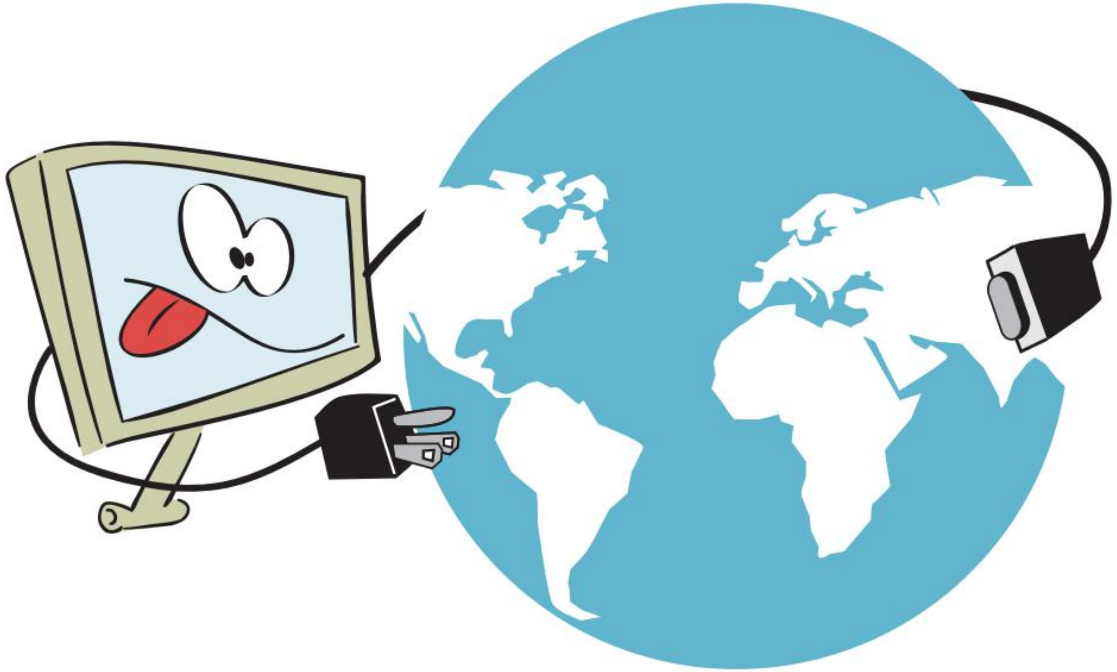
raw_input()

raw_input()

```
>>>someName = raw_input ('Enter your name: ')
Enter your name: leo
>>>someName
'leo'

>>>fahrenheit = float(raw_input('temperature in Fahrenheit: '))
temperature in Fahrenheit: 90
>>>fahrenheit
90.0
```

Input from the Web

```
>>>import urllib2
>>>file = urllib2.urlopen('http://helloworldbook2.com/data/message.txt')
>>>message = file.read()
>>>print message
```



GUIs—Graphical User Interfaces

EasyGui is a Python module that makes it very easy to make simple GUIs

install EasyGui

- download: https://sourceforge.net/projects/easygui
- for win

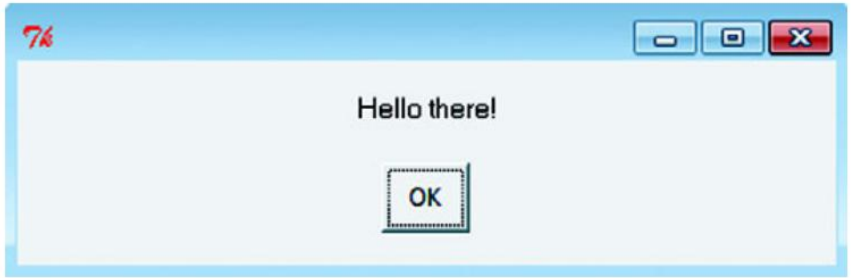
```
Run cmd as administrator
cd xx\2016 First Updates\robertlugg-easygui-cbd30b0
C: \"Program Files"\Psychopy2\python.exe setup.py install
```

- for linux

```
open terminal: ctrl+alt+t
cd xx\2016 First Updates\robertlugg-easygui-cbd30b0
sudo python setup.py install
```

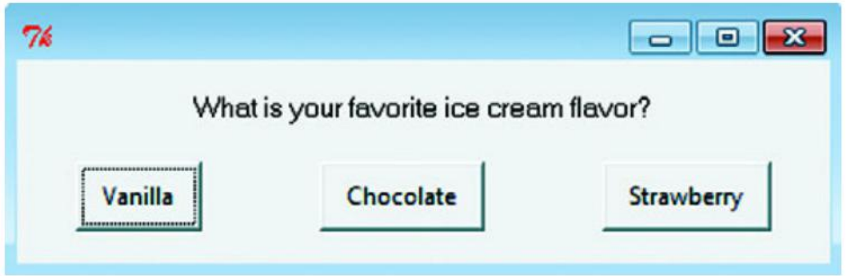
EasyGui

```
>>> import easygui
>>> easygui.msgbox('Hello there!')
'OK'
```

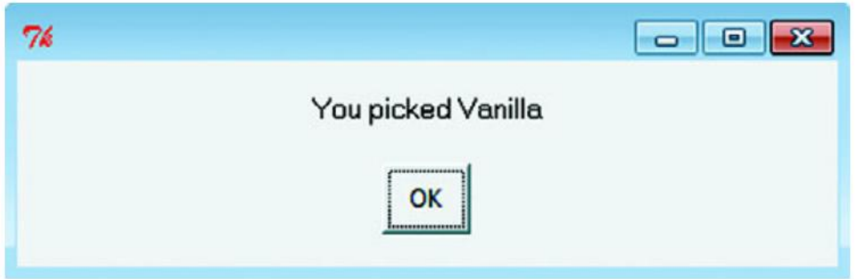


Hello there!

```
>>>import easygui
>>>flavor = easygui.buttonbox('What is your favorite ice cream flavor?',
choices = ['Vanilla', 'Chocolate', 'Strawberry'] )
>>>easygui.msgbox ('You picked ' + flavor)
```



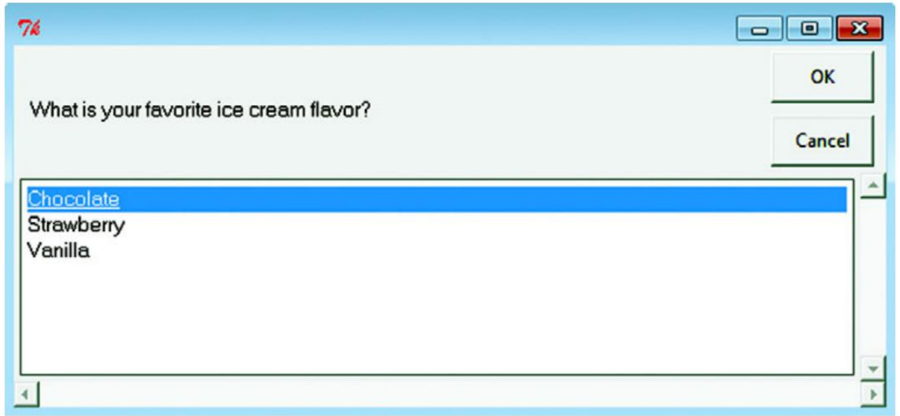
What is your favorite ice cream flavor?



You picked Vanilla

EasyGui

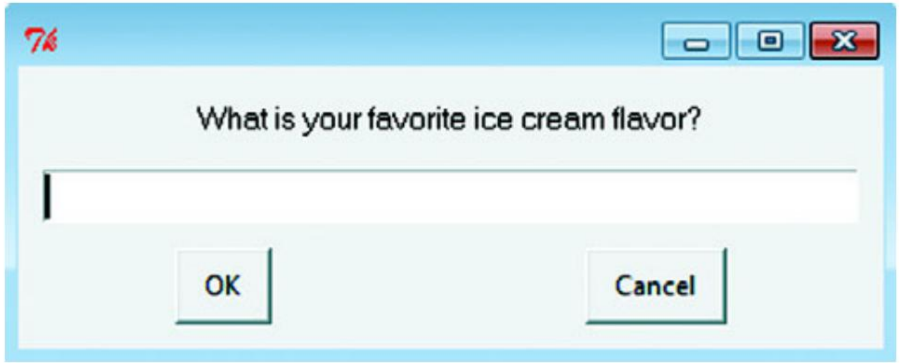
```
>>>import easygui
>>>flavor = easygui.choicebox('What is your favorite ice cream flavor?',
choices = ['Vanilla', 'Chocolate', 'Strawberry'] )
easygui.msgbox ('You picked ' + flavor)
```



What is your favorite ice cream flavor?

Chocolate
Strawberry
Vanilla

```
>>>import easygui
>>>flavor = easygui.enterbox('What is your favorite ice cream flavor?')
>>>easygui.msgbox ('You entered ' + flavor)
```

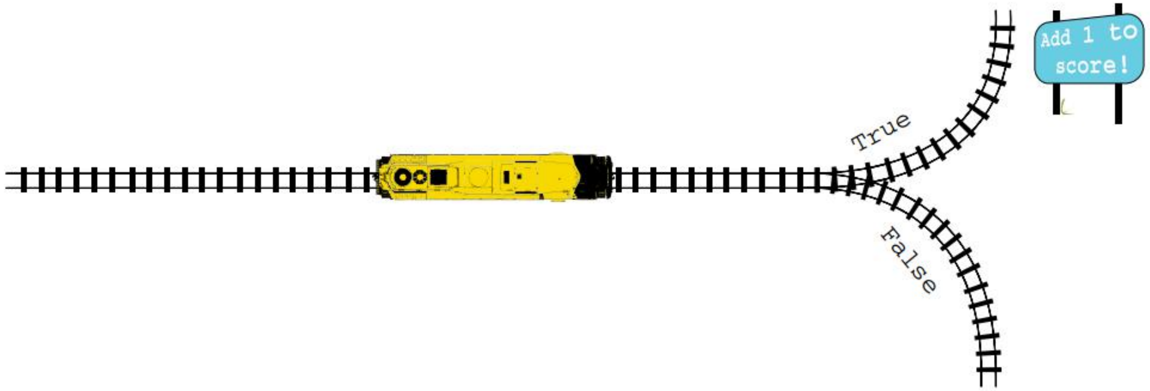


What is your favorite ice cream flavor?

Decisions, Decisions

Python uses the keyword if to test conditions, like this:

```
if timsAnswer == correctAnswer:
    print 'You got it right!'
    score = score + 1
print 'Thanks for playing.'
```

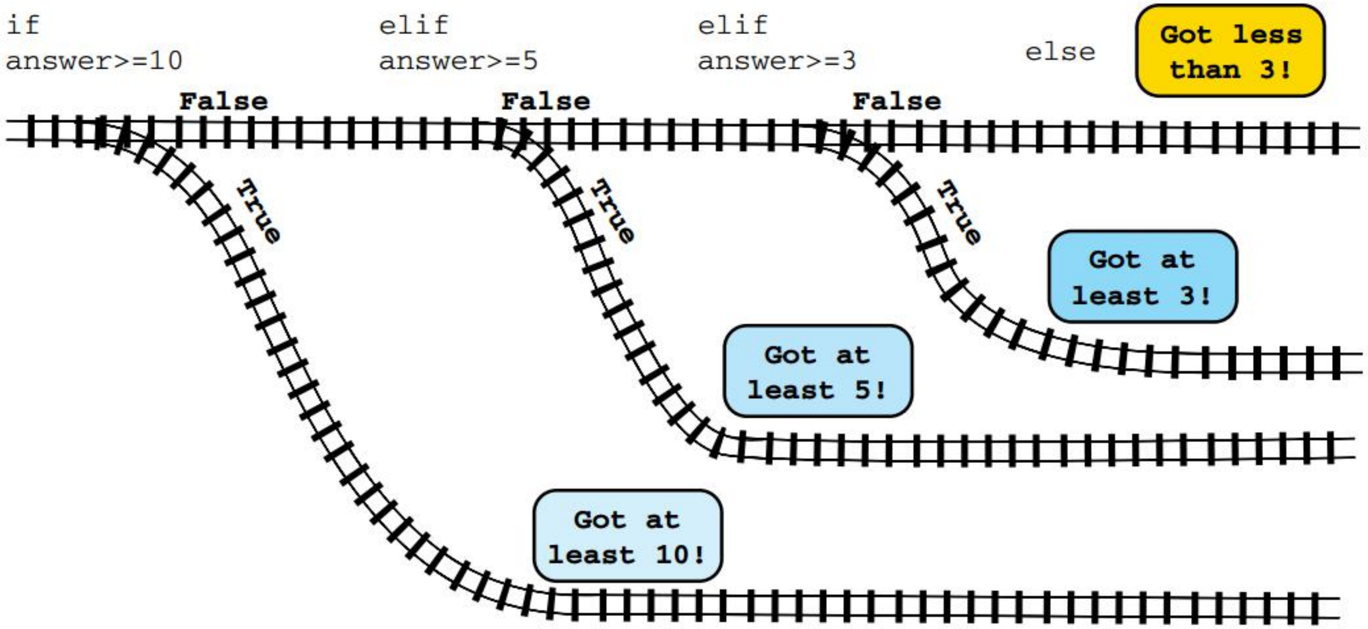


Indenting(4 space/1 tab)

block

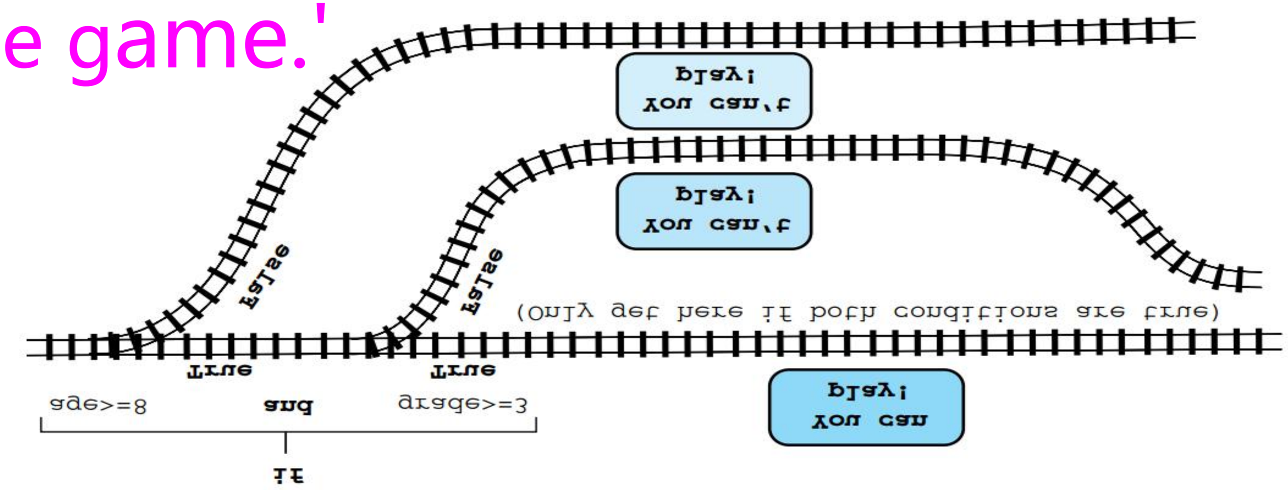
if...elif...else

```
if answer >= 10:
    print 'You got at least 10!'
elif answer >= 5:
    print 'You got at least 5!'
elif answer >= 3:
    print 'You got at least 3!'
else:
    print 'You got less than 3.'
```



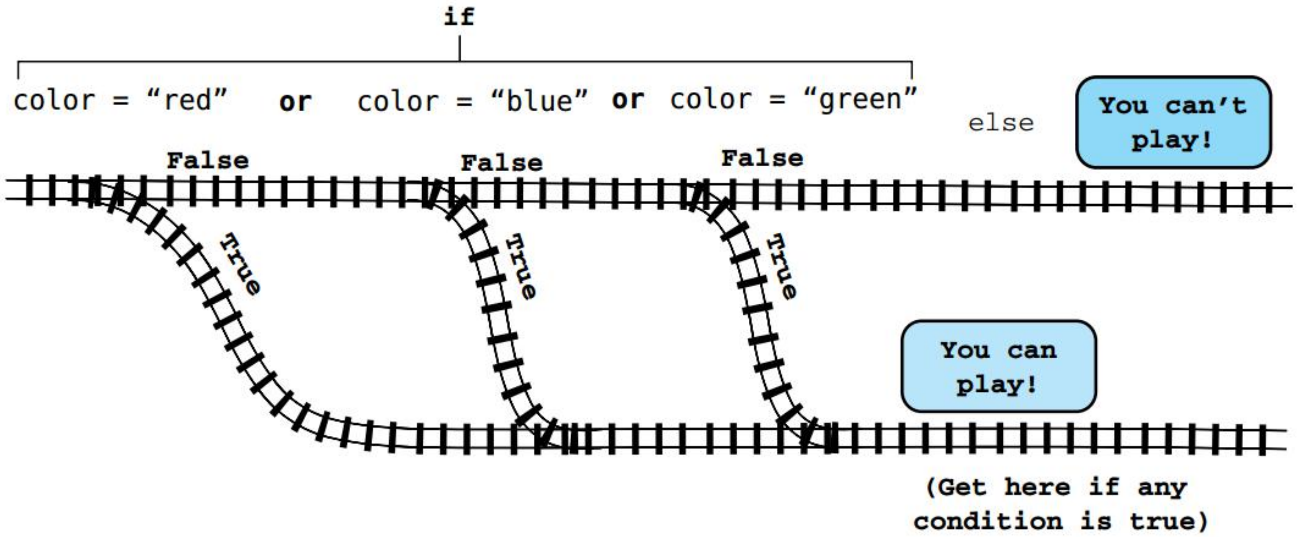
Using and

```
age = float(raw_input('Enter your age: '))
grade = int(raw_input('Enter your grade: '))
if age >= 8 and grade >= 3:
    print 'You can play this game.'
else:
    print 'Sorry, you can't play the game.'
```



Using or

```
color = raw_input("Enter your favorite color: ")
if color == "red" or color == "blue" or color == "green":
    print "You are allowed to play this game."
else:
    print "Sorry, you can't play the game."
```



Using not & List of operators

```
if not (age < 8):
if age >= 8:
```

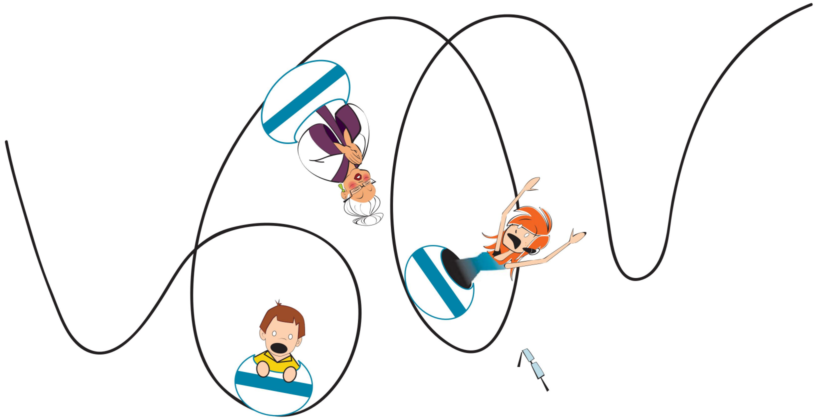
Math Operator		Comparison Operators	
=	assignment	==	equality
+	additon	<	less than
-	subtraction	>	greater than
+=	increment	<=	less than or equal to
-=	decrement	>=	greater than or equal to
*	multiplication	!=	not equal to
/	division	<>	not equal to
%	modulus		
**	exponentiation		

Loop the Loop

Computer programs often repeat the same steps over and over again--looping

There are two main kinds of loops:

- Those that repeat a certain number of times—These are called counting loops
- Those that repeat until a certain thing happens—These are called conditional loops



Counting Loops

```
for looper in [1, 2, 3, 4, 5]:
        print looper, "times 8 =", looper * 8
```

```
>>>
1 times 8 = 8
2 times 8 = 16
3 times 8 = 24
4 times 8 = 32
5 times 8 = 40
```

```
for looper in range(5):
        print looper, "times 8 =", looper * 8
```

```
for i in range(5):
        print i, "times 8 =", i * 8
```

Counting without numbers

```
for cool_guy in ["Spongebob", "Spiderman", "Justin Timberlake", "My Dad"]:
    print cool_guy, "is the coolest guy ever!  “
```

?

Conditional Loop

```
print "Type 3 to continue, anything else to quit."
someInput = raw_input()
while someInput == '3':
        print "Thank you for the 3. Very kind of you."
        print "Type 3 to continue, anything else to quit."
        someInput = raw_input()
print "That's not 3, so I'm quitting now."
```

Bailing out of a loop—break and continue

Jumping ahead—continue

```
for i in range(1, 6):
    print
    print 'i =', i,
    print 'Hello, how',
    if i == 3:
        continue
    print 'are you today?'
```

```
>>>
i = 1 Hello how are you today?
i = 2 Hello how are you today?
i = 3 Hello how
i = 4 Hello how are you today?
i = 5 Hello how are you today?
```

Bailing out of a loop—break and continue

Bailing out—break

```
for i in range(1, 6):
    print
    print 'i =', i,
    print 'Hello, how',
    if i == 3:
        break
    print 'are you today?'
```

```
>>>
i = 1 Hello how are you today?
i = 2 Hello how are you today?
i = 3 Hello how
```

Comments

```
"""   """
```

```
""" Here is a comment that is on multiple
lines, using a triple-quoted string.
It's not really a comment, but it
behaves like one.
"""
```

```
#
```

```
# ***************
# This is a program to illustrate how comments are used in Python
# The row of stars is used to visually separate the comments
# from the rest of the code
# ***************
```

Lists

```
>>>family = ['Mom', 'Dad', 'Junior', 'Baby']
>>>friends = [ ]
>>>friends.append('David')
>>>print friends
['David']
>>>my_list = [5, 10, 23.76, 'Hello', myTeacher, 7, another_list]
>>>letters = ['a', 'b', 'c', 'd', 'e']
>>>print letters[0]
a
```

"Slicing" a list

```
>>>print letters[1:4]
['b', 'c', 'd']
>>>print letters[1]
b
>>>print letters[1:2]
['b']
>>>print letters[:2]
['a', 'b']
>>> print letters[2:]
['c', 'd', 'e']
```

```
>>>print letters[ : ]
['a', 'b', 'c', 'd', 'e']
>>>letters[2] = 'z'
>>>print letters
['a', 'b', 'z', 'd', 'e']
```

Other ways of adding to a list

append( ) adds one item to the end of the list

extend( ) adds multiple items to the end of the list

insert( ) adds one item somewhere in the list

```
>>> letters.append('n')
>>> print letters
['a', 'b', 'c', 'd', 'e', 'n']
>>> letters.extend(['p', 'q', 'r'])
>>> print letters
['a', 'b', 'c', 'd', 'e', 'n', 'g', 'p', 'q', 'r']
```

```
>>> letters.insert(2, 'z')
>>> print letters
['a', 'b', 'z', 'c', 'd', 'e', 'n', 'g', 'p', 'q', 'r']
```

Deleting from a list

```
>>>letters = ['a', 'b', 'c', 'd', 'e']
>>> letters.remove('c')
>>> print letters
['a', 'b', 'd', 'e']
>>> del letters[3]
>>>print letters
['a', 'b', 'd']
>>> letters.pop()
>>> print letters
['a', 'b']
```

```
>>>letters = ['a', 'b', 'c', 'd', 'e']
>>> letters.pop(2)
>>> print letters
['a', 'b', 'd', 'e']
```

Searching a list

- Find out whether an item is in a list or not
- Find out where an item is in the list (its index)

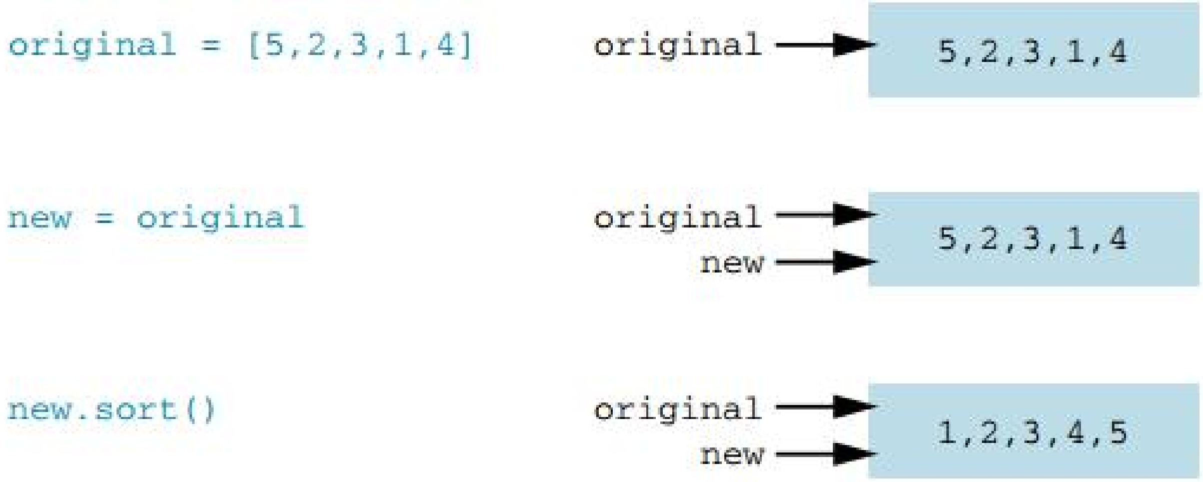
```
>>>letters = ['a', 'b', 'c', 'd', 'e']
>>> 'a' in letters
True
```

```
if 'a' in letters:
    letters.remove('a')
```

```
>>> print letters.index('d')
3
```

Sorting lists

```
>>>letters = ['d', 'a', 'e', 'c', 'b']
>>>letters.sort()
>>>print letters
['a', 'b', 'c', 'd', 'e']
>>>letters = ['d', 'a', 'e', 'c', 'b']
>>>letters.reverse()
>>>print letters
['e', 'd', 'c', 'b', 'a']
```



```
original = [5,2,3,1,4]
new = original
new.sort()
```

```
>>> original = [5, 2, 3, 1, 4]
>>> newer = sorted(original)
>>> print original
[5, 2, 3, 1, 4]
>>> print newer
[1, 2, 3, 4, 5]
```

Mutable and immutable

immutable

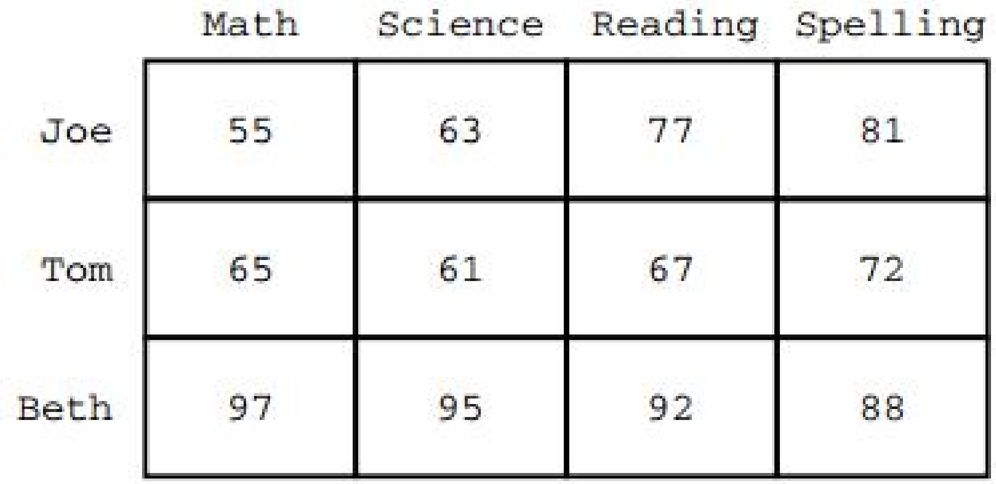
mutable

number, string, list?

tuple~an immutable list

```
>>>my_tuple = ("red", "green", "blue")
>>>background = (128, 128, 128)
```

Lists of lists: tables of data



	Math	Science	Reading	Spelling
Joe	55	63	77	81
Tom	65	61	67	72
Beth	97	95	92	88

```
>>> joeMarks = [55, 63, 77, 81]
>>> tomMarks = [65, 61, 67, 72]
>>> bethMarks = [97, 95, 92, 88]
>>> classMarks = [joeMarks, tomMarks, bethMarks]
>>> print classMarks
[[55, 63, 77, 81], [65, 61, 67, 72], [97, 95, 92, 88]]
```

```
>>> for studentMarks in classMarks:
        print studentMarks
[55, 63, 77, 81]
[65, 61, 67, 72]
[97, 95, 92, 88]
>>> print classMarks[0]
[55, 63, 77, 81]
>>> print classMarks[0][2]
77
```

Dictionaries

A Python dictionary is a way of associating two things to each other. These two things are called the key and the value

```
>>>phoneNumbers = {}
>>>phoneNumbers["John"] = "555-1234"
>>>print phoneNumbers
{'John': '555-1234'}
>>>phoneNumbers = {"John": "555-1234"}
>>>phoneNumbers["Mary"] = "555-6789"
>>>phoneNumbers["Bob"] = "444-4321"
>>>phoneNumbers["Jenny"] = "867-5309"
```

Dictionaries~operation

```
>>>phoneNumbers.keys()
['Bob', 'John', 'Mary', 'Jenny']
>>>phoneNumbers.values()
['444-4321', '555-1234', '555-6789', '867-5309']
>>>del phoneNumbers["John"]
>>>phoneNumbers.clear()
>>>"Bob" in phoneNumbers
True
```

How to break programs into smaller parts?

Functions

- like building blocks of code that you can use over and over again

Objects

- a way of describing pieces of your program as self-contained units

Modules

- separate files that contain parts of your program

Functions

```
def calculateTax(price, tax_rate):
        Global price
        total = price + (price * tax_rate)
        print my_price
        return total
my_price = float(raw_input ("Enter a price: "))
totalPrice = calculateTax(my_price, 0.06)
print "price = ", my_price, " Total price = ", totalPrice
```

Local variables v.s. Global variables

Objects

Lists are a way to collect variables (data) together

Functions are a way to collect some code together into a unit that you can use over and over again

Objects are a way to collect functions and data together

Objects

Real objects in the real world have

- Things that you can do to them (actions)
- Things that describe them (attributes or properties)

Objects in Python

- Object = attributes + methods
- object.attribute

  ball.color = 'green'
- object.method()

  ball.kick()
- dot notation

Creating objects

two steps to creating an object

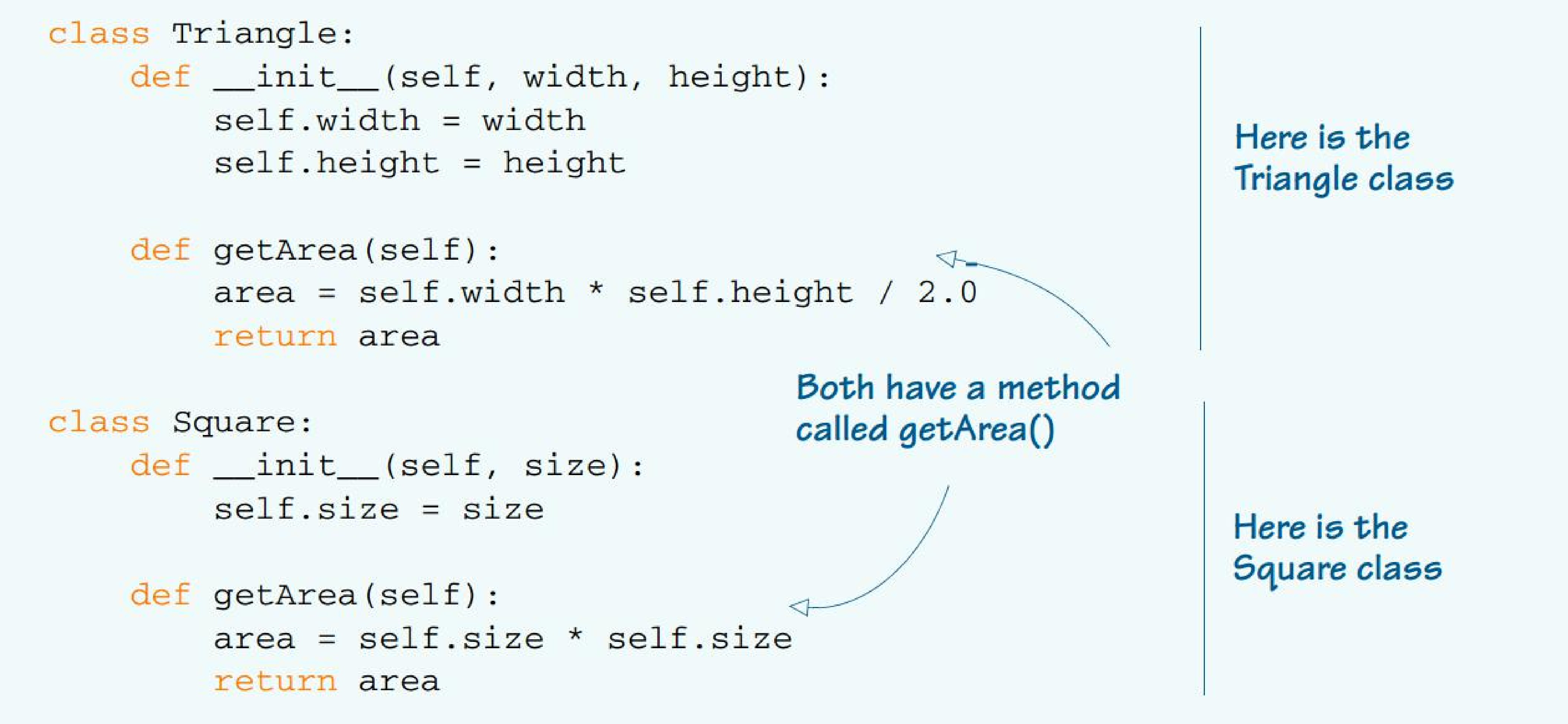
- define what the object will look like and act like -- class
- use the class to make an actual object -- instance of the class

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction
    def __str__(self):
        msg = "Hi, I'm a " + self.size + " " + self.color + " ball!"
        return msg
myBall = Ball("red", "small", "down")
print myBall
```

Hm... How would I describe this house? Upper class, middle class, or lower class?

Polymorphism

Polymorphism—same method, different behavior



```
class Triangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def getArea(self):
        area = self.width * self.height / 2.0
        return area

class Square:
    def __init__(self, size):
        self.size = size

    def getArea(self):
        area = self.size * self.size
        return area
```

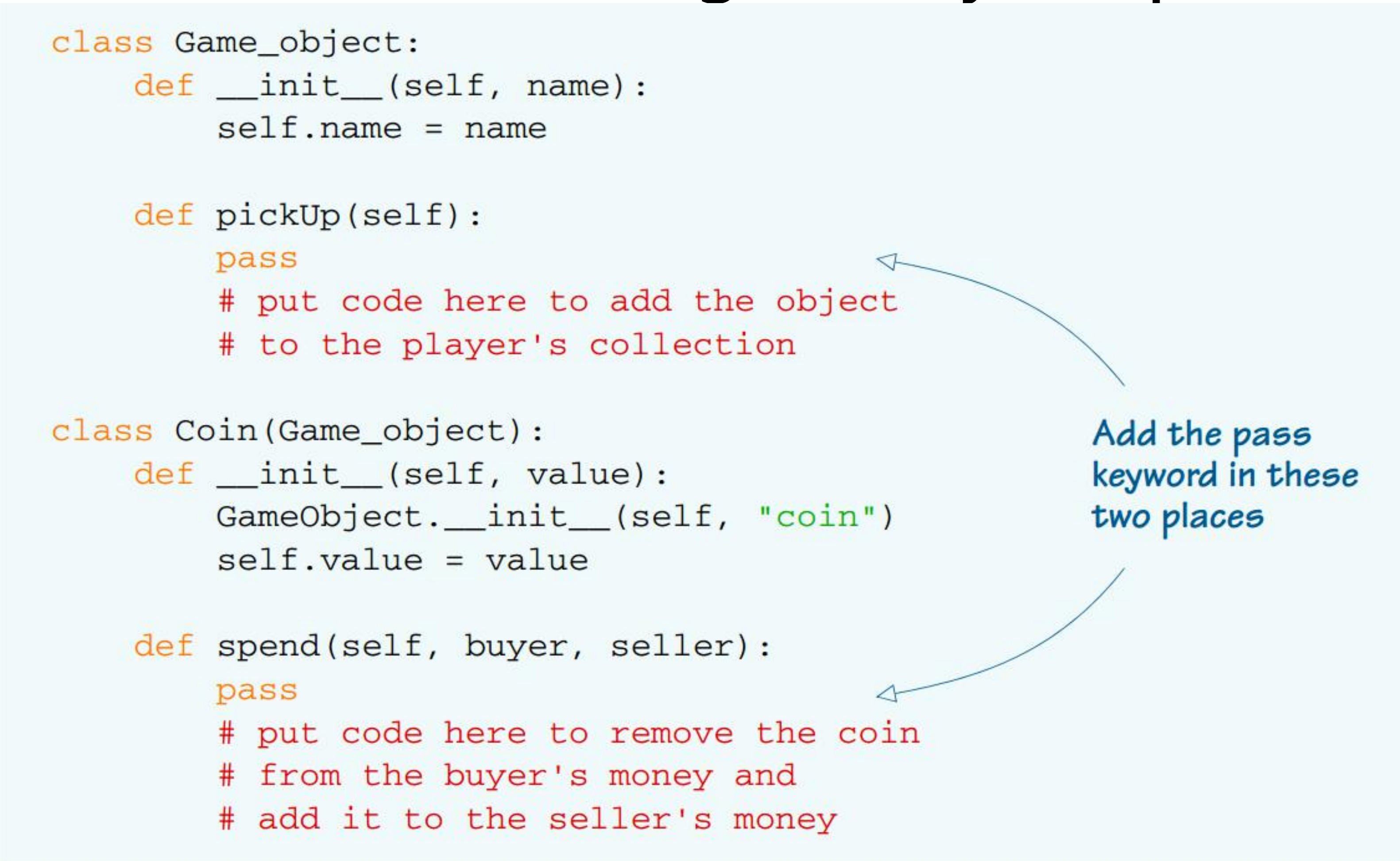
Here is the Triangle class

Both have a method called getArea()

Here is the Square class

Inheritance

Inheritance—learning from your parents



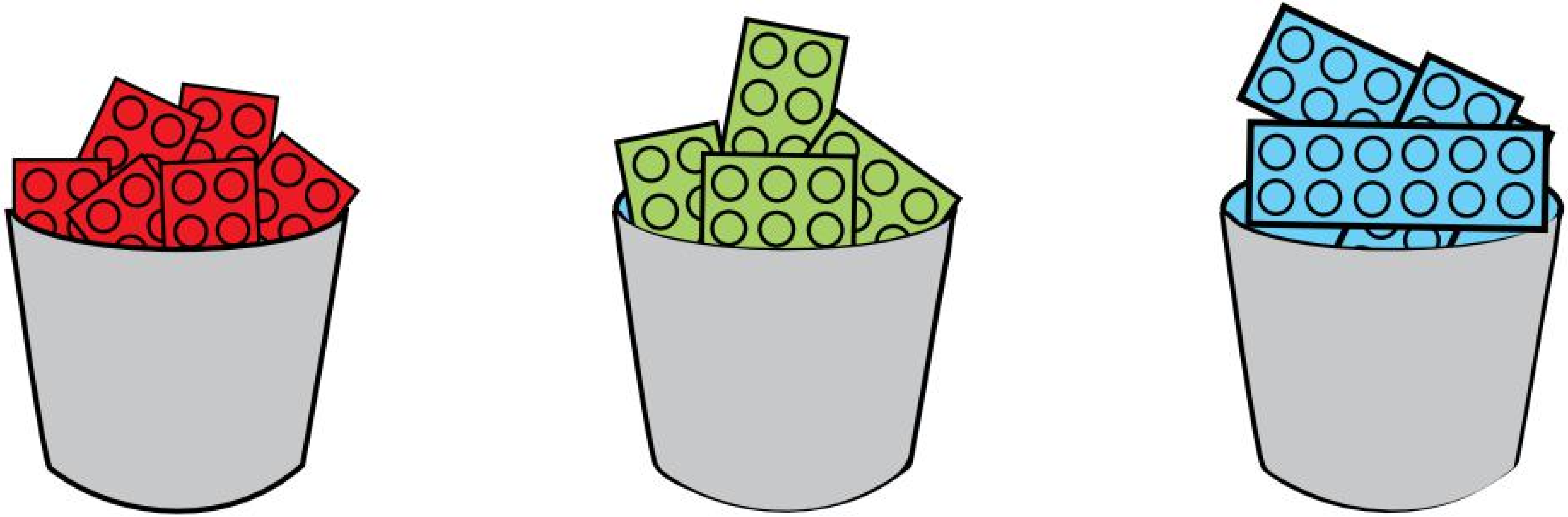
```
class Game_object:
    def __init__(self, name):
        self.name = name

    def pickUp(self):
        pass
        # put code here to add the object
        # to the player's collection

class Coin(Game_object):
    def __init__(self, value):
        GameObject.__init__(self, "coin")
        self.value = value

    def spend(self, buyer, seller):
        pass
        # put code here to remove the coin
        # from the buyer's money and
        # add it to the seller's money
```

Add the pass keyword in these two places

Modules

Modules are smaller pieces of a bigger program

- It makes the files smaller
- Once you create a module, you can use it in lots of programs
- You don't always need to use all the modules together



Create and Use Modules

```
# this is the file "my_module.py"
# we're going to use it in another program
def c_to_f(celsius):
    fahrenheit = celsius * 9.0 / 5 + 32
    return fahrenheit
```

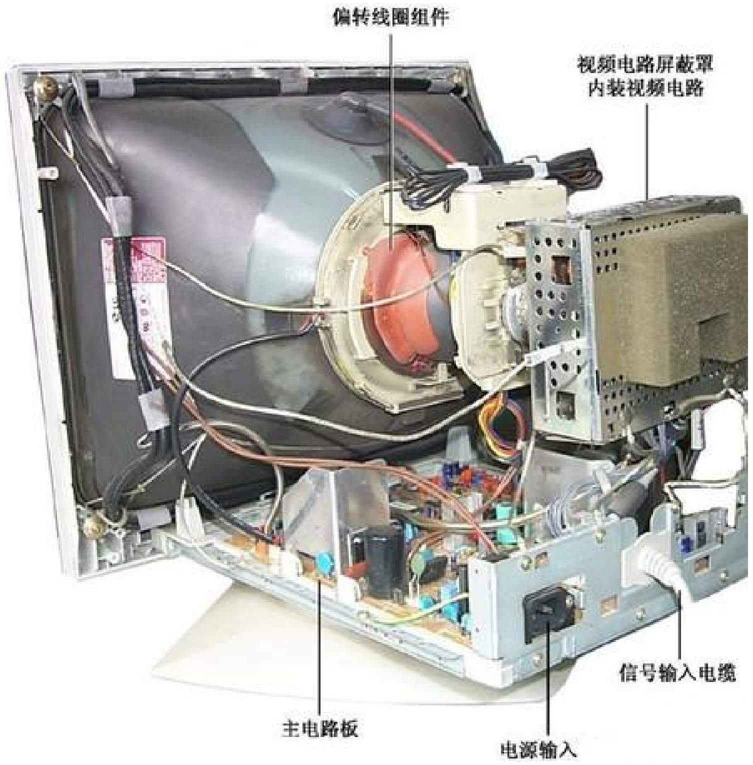
```
>>>import my_module
>>>my_module.c_to_f(90)
```

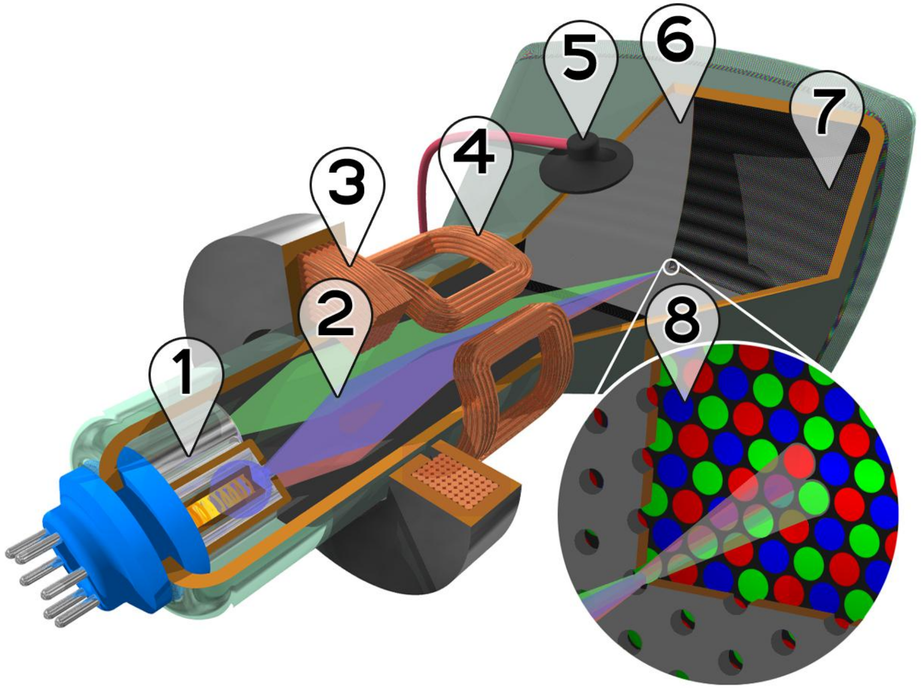
Standard modules

```
>>>from time import sleep
>>>time.sleep(2)

>>> import random
>>> random.randint(0, 100)
4
>>>random.random() * 10
3.61204895736
```

Chapter3 Monitor

Cathode Ray Tube (CRT)



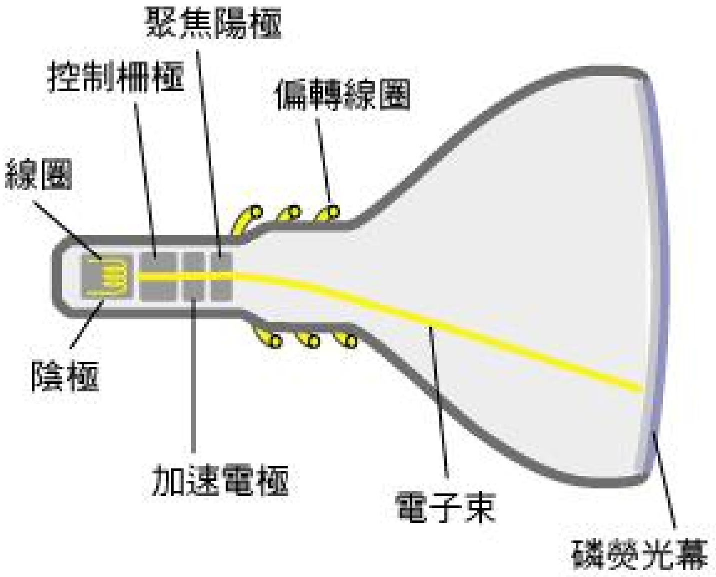


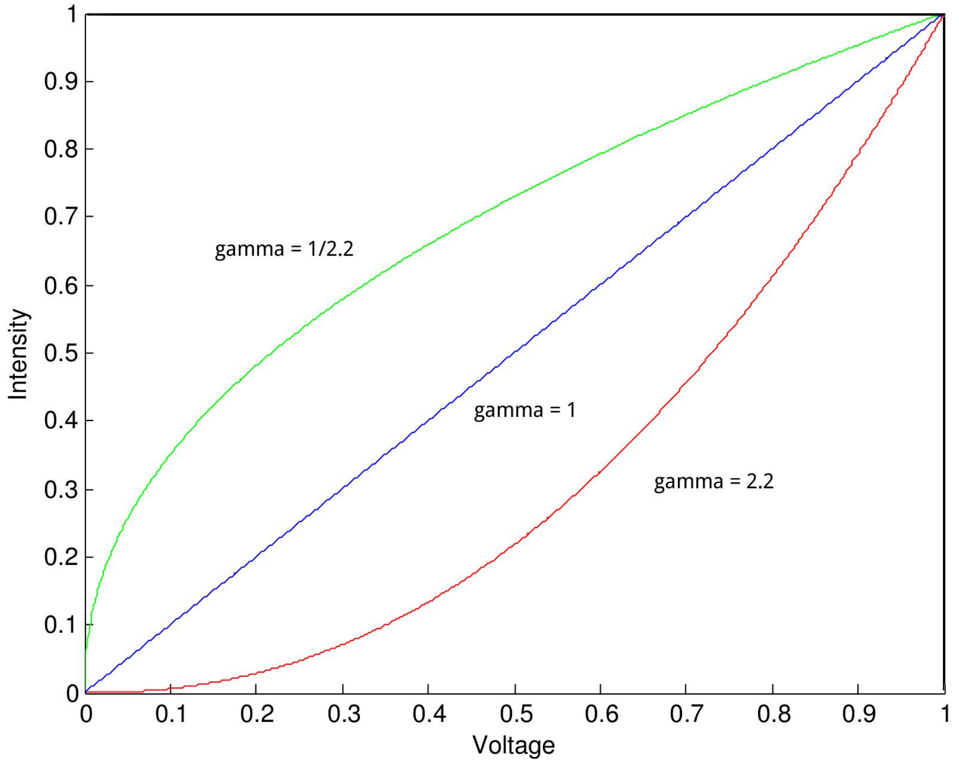
1. Three electron emitters (for red, green, and blue phosphor dots)
2. Electron beams
3. Focusing coils
4. Deflection coils
5. Anode (collector)
6. Mask for separating beams for red, green, and blue part of displayed image
7. Phosphor layer with red, green, and blue zones
8. Close-up of the phosphor-coated inner side of the screen

Monitor Calibratation

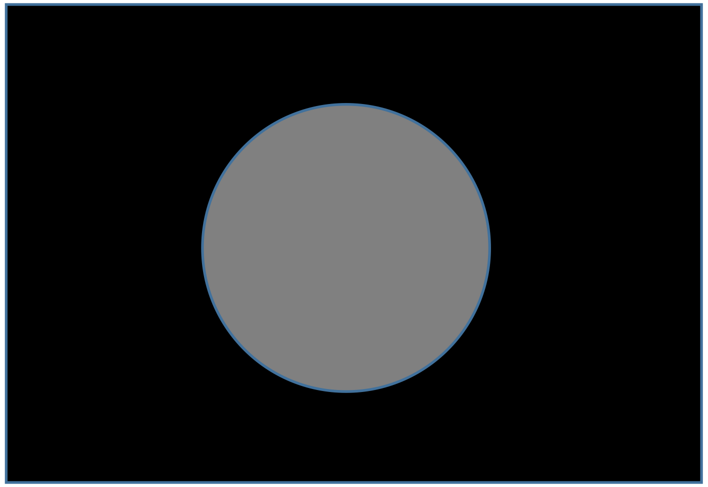
$I = kV$ note: I(intensity), k(constant),V(Voltage)

$I = kV^{\gamma}$ note: gamma ≈ 2.2





Calculate Gamma



```
from psychopy import monitors
import numpy as np

# constants for gamma correction
inp = np.array([0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224,
240, 255])
lum = np.array([6.08, 8.2, 11.4, 15.8, 21.59, 28.73, 37.32, 47.4, 59.14, 71.43,
86.3, 102.4, 120.9, 141.5, 166.7, 188.6, 214.8])
g = monitors.GammaCalculator(inp, lum)
```

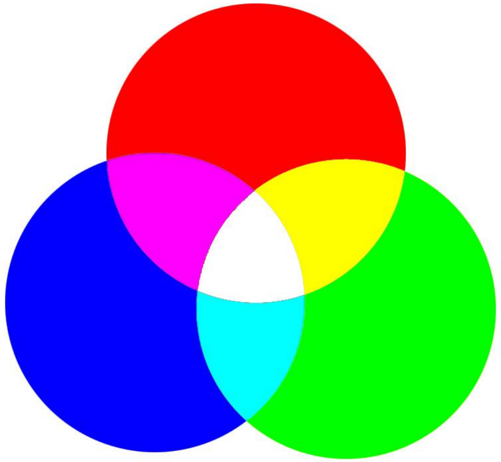
Some index of monitor

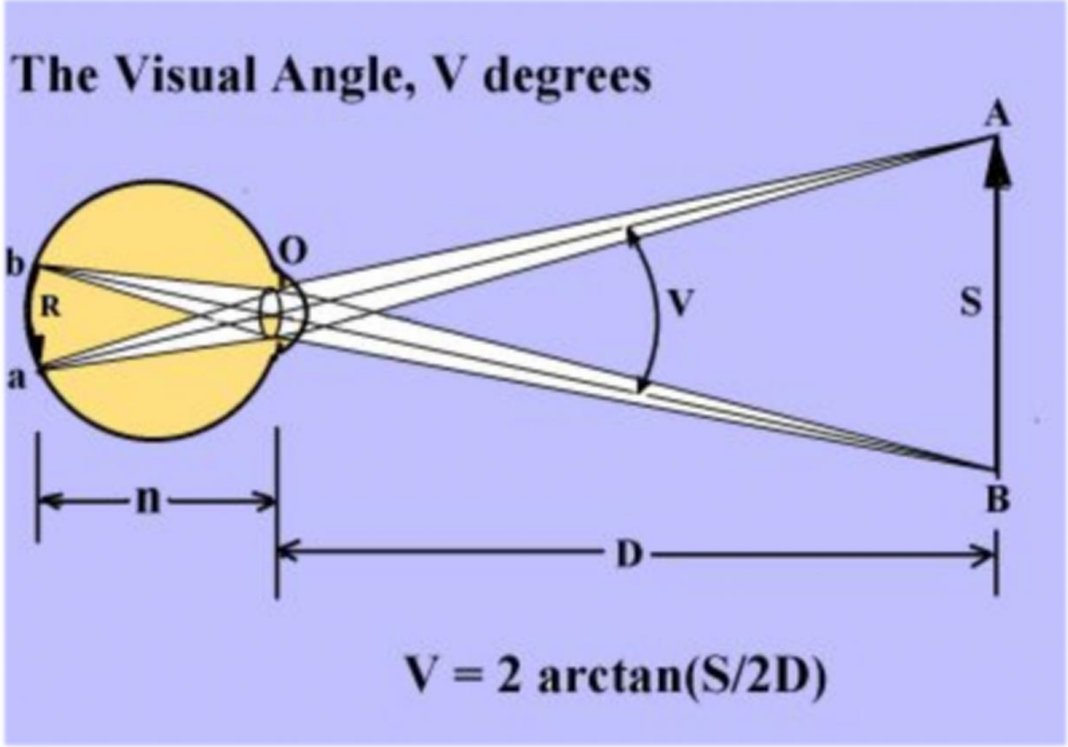
Pixel

Resolution

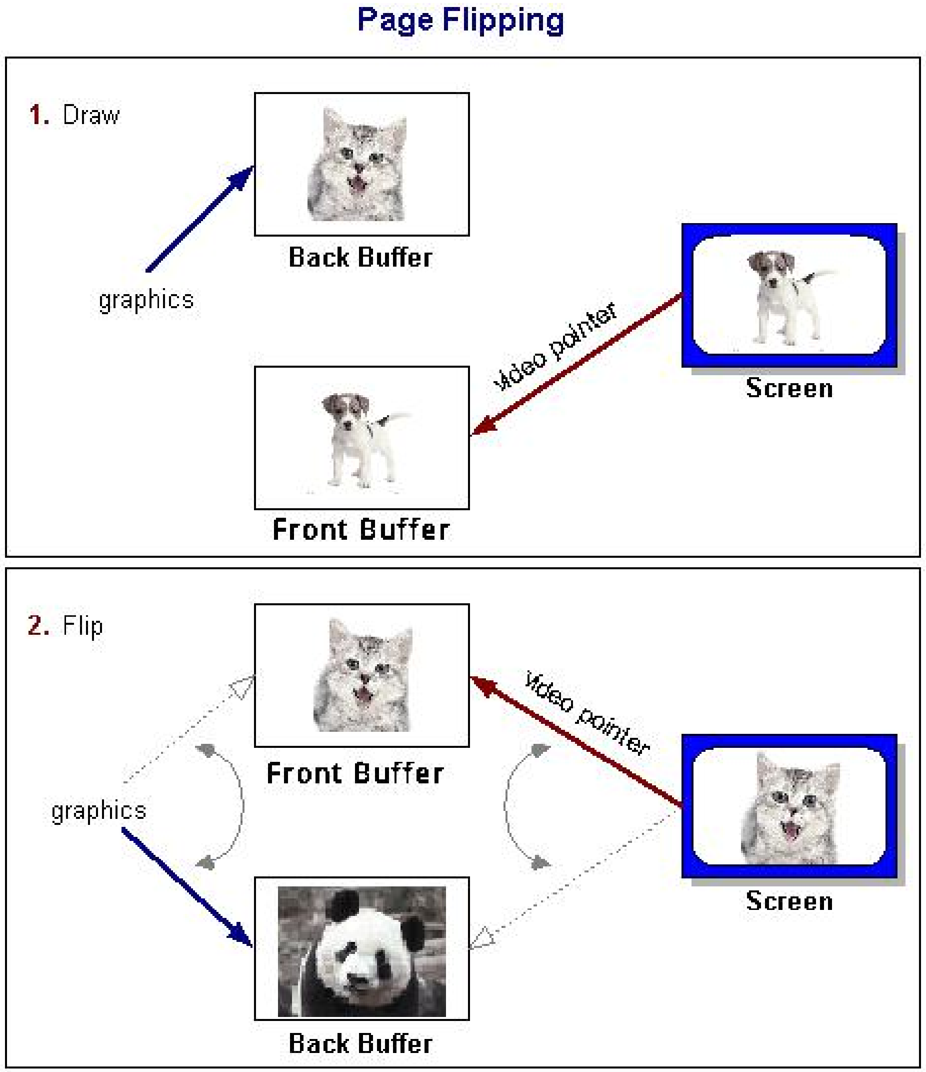
Vertical Scan Frequency/ Refresh Rate / Frame Rate

Visual Angle





Double buffering

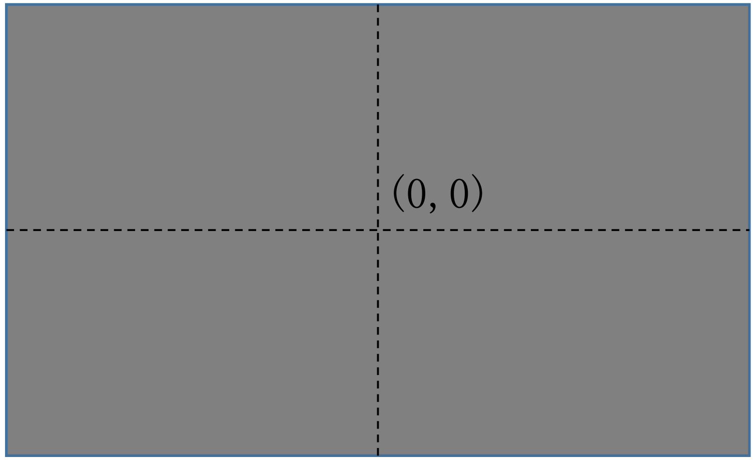


In the page-flip method, both buffers are capable of being displayed (both are in VRAM)
At any one time, one buffer is actively being displayed by the monitor(Front Buffer), while the other, background buffer is being drawn(Back Buffer).When drawing is complete, the roles of the two are switched
The page-flip is typically accomplished by modifying the value of a pointer to the beginning of the display data in the video memory

Chapter4 Introduction to Psychopy Functions

Window & TextStim

```
>>>from psychopy import visual
>>>myWin = visual.Window(size=(800,600), color=(128,128,128),
fullscr=False, units='pix', blendMode='avg', colorSpace='rgb255',
gamma=2.2)
>>>msg1 = visual.TextStim(myWin, text=u"Hello world!", pos=(0,0),
color='red', bold=True, height=18)
>>>msg1.draw()
>>>myWin.flip()
>>>myWin.close()
```



ImageStim

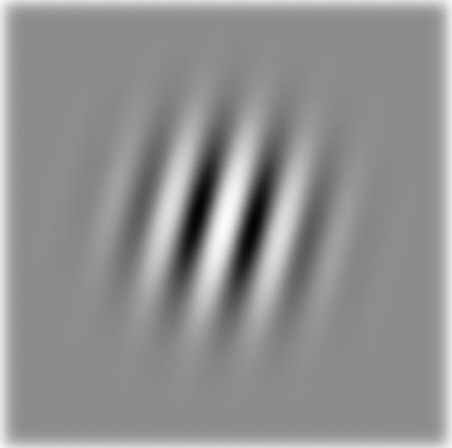
```
from psychopy import visual
myWin = visual.Window(size=(800,600), color=(128,128,128),
fullscr=False, units='pix', blendMode='avg', colorSpace='rgb255',
gamma=2.2)
image = visual.ImageStim(myWin,image='face.jpg', mask=None,
pos=(0.0, 0.0), size=(1.0, 1.0), ori=1)
for i in range(100):
    image.draw()
    myWin.flip()
myWin.close()
```

Line, Rect, Dot, Circle

```
from psychopy import core, visual, event
myWin = visual.Window(size=(800,600), color=(128,128,128),   fullscr=False, units='pix',
blendMode='avg', colorSpace='rgb255')
line = visual.Line(myWin, start=(-400,0), end=(400,0), lineColor='black', lineWidth=4)
rect = visual.Rect(myWin,width=300,height=300, lineColor='black', lineWidth=4,
fillColor='green')
circle = visual.Circle(myWin, radius=90, edges=36, lineColor='black', lineWidth=4,
fillColor='red')
dot = visual.DotStim(myWin, units='pix', nDots=1, dotSize=8, color='black')

while not event.getKeys(keyList=['escape','q']):
    [ s.draw() for s in [line, rect, circle, dot] ]
    myWin.flip()
myWin.close()
```

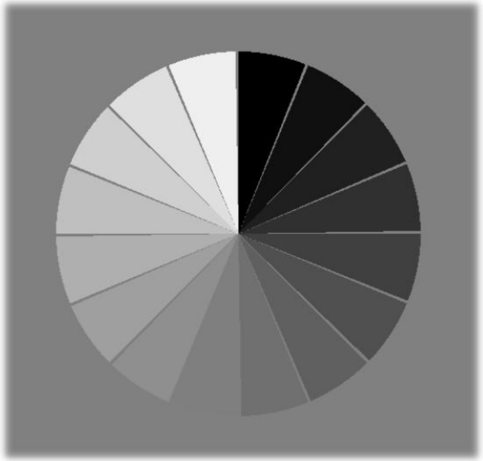
GratingStim

```
from psychopy import core, visual, event
myWin = visual.Window(size=(800,600), color=(128,128,128),   fullscr=False, units='pix',
blendMode='avg', colorSpace='rgb255')
fix = visual.TextStim(myWin, text='+', pos=(0,0), color='black', bold=True, height=40)
gaborL = visual.GratingStim(myWin, tex='sin', mask='gauss', pos=(-200,0), size=100, sf=0.1,
ori=45)
gaborR = visual.GratingStim(myWin, tex='sin', mask='gauss', pos=(200,0), size=100, sf=0.1, ori=0)
Ori = 45
while not event.getKeys(keyList=['escape','q']):
    gaborL.ori = Ori
    Ori+=1
    [ s.draw() for s in [gaborL, gaborR, fix] ]
    myWin.flip()
myWin.close()
```



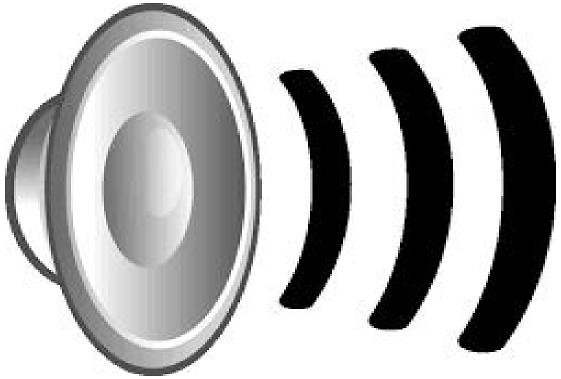
RadialStim

```
from psychopy import core, visual, event
myWin = visual.Window(size=(800,600), color=(128,128,128),   fullscr=False, units='pix', blendMode='avg', colorSpace='rgb255')
fix = visual.TextStim(myWin, text='+', pos=(0,0), color='black', bold=True, height=40)
wdg = visual.RadialStim(myWin, tex='None', visibleWedge=(0,360.0/16), size=400)
while not event.getKeys(keyList=['escape','q']):
    for i in xrange(16):
        ori = i*360./16
        col = -1 + i*16.0/256*2
        wdg.setOri(ori)
        wdg.setColor((col, col, col))
        wdg.draw()
    myWin.flip()
myWin.close()
```

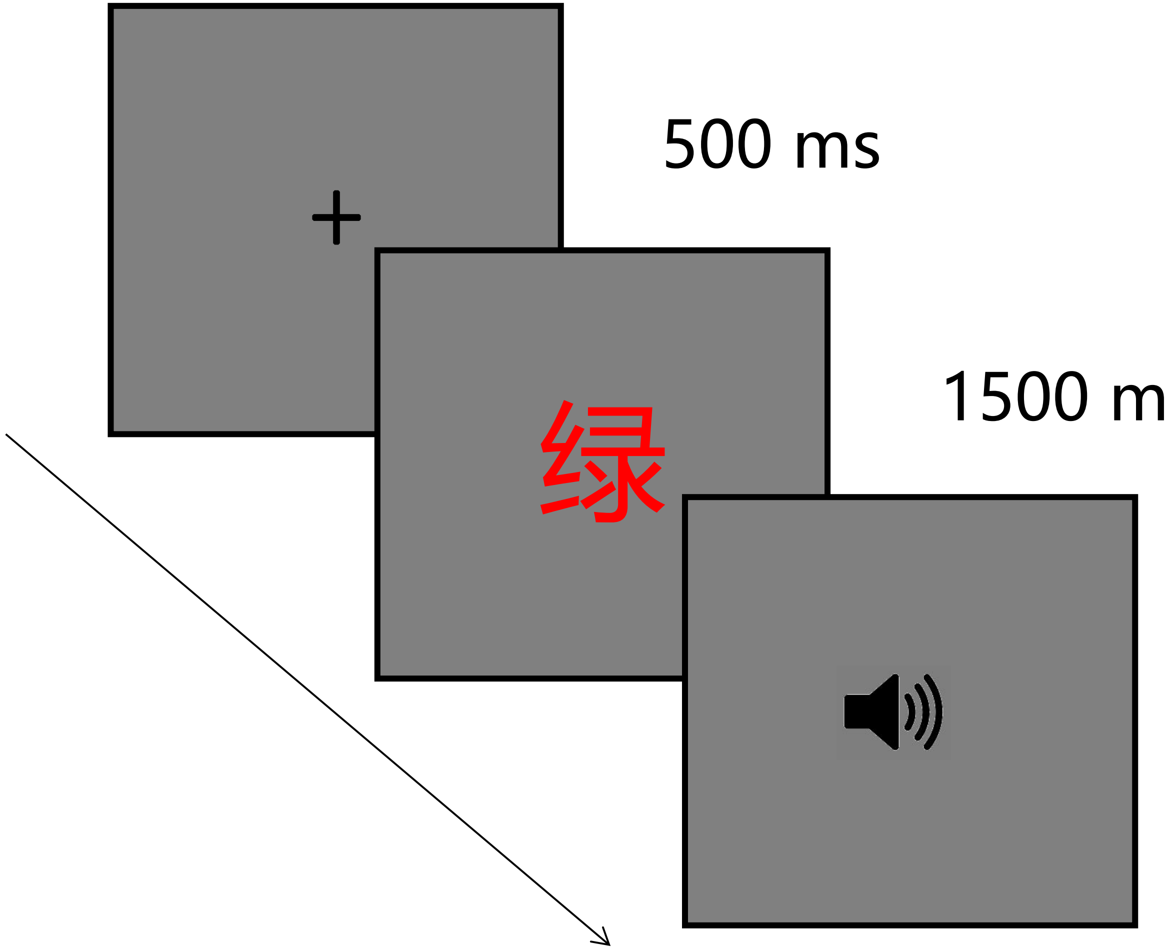


Sound

```
from psychopy import core, sound
beep = sound.Sound(value=2000,secs=0.2,sampleRate=44100,
bits=8)
qq = sound.Sound(value='msg.wav',secs=0.2,sampleRate=44100,
bits=8)
beep.play()
core.wait(0.5)
qq.play()
core.wait(0.5)
core.quit()
```



An Example of Exp.~Stroop



500 ms

1500 ms or untill response

False alarm

THANKS!

A&Q