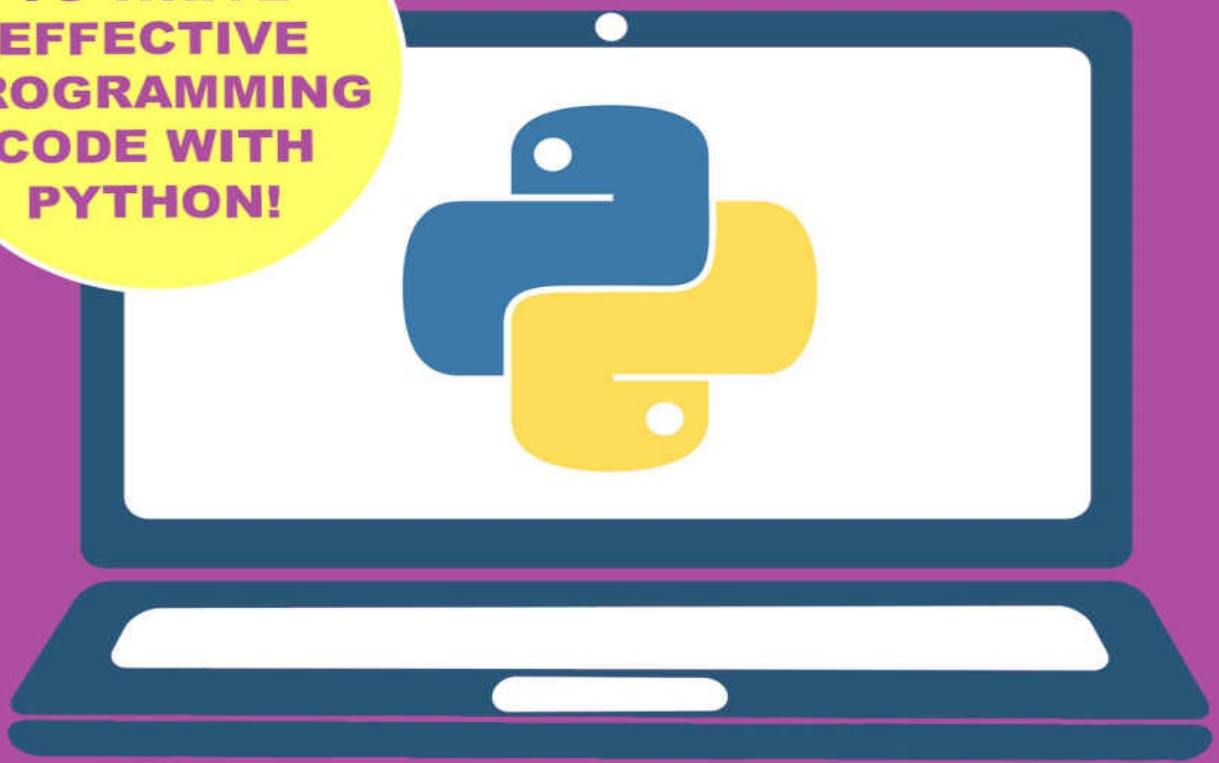
Python

TIPS AND TRICKS TO PROGRAMMING CODE WITH PYTHON

LEARN TO WRITE EFFECTIVE PROGRAMMING CODE WITH PYTHON!



CHARLIE MASTERSON

Python:

Tips and Tricks to Programming Code with Python

Charlie Masterson

Table of Contents

Introduction

Congratulations on downloading *Python: Tips and Tricks to Programming Code with Python* and thank you for doing so.

The following chapters will discuss some tips and tricks to help you get the most out of Python. This book will go over things that can improve your programming outcome with Python. All of you work will improve with this book.

Each chapter will cover a different tip with coding that you can use today. In the first chapter you will learn how to change the color of your images. In chapter two you will learn how to make better data graphics. In chapter three you will learn how to use code to send and check your e-mails, as well as receive text messages. In chapter four you will learn how to manipulate your images. In chapter five you will learn how to schedule things for you program to do. In chapter six you will learn how to make an animated banner. In chapter seven you will learn how to code games.

This book can provide any programmer with some interesting and fun codes to learn from.

This book will help the already experienced programmer more, but a beginner can learn a lot from it too. You're sure to find lots of helpful information within these pages, so let's begin.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, please enjoy!

Chapter 1:
Image Colors

Let's start this with some fun image programming. It's always nice to have some fun programming up your sleeve, and this code will give you just that.

In this chapter we will look at codes to help you change the color of an image, so you can make it look the way you want it to. This code will give you some fun things to do to your images when coding. It's also just a fun thing to play around with.

1. Type the following in a Python IDLE window.

CODE:

```
from ImageLibrary import *
myfile=pickAFile()
pict=makePicture(myfile)
show(pic)
```

This tells Python to use every function that is in your image library.

This should not be done in a main window. You need to write this in a new window. Do this by choosing, "file -> New Window" and then paste the code in an empty window, then pick "run". You will then be asked if want to save your file. The file needs to be saved in an arbitrary location.

The " from ImageLibrary import* " needs to be placed into every IDLE window. myfile=pickAFile() lets the computer know you need a window that you can pick the .jpg image file that you want to play with.

The picture should be small so that everything doesn't take a long time. pickAFile() will give you a return of the full name of your image file.

2. Ask Python to print myfile so you will be able to see how the image looks. As a reminder, print command looks like print myfile.

The string should look like:
C:/Users/Public/Pictures/image.jpg

That means myfile is a string that states the picture that you want. You could use a string so you won't have to call pickAFile() each time that you need anything new.

Like this:
pic=makePicture("/users/nameGoesHere/Pictures/image.jpg")

Make your picture:

```
from ImageLibrary import *
filename=pickAFile()
pic=makePicture(filename)
show(pic)
```

This will load you picture into memory so you can change it. You don't need to call makePicture(myfile) unless you plan on putting the results in a useful place such as assigning it to a variable such as pic .

Make sure that you get the lower and upper case letters right because Python is case sensitive.

Whenever you change something about the picture, you have to call show(pic) so you will be able to see your changes. Your changes do not update automatically when they are changed.

3. Open the image and display it

CODE:

```
from ImageLibrary import *
myfile=pickAFile()
pic=makePicture(myfile)
show(pic)
```

First thing you are going to do is make every red pixel level to 255:

```
for p in getPixels(pict):
        setRed(p,255)
show(pic)
```

3a. Change red, green, and blue levels until the image is black, then white, the green.

Another neat function is setColor(pixel,color) will change a pixel color. The list of predefined colors are cyan, magenta, pink, orange, yellow, darkGrey, lightGrey, grey, blue, green, red, black, white. You can also use this function to change a pixel color to orange instead of the regular blue, red, and green colors.

3b. Change the picture color using the for loop, setColor (pixel, color) , and then you can use the colors that have already been defined.

You can also use makeColor(redValue, greenValue, blueValue). This lets you create any color you want. To make purple you use: Purple=makeColor (255, 0, 255)

3c. Play with r, g, b values to figure out how to create yellow color. Use the earlier code to make the change to the complete picture.

You can use the makeLighter (color) function to lighten parts of color. It actually subtracts 40 from each main color green, red, and blue. This also can be used to lighten the whole photo.

```
for p in getPixels(pic):
        col=getColor(p)
        makeLighter(col)
        setColor(p, col)
```

You can also use makeDarker(color) that will make the picture darker.

3d. Apply the makeDarker to each pixel three times.

What will happen?

You may possibly see an error such as:

CODE:

```
Traceback (most recent call last):
  File "/User/name/Documents/jmss/exercise1.py", line 14 <module>
    show(file)
  File "/User/name/Documents/ImageLibrary.py", line 230, in show
    resolution = (picture.getWidth(), picture.getHeight ())
AttributeError: 'str' object has no attribute 'getWidth'
```

If this is the case, don't panic. Look at the line number and filename. They refer back to the file. This is talkingabout show(file) . With this error you can see show was being called with file - and not that of a picture.

Many of your errors will probably look like this; innocent mistakes that you can easily fix.

Now you know this neat tip on how to change the coloring of an image just in case you want to. This is a great tool to have when you want to add some fun images to your website or app.

Chapter 2:
Text Messages and Emails

Having to take the time to reply to several emails all at once isn't exactly fun. Unfortunately you can't really write a program to take care and reply to all emails because each email normally requires an individual response.

But do not worry as there is still a lot of ways that you can do to *automate* email tasks.

As an example, if you have a list of customer information, and then need to send all those customers an email that will depend on their location and age, you may be able to use commercial software, but as an alternative you can also write code to send those emails which will save you time instead of having to copy and paste individually.

In this chapter we dive into ways on how we can program code using Python that involves email and text messages.

There's a way to write a Python program that will let you receive SMS texts and emails to let you know events even if you aren't sitting in front of your computer.

Let's say you have started a task that will take a few hours to finish. You're not going to want to keep checking your computer regarding its status every now and then. The program will send you a text message to let you know when the task is finished, allowing you to direct your attention towards something else, even if it means being away from your computer.

Using Python to access your email

You can write Python code that triggers a function which does everything the SMTP or Simple Mail Transfer Protocol used to send and receive emails is expected to do.

The following shows you how.

CODE:

```
import smtplib as smtp
smtp0bj = smtp.SMTP('smtp.yours.com', 465)
smtp0bj.ehlo()
smtp0bj.starttls()
smtp0bj.login('buddy@example.com', 'yourPasswordHere')
smtp0bj.sendmail ('yourEmail@yahoo.com', 'receiverEmail@yahoo.com', 'Subject: Goodbye.\nThat was a great get-together! Very much enjoyed it.')
smtp0bj.quit()
```

Naturally you can't just type the code as is because temporary placeholders are used

where the personal information will go. You will need to fill in your own information before you can make it work to your needs.

As I continue, I will go over how to put sample information into those placeholders so that you can connect and log into your SMTP server, send your email, and then have it disconnected.

If you have ever setup a program to connect to your email account then you are probably familiar with configuring the SMTP port and server. Otherwise you can always make some research online on how this works.

Each email provider will have different settings, but you can easily search <your provider> smtpsettings and you should be able to find the port and server.

After you have figured out your domain name and port you can create a SMTP object:

CODE:

```
smtpInst = smtplib.SMTP('smtp.gmail.com', 465)
type(smtpInst)
<class 'smtplib.SMTP'>
```

If your call isn't successful then the SMTP may not support the TLS or Transport Layer Security on port 465, so you should try changing 465 to 587.

Once you have setup the SMTP you can run the ehlo() method. This needs to be run as soon as you have finished writing the SMTP object.

```
smtpInst.ehlo()
```

If you're using port 587 then you will have to call the starttls() method next.

```
smtpInst.starttls()
```

Now you're ready to login to your e-mail server:

```
smtpInst.login(' yourEmail@yahoo.com ', 'yourPassword')
```

Password security

Be careful when including email passwords in the code. If somebody copies your program they may be able to access your e-mail. It's best if you call the function input() where you have to put in the password. Though it might feel like an inconvenience having to type in your password every time the program is ran, it will keep your email from being copied.

Now you are ready to send an e-mail.

smtpInst.sendmail(‘ your_email@gmail.com ‘, ‘ receiver@example.com ‘, ‘Subject: Goodbye. \nThat was a great get-together! Very much enjoyed it.’)

Then you need to add code in order to disconnect.

smtpInst.quit()

You can also retrieve and delete emails in Python with IMAP by using this code:

CODE:

```
import imapclient as imapc
imapInst = imapc.IMAPClient( ‘imap.yahoo.com’, ssl=True)
imapInst.login(‘email@yahoo.com ‘, ‘ yourPasswordHere ‘)
// ‘email@yahoo.com Firstname Lastname authenticated (Success)’
imapInst.select_folder(‘INBOX’, readonly=True)
UIDs = imapInst.search([ ‘SINCE 14-Jan-2016’])
rawMsg = imapInst.fetch([UID], [ ‘BODY[]’, ‘FLAGS’])
import pyzmail as pyzm
msg = pyzm.PyzMessage.factory(rawMessages[UID][ ‘BODY[]’])
msg.get_subject()
message.get_addresses(‘from’)
[( ‘John Doe’, ‘jdoe@sample.com’)]
message.get_addresses( ‘to’ )
[('Firstname Lastname', ‘fnamelname@yahoo.com’)]
message.get_addresses( ‘cc’ )
message.get_addresses( ‘bcc’ )
message.text_part != None
message.text_part.get_payload().decode(message.text_part.charset)
message.html_part != None
message.html_part.get_payload().decode(message.html_part.charset)
imap0bj.logout()
```

As you go through these steps, you should actually see various little output lines appear which tell you the relevant information in question.

Using Twilio to receive text messages

Now let’s look at how to have it send you texts. First you will need to go to the Twilio website and open a Twilio account. Twilio is a platform that offers communication services such as messaging and video. Once you have finished your registration, enter the following code into an interactive shell. Make sure that you fill in your actually account information into the corresponding spaces.

Below is what you would program:

CODE:

```
from twilio.rest importTwilioRestClient
acctSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
auth = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
twilioc = TwilioRestClient (acct, auth)
myNumber = '+11234567890'
myCell = '+15556667777'
msg = twilioc.messages.create(body='Mrs. Clause - I need you here - We are short on cookies.',
from=myTwilioNumber, to=myCellPhone)
```

You can expect a text shortly after running that code which will say something along the lines of *Mrs. Claus – I need you here – We are short on cookies ,* after the message starts by telling you of your trial for using Twilio.

Now you know some tips to help you use Python in sending emails and receiving text messages. This should help you make your work just a little easier and learn the capabilities of the Python programming language.

Chapter 3:
Image Manipulation

Earlier I showed you how to change the color of images, so now we're going to look at other ways to manipulate your images by cropping, flipping, and other fun manipulations using Python.

Image properties

You can find lots of important information about an image file like, height, width, filename, and format. Make sure that you use the filename of an image that you have on your computer and replace the one in the code.

Enter the following in your Python IDLE and run it:

CODE:

```
from PIL import Image as Imgl
carIm = Imgl.open('car.jpg')
carIm.size(816, 1088)

imgWidth, imgHeight = carIm.size
```

Values of the variables after running the code:

imgWidth:
816

imgHeight:
1088

carIm.filename:
'car.jpg'

carIm.format:
JPG

carIm.format_description:
'Joint Photographic Experts Group'

Then we end the code by saving the image:

```
carIm.save('car.jpg')
```

Background color

Now let's play with the color of the background.

```
from PIL import Image as Imgl
img = Imgl.new( 'RGBA', (150, 300), 'red' )
img.save('red.png' )
```

You will get an object that works for images that are 300 pixels tall and 150 pixels wide, and then has a background that is red.

Cropping an image

Now let's look at how to crop an image. With cropping, the original image is not changed, instead the original is left and a new image is returned.

Enter this code:

```
cropIm = carIm.crop((330, 340, 550, 530))
cropIm.save('cropimage.png')
```

Copy and Paste

The following will allow you to copy and paste an image onto another image.

First enter the following:

```
carIm = Image.open( 'car.png' )
carCopyIm = carIm.copy()
```

Both of these variables have two images. Since you now have a picture that is saved as carCopyIm , you change carCopyIm and then save them with a different filename.

Now let's paste another image onto an image:

```
faceImg = carIm.crop((330, 340, 509, 599))
faceImg.size
(220, 210)
carCopyIm.paste(faceIm, (0, 0))
carCopyIm.paste(faceIm, (400, 500))
carCopyIm.save( 'carPasted.jpg' )
```

Just so you know, the computer clipboard is not used when you copy and paste this way.

Image Tile

Now maybe you want to tile your image across the whole image. Continue your code with this:

```
carImWidth, carImHeight = carIm.size
faceWidth, faceHeight = faceImg.size
carCopyTwo = carIm.copy()
for l in range(0, carImWidth, faceWidth):
        for t in range(0, carImHeight, faceHeight):
        print(l, t)
        carCopyTwo.paste(faceIm, (left, top))
carCopyTwo.save( 'tiledCar.png' )
```

Resizing your image

Now let's look at some code that will allow you to resize your image.

```
w, h = carIm.size
quarterImg = carIm.resize((int(w / 2), int (h / 2)))
quarterImg.save( 'quartersize.jpg' )
svelteImg = carIm.resize ((w, h + 305))
svelteImg.save('svelteCar.jpg')
```

Rotating images

Now let's look at flipping and rotating images.

```
carIm.rotate(45).save( 'rotated45.jpg' )
carIm.rotate(135).save( 'rotated135.jpg' )
carIm.rotate(225).save( 'rotated225.jpg' )
```

There is a keyword called rotate which will enlarge your photo so that it will fit in the rotated new image:

```
carIm.rotate(5).save( 'rotate5.jpg' )
carIm.rotate(5, expand=True) .save('rotate5expand.jpg')
```

Mirror flipping the image

You can also mirror flip the image as well.

```
carIm.transpose(Image.FLIP_LEFT_RIGHT).save('xflip.jpg')
carIm.transpose(Image.FLIP_TOP_BOTTOM).save('yflip.jpg')
```

You have now learned how to manipulate graphical images any way you want to using Python.

This shows that you have control over your images, so that they can look the best that they can and serve your programming goals.

In the next chapter we discuss ways on how we can use Python in order to manipulate dates and times.

Chapter 4:
Schedules and Times

Everybody has lots of things that they keep on their computer, so why not know how to automate some of these things?

For instance, the clock in your computer is able to help you run code on dates and time periods that you instruct it to.

Timestamps

You can use the following code to retrieve an epoch timestamp. Running this script will display a different time and date depending on when you run it:

CODE:

```
import time as t
t.time()
```

You can use epoch timestamps as profile code to measure the amount of time it will take for a code to run. You can also use time.time() in the beginning of your code and again at the end of where you want it measured.

You will be able to take the first time and subtract it from the second time in order to get the total elapsed time.

Enter this in a new file editor:

```
import time as t
def calculateProduct():
        prod = 1
        for x in range(1, 1000):
        prod = prod * x
        return prod

start = t.time()
        prod = calculateProduct()
end = t.time()
print( 'The final number contains %d integers.' % len(str(prod)))
print( 'Calculation time:\t%s seconds.' % (end – start))
```

Pausing a program

You should save it as calcProd.py and then execute the code.

Now let's look on how you can pause your program:

```
import time as t
numTimes = 4
for x in range(numTimes):
        print("tik")
        t.sleep(1)
        print("tok")
        t.sleep(1)
t.sleep(4)
```

It's important to note that running the Control+C command will not prevent the process from running in the Python IDLE. IDLE will let it run the whole pause process *before* it will perform the Keyboard Interrupt exception.

Programming a stopwatch function

Now who doesn't want to know how to write a stopwatch code so that you can time things when you need it?

This program will track the elapsed time between the press of the ENTER key and will give you the lap time, lap number, and total time.

First you will need to set the program to track time. Write this code into the editor while using a TODO placeholder comment.

```
import time as t

# Show the instructions for the program

print('Enter key: start/lap.\n
Type "EXIT" to quit.' )
raw_input()
print( 'Started.' )
start = t.time()
last = startTime
lap = 1
```

You need to start the first lap, take note of what your time is, then set your count to 1.

Next you need to print and track lap times. Add this to your code:

```
        while raw_input().toUpper() != "EXIT":
        raw_input()
        lapT = round(t.time() – last, 3)
        totalT = round(t.time() – start, 3)
```

```
    print('Current lap: %s: %s [Total: %s]' % (lap, totalT, lapT), end=' ')
    lap += 1
    lastT = t.time() # rest last lap time
    print "The program is finished."
```

Knowing how to program a time tracker will make it easier for you to code other types of programs.

There may be apps that you can get that will do this for you, but you really don't know what you're signing up for and you also have to deal with those incessant ads.

Pausing a program until a specific date

Earlier you learned the code to pause a program for a certain amount of time, here is the code to pause a program until a specific date. In this example I will be using the Christmas date.

```
import time as t
import datetime as dt
christmas2017 = dt.datetime(2017, 12, 25, 0, 0, 0)
while dt.datetime.now() < christmas2017:
    t.sleep(1)
```

Now you know how to automate a lot of different timing programs using Python on your computer to make your work more efficient.

Chapter 5:
Animated Banner

Animated banners can be a fun eye candy graphic effects that you can use. With Python you can create animated banners for your website very easily. Important information will be mentioned in notes within the code.

CODE:

```
import os

import time

displayWidth = 50

#the message you want the banner to read
message = "HELLO! "
drawMessage = ["", "", "", "", "", "", ""]

#dictionary that maps the letter to their seven line banner equivalents
#every let maps to seven strings

charmap = { " ": [ "",
                   "",
                   "",
                   "",
                   "",
                   "",
                   ""],
                    "E" : [ "*****",
                   "*     ",
                   "*     ",
                  "*****",
                  "*     ",
                  "*     ",
                  "*****" ],
                  "H" :  [ "*     *",
                  "*     *",
                  "*     *",
                  "*****",
                  "*     *",
                  "*     *",
                  "*     *" ],
                   "O" : [ "*****",
                  "*     *",
                  "*     *",
                  "*     *",
```

```
            "*      *",
            "*      *",
            "*****" ],
             "L" :  [ "*         ",
            "*         ",
            "*         ",
            "*         ",
            "*         ",
            "*         ",
            "*****" ],
             "!" :  [ "    *    ",
           "    *    ",
           "    *    ",
           "    *    ",
           "    *    ",
           "         ",
           "    *    " ]
        }

#make the marquee. This is done by creating display for each character #in your message, and for each line
for rows in range(len(message)):
        for character in message:
        drawMessage[rows] += (str (charmap [character] [rows]) + " ")

off = WIDTH
while True:
        os.system("cls")
        #print every line of your message, even the offset
        for rows in ran(7):
            print(" " * offset + drawMessage[rows] [max(0,off*-1) :displayWidth – off])
        off -= 1
        if off <= ((len(message)+2)*6)*-1:
             off = WIDTH
        time.sleep (0.05)
```

Now you can put an animated display for your website or in any other special graphic requirements. You can play around with the code to make the look different, and say anything that you want by changing the text.

Bonus Chapter:
Fun Games to Code

You may not think that learning how to code various games is very informative, but you might find that you do learn a lot from this process. It is also a fun way to play around with Python. If you're interested in learning how to code games with Python, this is a great way to do so.

Tic Tac Toe

Here is your first game code. This is built with Python 3, unlike other things in this guide which are built with Python 2. This code in particular is relatively straightforward. It's not very difficult to understand and follow as you go, but here's a brief explanation:

The first thing that we do is define a function for printing the Tic Tac Toe board. This function takes a single argument, *boardState* . This function is passed in essential information relative to the board, such as which spaces are filled, and uses that information in order to print out a working game board. Every turn asks the user which cell number they'd like to print out their given symbol: *x* or *o*.

CODE:

```
#!python3

def printBoard(boardState):

        print "Current board\n"
   rangenum = 3

        for x in range(rangenum):
        print " "
        for y in range(rangenum):
        if boardState[x*3+y] == 0:
        print 'O'
        elif boardState[x*3+y] == 1:
        print 'X'
        elif boardState[x*3+y != -1:
        print boardState[x*3+y] -1
        else:
        print ' '
        if y != 2:
        print "| "

        print
        if x != 2:
        print "----------"
```

```
        else:
        print

def instructions():
        print “Use the given cell numbers to make your move.”
        printBoard(range(2, 10))

def getInput(isTurn):
        v = False
        while not v:
        try:
        usr = input(“Where would you like to place “ + turn + “ (1-9)? “)
        usr = int(usr)
        if not usr < 1 and not usr > 9:
        return usr - 1
        else:
        print “You can't do that. Try over. \n”
        instructions()
        except Exception as exc:
        print usr + “ is not a valid move! Please try again. \n”

def checkWin(currentBoard):
        winCondition = ((1, 2, 3), (4, 5, 6), (7, 8, 9), (1, 4, 7), (2, 5, 8), (3, 6, 9), (1, 5, 9), (3, 5, 7))
        for x in winCondition:
        try:
        if currentBoard[x [0]-1] == currentBoard[x [1]-1] and currentBoard [x [1]-1] == currentBoard
[x [2] -1]:
        return currentBoard[each[0] -1]
        except:
        pass
        return -1

def quitGame(currentBoard, message):
        printBoard(currentBoard)
        print message
        quit()

def main():

        instructions()

        boardSize = 9
        for x in range(boardSize):
        boardSize.append(-1)
```

```
        hasWon = False
        currentMove = 0

        while not hasWon:

      printBoard(currentBoard)
      print "Turn number " + str(move+1)
      if currentMove % 2 == 0:
            turn = 'X'
      else:
            turn = 'O'

        #receive information from the player
        usr = input(currentTurn)

   while currentBoard[usr] != -1
        print "Invalid move! Cell already taken. Please try again. \n"
        usr = input(currentTurn)

     currentBoard[usr] = 1 if turn == 'X' else 0

      currentMove += 1
      if currentMove > 4:
            win = checkWin(currentBoard)
            if win != -1
            output = "The winner is"
            output += "X" if winner == 1 else "O"
            output += " : ) "
            quitGame (board, out)
            elif move ==0:
            quitGame(currentBoard, "Looks like a draw to me.")

if _name_ == "_main_"
        main()
```

A Tetris clone

The next game we're going to code is Tetris. A lot of people enjoy playing Tetris, so this may be something that you may have played a lot. You have to install PyGame in order to follow this tutorial. PyGame features a set of modules made for the purpose of programming video games.

This one's actually the most in-depth project we'll be doing in this book, but at the same

time, it's one of the most freeform. In saying that, I mean that there are a lot of *concepts* covered in this chapter that, should game design be your thing, you can use in order to fuel other concepts. For example, you learn how to draw a block with a given length and width via the pixel drawing tools provided by PyGame. You'll also be able to somewhat pick your way through this as we go, but I'll give a quick overview of the primary functions and portions.

The first thing that we do is import all necessary modules. This is relatively straightforward and doesn't take too much explanation. I import them with single letter names for brevity in the sample code, but you can maintain their whole names if you'd like. Also, to keep things orderly, I do them on separate lines.

We then start by declaring global variables that are essential for running the program. These include declaring the frame speed (without declaring this, your game runs relative to the speed of your CPU, which means it could go way faster than intended for some people!)

We also declare the dimensions of the window and the size of the boxes which define our board and make up our shape here, as well as fall trajectory.

Afterwards we define our colors using RGB values. You can change these to whatever you like if you wish. As it stands, we've got a really cool sort of '80s nostalgia feel to it.

After declaring all of that, we declare our shape templates. These contain the data which define our block structures, as well as their permutation data for when the player tries to rotate a block.

Then we write our two primary *run* and *main* functions. These contain a lot of the actual logic of the game, as well as the game loop itself. In case you didn't know, what a game loop basically means is a *set of code which is run over and over until a certain condition is met*. This condition varies by game, but is generally something like *until the player types "exit"* or *while a winner hasn't been declared*, to put it into pseudo code. The rest of this code is pretty self-explanatory in its mission.

In case you somehow haven't played Tetris, the goal of this Tetris clone - called "Tetro" - is to go for as long as possible putting blocks together. What you're trying to do is fill every row up completely, side to side. Once a row of blocks is completed, that row is eliminated. If at any point a column of blocks extends past the top of the screen, then you lose. The rate of the blocks falling gradually speeds up as the game goes on and you gain points. This game, though slow and methodical at first, can become quite the frenzy of frantic block placement!

Anyway, without much further ado, let's go ahead and get to the actual bulk of the game

code. It’s very much a long one but worth checking out.

CODE:

```
import sys as s
import time as t
import random as r
import pygame as p
from p.locals import *


frames = 30
width = 1280
height = 720
sizeofbox = 25
bwidth = 10
bheight = 20
empty = ' '
sidewaysRate = 0.15
downRate = 0.1
xmarg = int((width - bwidth * sizeofbox) / 2)
topmarg = height - (bheight * sizeofbox) - 5


lgrey = (245, 245, 245)
grey = (215, 215, 215)
black = (0, 0, 0)
salmon = (232, 103, 103)
lsalmon = (255, 176, 176)
lavender = ( 143, 147, 193)
llavender = (176, 182, 255)
turq = (0, 155, 135)
lturq = (0, 193, 168)
orangecream = (244, 182, 157)
lorangecream = (255, 198, 176)


borderCol = llavender
bgCol = black
textCol = lgrey
textShadowCol = grey
colorset = (  turq,   lavender,  salmon,    orangecream)
lightcolorset = (lturq, llavender, lsalmon, lorangecream)
assert len(colorset) == len(lightcolorset)


twidth = 5
theight = 5
```

```
stemp = [['     ',
         '     ',
         '  .. ',
         ' ..  ',
         '     '],
        ['     ',
         '  .  ',
         '  .. ',
         '    .',
         '     ']]


ztemp = [['     ',
         '     ',
         ' ..  ',
         '  .. ',
         '     '],
        ['     ',
         '  .  ',
         ' ..  ',
         '  .  ',
         '     ']]


otemp = [['     ',
         '     ',
         ' ..  ',
         ' ..  ',
         '     ']]


jtemp = [['     ',
         '  .  ',
         ' ...  ',
         '     ',
         '     '],
        ['     ',
         '  .. ',
         '  .  ',
         '  .  ',
         '     '],
        ['     ',
         '     ',
         '     ',
         ' ...  ',
         '    .',
         '     '],
        ['     ',
         '  .  ',
         '  .  ',
         '  .. ',
```

```
        '    ']]


ltemp = [['    ',
        '  . ',
        ' ... ',
        '    ',
        '    '],
       ['    ',
        '  . ',
        '  . ',
        '  .. ',
        '    '],
       ['    ',
        '    ',
        ' ... ',
        '  .  ',
        '    '],
       ['    ',
        ' .. ',
        '  . ',
        '  . ',
        '    ']]


ttemp = [['    ',
        '  . ',
        ' ... ',
        '    ',
        '    '],
       ['    ',
        '  . ',
        '  .. ',
        '  . ',
        '    '],
       ['    ',
        '    ',
        ' ... ',
        '  . ',
        '    '],
       ['    ',
        '  . ',
        ' .. ',
        '  . ',
        '    ']]


itemp = [['  .  ',
        '  . ',
        '  . ',
```

```
        '  .  ',
        '     ''],
       ['     ',
        '     ',
        '     ',
        '.... ',
        '     ',
        '     ',
        '     ']]


pieceSet = {"S": stemp,
            "Z": ztemp,
            "J": jtemp,
            "L": ltemp,
            "I": itemp,
            "O": otemp,
            "T": ttemp}


def main():
    global frameclock, dispsurf, font, lfont
    p.init()
    frameclock = p.time.Clock()
    dispsurf = p.display.set_mode((width, height))
    font = p.font.SysFont("Arial", 24)
    lfont = p.font.SysFont("Arial", 110)
    p.display.set_caption('Tetro')


    drawTextLarge('Tetro')
    while True:
        run()
        drawTextLarge('You lose.')


def run():
    newboard = returnEmptyBoard()
    lastDown = t.time()
    lastSideways = t.time()
    fallt = t.time()
    moveDown = False
    moveLeft = False
    moveRight = False
    score = 0
    currentLevel, fallRate = getCurrentLvlAndFallRate(score)


    activePiece = getPiece()
    followingPiece = getPiece()
```

```
    while True:
        if activePiece == None:
            activePiece = followingPiece
            followingPiece = getPiece()
            fallTimer = t.time()



        if not validPos(newboard, activePiece):
            return



        checkQuit()



        for evx in p.event.get():
            if evx.type == KEYUP:
                if (evx.key == K_p):
                    dispsurf.fill(bgCol)
                    drawTextLarge("Game is paused")
                    fallt = t.time()
                    lastDown = t.time()
                    lastSideways = t.time()
                elif (evx.key == K_a):
                    moveLeft = False
                elif (evx.key == K_d):
                    moveRight = False
                elif (evx.key == K_s):
                    moveDown = False



            elif evx.type == KEYDOWN:
                if (evx.key == K_a) and validPos(newboard, activePiece, adjacentX=-1):
                    activePiece['pieceX'] -= 1
                    moveLeft = True
                    moveRight = False
                    lastSideways = t.time()
                elif (evx.key == K_d) and validPos(newboard, activePiece, adjacentX=1):
                    activePiece['pieceX'] += 1
                    moveRight = True
                    moveLeft = False
                    lastSideways = t.time()
                elif (evx.key == K_w):
                    activePiece['currentRotate'] = (activePiece['currentRotate'] + 1) %
len(pieceSet[activePiece['pieceShape']])
                    if not validPos(newboard, activePiece):
                        activePiece['currentRotate'] = (activePiece['currentRotate'] - 1) %
len(pieceSet[activePiece['pieceShape']])
                elif (evx.key == K_q):
                    activePiece['currentRotate'] = (activePiece['currentRotate'] - 1) %
```

```
len(pieceSet[activePiece['pieceShape']])
                if not validPos(newboard, activePiece):
                    activePiece['currentRotate'] = (activePiece['currentRotate'] + 1) %
len(pieceSet[activePiece['pieceShape']])
            elif (evx.key == K_s):
                moveDown = True
                if validPos(newboard, activePiece, adjacentY=1):
                    activePiece['pieceY'] += 1
                lastDown = t.time()
            elif evx.key == K_SPACE:
                moveDown = False
                moveRight = False
                moveLeft = False
                for x in range(1, bheight):
                    if not validPos(newboard, activePiece, adjacentY=i):
                        break
                    activePiece['pieceY'] += x - 1



        if (moveLeft or moveRight) and t.time() - lastSideways > sidewaysRate:
            if moveLeft and validPos(newboard, activePiece, adjacentX=-1):
                activePiece['pieceX'] -= 1
            elif moveRight and validPos(newboard, activePiece, adjacentX=1):
                activePiece['pieceX'] += 1
            lastSideways = t.time()



        if moveDown and t.time() - lastDown > downRate and validPos(newboard, activePiece,
adjacentY=1):
            activePiece['pieceY'] += 1
            lastDown = t.time()



        if t.time() - fallt > fallRate:
            if not validPos(newboard, activePiece, adjacentY=1):
                addPieceToBoard(newboard, activePiece)
                score += removeFinished(newboard)
                currentLevel, fallRate = getCurrentLvlAndFallRate(score)
                activePiece = None
            else:
                activePiece['pieceY'] += 1
                fallt = t.time()



        dispsurf.fill(bgCol)
        drawB(newboard)
        drawCurrentState(score, currentLevel)
        drawFollowingPiece(followingPiece)
        if activePiece != None:
            drawGivenPiece(activePiece)
```

```
        p.display.update()
        frameclock.tick(frames)


def createStringObjects(text, font, color):
    x = font.render(text, True, color)
    return x, x.get_rect()


def end():
    p.quit()
    s.exit()


def checkKey():
    checkQuit()


    for evx in p.event.get([KEYDOWN, KEYUP]):
        if evx.type == KEYDOWN:
            continue
        return evx.key
    return None


def drawTextLarge(text):
    titles, titler = createStringObjects(text, lfont, textShadowCol)
    titler.center = (int(width / 2), int(height / 2))
    dispsurf.blit(titles, titler)


    titles, titler = createStringObjects(text, lfont, textShadowCol)
    titler.center = (int(width / 2 - 3), int(height / 2 - 3))
    dispsurf.blit(titles, titler)


    pressS, pressR = createStringObjects('Press anything to play.', font, textCol)
    pressR.center = (int(width / 2), int(height / 2) + 100)
    dispsurf.blit(pressS, pressR)


    while checkKey() == None:
        p.display.update()
        frameclock.tick()
```

```
def checkQuit():
    for e in p.event.get(QUIT):
        end()
    for e in p.event.get(KEYUP):
        if e.key == K_ESCAPE:
            end()
        p.event.post(e)



def getCurrentLvlAndFallRate(score):
    currentLevel = int(score / 10) + 1
    fallRate = 0.26 - (currentLevel * 0.03)
    return currentLevel, fallRate



def getPiece():
    pieceShape = r.choice(list(pieceSet.keys()))
    piecex = {'pieceShape': pieceShape,
              'currentRotate': r.randint(0, len(pieceSet[pieceShape]) - 1),
              'pieceX': int(bwidth / 2) - int(twidth / 2),
              'pieceY': -2,
              'color': r.randint(0, len(colorset)-1)}
    return piecex



def addPieceToBoard(board, activePiece):
    for x in range(twidth):
        for y in range(theight):
            if pieceSet[activePiece['pieceShape']][activePiece['currentRotate']][y][x] != empty:
                board[x + activePiece['pieceX']][y + activePiece['pieceY']] = activePiece['color']



def returnEmptyBoard():
    board = []
    for i in range(bwidth):
        board.append([empty] * bheight)
    return board



def isOn(x, y):
```

```
    return x >= 0 and x < bwidth and y < bheight



def validPos(board, activePiece, adjacentX=0, adjacentY=0):
    for x in range(twidth):
        for y in range(theight):
            aboveBoard = y + activePiece['pieceY'] + adjacentY < 0
            if aboveBoard or pieceSet[activePiece['pieceShape']][activePiece['currentRotate']][y][x] ==
empty:
                continue
            if not isOn(x + activePiece['pieceX'] + adjacentX, y + activePiece['pieceY'] + adjacentY):
                return False
            if board[x + activePiece['pieceX'] + adjacentX][y + activePiece['pieceY'] + adjacentY] !=
empty:
                return False
    return True



def completeLine(b, y):
    for x in range(bwidth):
        if b[x][y] == empty:
            return False
    return True




def removeFinished(board):
    numy = bheight - 1
    numberRemoved = 0
    while numy >= 0:
        if completeLine(board, y):
            for downYax in range(y, 0, -1):
                for x in range(bwidth):
                    board[x][downYax] = board[x][downYax-1]
            for x in range(bwidth):
                board[x][0] = empty
            numberRemoved += 1
        else:
            numy -= 1
    return numberRemoved




def toPix(boxx, boxy):
    return (xmarg + (boxx * sizeofbox)), (topmarg + (boxy * sizeofbox))
```

```
def drawGivenBox(boxx, boxy, color, pixelx=None, pixely=None):
    if color == empty:
        return
    if pixelx == None and pixely == None:
        pixelx, pixely = toPix(boxx, boxy)
    p.draw.rect(dispsurf, colorset[color], (pixelx + 1, pixely + 1, sizeofbox - 1, sizeofbox - 1))


    p.draw.rect(dispsurf, lightcolorset[color], (pixelx + 1, pixely + 1, sizeofbox - 4, sizeofbox - 4))


def drawB(board):
    p.draw.rect(dispsurf, borderCol, (xmarg - 3, topmarg - 7, (bwidth * sizeofbox) + 8, (bheight *
sizeofbox) + 8), 5)


    p.draw.rect(dispsurf, bgCol, (xmarg, topmarg, sizeofbox * bwidth, sizeofbox * bheight))
    for posx in range(bwidth):
        for posy in range(bheight):
            drawGivenBox(posx, posy, board[posx][posy])


def drawCurrentState(score, currentLevel):
    scoreS = font.render('Score: %s' % score, True, textCol)
    scoreR = scoreS.get_rect()
    scoreR.topleft = (width - 300, 20)
    dispsurf.blit(scoreS, scoreR)
    currentLevelS = font.render('Level: %s' % currentLevel, True, textCol)
    currentLevelR = currentLevelS.get_rect()
    currentLevelR.topleft = (width - 300, 40)
    dispsurf.blit(currentLevelS, currentLevelR)


def drawGivenPiece(activePiece, pixX=None, pixY=None):
    pieceShape = pieceSet[activePiece['pieceShape']][activePiece['currentRotate']]
    if pixX == None and pixY == None:
        pixX, pixY = toPix(activePiece['pieceX'], activePiece['pieceY'])


    for sizex in range(twidth):
        for sizey in range(theight):
            if pieceShape[sizey][sizex] != empty:
                drawGivenBox(None, None, activePiece['color'], pixX + (x * sizeofbox), pixY + (y *
sizeofbox))
```

```
def drawFollowingPiece(activePiece):
    nextS = font.render('Next block:', True, textCol)
    nextR = nextS.get_rect()
    nextR.topleft = (width - 300, 75)
    dispsurf.blit(nextS, nextR)
    drawGivenPiece(activePiece, pixX=width-300, pixY=100)


if __name__ == '__main__':
    main()
```

Fun Console RPG

This one's actually a bit different. Instead of just showing you all the code and having you write it in and press play like an old programming manual for a computer running BASIC, we're going to take this step by step and work on some basic components of a console RPG. Specifically, we're going to work on a battle system and a very basic inventory. We're just going to be coding the shell of these things so that you can actually extend them later if you so choose. Think of it as a debugging experience.

So let's take this step by step and start making the frame for our code. Create a new file in your text editor of choice and call it consolerpg.py . You can call it whatever you like, but that's the name I'm going with for this example.

After that, we need to create some basic classes. Let's think about what we need.

First, we're going to need a player of course. Then, if we want a battle system, we're going to need a few enemies. Since these are both things with hit points and attacks, we can code this as a single thing called an *entity* and then have the others derive from there. As far as attacks go, we're going to have a relatively complicated system for those, but it should work itself out eventually.

The program flow will be that we randomly select an enemy to fight of the existing loaded enemies. We fight it until its dead, then another one spawns.

Let's go ahead and say this is turn-based, too; a turn-based battle system. Each turn will present you with an option to choose one of your spells, max out your hit points (HP), change your weapon from a list of weapons, list your inventory, or exit the game.

Right, so let's make a bullet list of things we need to implement, which we'll return to throughout this section:

- Entity Class
- Player Class
- Enemy Class

- Attack Class
- Item Class
- Weapon Class

We're going to comment out functions we need to add later as we go through our code because we're going to try to keep this all in one coherent thought pattern.

So the first thing we're going to do is create our entity, player, and enemy classes with the traits we'd expect such classes to have.

We're going to need the random and copy libraries for this, so we first import those:

```
import random, copy
```

Go into the file we created and create a new class called *Entity*. It should have a few innate properties such as health and name. Here's how mine looked:

```
class Entity(object):
        def __init__(self):
        self.health = 0
        self.name = ‘’
        self.attackList = []
```

There's not really any reason to set these because in Python, variables aren't inherited. However, it is generally solid programming practice in other object oriented languages and does provide a bit of a failsafe if something should go horribly wrong or be forgotten somewhere.

Anyway, with that done, we're going to need to create our player class. This is going to have health, of course. The initializer will take one argument: the player's name. We'll also have an equipped weapon, which I'm going to set with an imaginary weapon class which doesn't quite exist yet. It's going to take the arguments of a weapon's name and the weapon's modifier value. Then, I'm going to create an inventory, which will hold the weapons. After that, an attack list, with the player's attacks. You can get creative here if you'd like. I'm just using my own examples. Lastly, we need to create a variable called `isAlive` and make sure that it's true.

```
class Player(Entity):
   def __init__(self, name):
      self.health = 50
      self.name = name
      self.equippedWeapon = Weapon("Fist", 1.00)
      self.inventory = [Weapon("Fist", 1.00), Weapon("Dagger", 1.12), Weapon("Short sword", 1.45)]
      self.attackList = [Attack("Jab", 5, self.equippedWeapon.modifier), Attack("Uppercut", 8,
self.equippedWeapon.modifier), Attack("Kick", 7, self.equippedWeapon.modifier),
Attack("Roundhouse", 8, self.equippedWeapon.modifier)]
      self.isAlive = True
```

Note here that this code is inheriting from the entity class. This is super important, not necessarily here but in more ambitious code when you have an entity class, there are a lot of common functions you'd want to add to it, such as animation and bounding box collision which a great deal of in-game entities use, not just the players or non-playable characters (NPCs).

After that's done, we create the enemy class. This one's a lot more straightforward. We do need to set it up so that the programmer must declare the enemy's name, though.

Here's how mine looked:

```
class Enemy(Entity):
   def __init__(self, name, health):
      self.health = health
      self.name = name
      self.isAlive = True
```

But we also need to make another function within this class. This function will be called *getAttack*. It will only take an argument of self. It will return a random attack from the attackList.

```
        def getAttack(self):
        return random.choice(self.attackList)
```

Now we need to get to actually coding the infrastructure for our attacks. This is pretty straightforward too. We define the class, and then we create an initializer. This initializer accepts four arguments: self, name, damage, and modifier. The inherent damage of an attack is equal to the argument *damage* multiplied by the argument *modifier*. This will make more sense later.

```
class Attack(object):
        def __init__(self, name, damage, modifier):
        self.name = name
        self.damage = damage * modifier
```

Now we need to make the item and weapon classes. We don't actually *use* the item class in this example, but we do make extensive use of the weapon class. The item class for now simply takes a name, and the weapon class simply takes a name and modifier.

First, the item class:

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

Then, the weapon class:

```
class Weapon(Item):
        def __init__(self, name, modifier):
        self.name = name
        self.modifier = modifier
```

Perfect. Now we need to jump into some of the game logic. We're going to create our turn now. What this will do is take the player's attack, do it first, then do the enemy's attack if they're still alive. It then does some condition checking to see who's alive and respond accordingly. It takes an argument of the player's attack, the player object, and the enemy object.

```
def attackTurn(playerAttack, player, enemy):
        print "You used: %s" % playerAttack.name
        enemy.health -= playerAttack.damage
```

Now we check to see if the enemy is still alive after our attack. If so, they attack. If we die during this, we are set to be dead. Observe:

```
        if enemy.health > 0:
        enemyAttack = enemy.getAttack()
        print "Enemy %s used: %s" % (enemy.name, enemyAttack.name)
        player.health -= enemy.getAttack().damage
        if not player.health > 0:
        player.isAlive = False
```

And if the enemy isn't alive to attack anyway.

```
        else:
        enemy.isAlive = False
```

Simple. Now what we do is some conditional checking where we see who's alive and print out messages accordingly. If both are alive, we just continue the game loop we call this entire method from with no printed message.

```
    if player.isAlive and enemy.isAlive:
        return
    elif not player.isAlive and enemy.isAlive:
        print "You have died!"
        return
    elif not enemy.isAlive and player.isAlive:
        print "You killed the %s!" % enemy.name
        return
    else:
```

```
        print "A bizarre draw has occurred."
        return
```

Now we need to create a class to list and change weapons from the player inventory. This will simply take the argument of a player object.

```
def changeWeapons(p):
        for i in range(len(p.inventory)):
        print “%d: %s” % (i, p.inventory[i].name)
        i = raw_input(“Enter which one you’d like to set your weapon to: “)
        i = int(i)
        p.equippedWeapon = p.inventory[i]
```

Now with that function defined and declared, we need to get into the actual main bulk of our code. First we declare our player object and ask the player’s name. Since the player object takes a string argument of *name* anyhow, we can just do this straight from the initializer:

```
p = Player(raw_input(“What is your name?\t\t”)
```

Then we have to create an enemy list and define our enemies. Mine are silly:

```
enemyList = [Enemy("giant rat", 20), Enemy("land piranha", 30)]
```

Then we have to give these enemies attacks, like so:

```
enemyList[0].attackList = [Attack("Gnaw", 6, 1.00), Attack("Bite Down", 10, 1.00)]
enemyList[1].attackList = [Attack("Gnash", 20, 1.00), Attack("Fin Slap", 4, 1.00)]
```

After that’s done, we create a variable called *run* for our game loop, and a variable called *currentEnemy* used to store whichever enemy is currently active.

```
run = True
currentEnemy = None
```

Now, we start our game loop. At the beginning of every iteration, it checks if the current enemy is either a.) dead or b.) doesn’t exist at all. Either way, it creates a new enemy with a copy of an enemy object from our enemy list. You have to create a copy, otherwise it massively glitches out.

```
while run:
   if currentEnemy == None or if currentEnemy.isAlive == False:
      currentEnemy = copy.deepcopy(random.choice(enemyList))
```

Then we check to see the player is alive. If so, we run our normal loop. The first thing we do is print the player name and health, and the enemy name and health:

```
if p.isAlive == True:
    print "\n%s CURRENT HEALTH:\t%d" % (p.name, p.health)
    print "%s CURRENT HEALTH:\t%d\n\n" % (currentEnemy.name, currentEnemy.health)
```

After that, we print out our options. This is a little tedious to code, but it'll be worth it. Our options are:

- Choose *x* attack
- Restore health
- Change weapons
- Exit

Here's how mine ended up looking:

```
for i in range(len(p.attackList)):
    print "Enter %d to use the move: %s" % (i, p.attackList[i].name)

print "Enter %d to max your HP." % len(p.attackList)
print "Enter %d to change weapons." % (len(p.attackList) + 1)
print "Enter %d to exit." % (len(p.attackList) + 2)
```

After that, we get the player's input and parse it, acting accordingly.

```
if playerIn < len(p.attackList):
    attackTurn(p.attackList[playerIn], p, currentEnemy)
elif playerIn == len(p.attackList):
    p.health = 50
elif playerIn == (len(p.attackList) + 1):
    changeWeapons(p)
elif playerIn == (len(p.attackList) + 2):
    run = False
else:
    "That's not a valid input.\n\n\n"
```

Then, if the player isn't even alive after all, we just print "You died! Game over." and terminate the loop.

```
else:
    print "You died! Game over."
    run = False
```

By the end, your Python program in its entirety should look something like this:

CODE:

```
import random, copy
```

```
class Entity(object):
    def __init__(self):
        self.health = 0
        self.name = ""
        self.attackList = []

class Player(Entity):
    def __init__(self, name):
        self.health = 50
        self.name = name
        self.equippedWeapon = Weapon("Fist", 1.00)
        self.inventory = [Weapon("Fist", 1.00), Weapon("Dagger", 1.12), Weapon("Short sword", 1.45)]
        self.attackList = [Attack("Jab", 5, self.equippedWeapon.modifier), Attack("Uppercut", 8,
self.equippedWeapon.modifier), Attack("Kick", 7, self.equippedWeapon.modifier),
Attack("Roundhouse", 8, self.equippedWeapon.modifier)]
        self.isAlive = True

class Enemy(Entity):
    def __init__(self, name, health):
        self.health = health
        self.name = name
        self.isAlive = True

    def getAttack(self):
        return random.choice(self.attackList)

class Attack(object):
    def __init__(self, name, damage, modifier):
        self.name = name
        self.damage = damage * modifier

class Item(object):
    def __init__(self, name):
        self.name = name

class Weapon(Item):
    def __init__(self, name, modifier):
        self.name = name
        self.modifier = modifier

def attackTurn(playerAttack, player, enemy):
    print "You used: %s" % playerAttack.name
    enemy.health -= playerAttack.damage
    if enemy.health > 0:
        enemyAttack = enemy.getAttack()
        print "Enemy %s used: %s" % (enemy.name, enemyAttack.name)
        player.health -= enemy.getAttack().damage
        if not player.health > 0:
            player.isAlive = False
```

```
    else:
        enemy.isAlive = False

    if player.isAlive and enemy.isAlive:
        return
    elif not player.isAlive and enemy.isAlive:
        print "You have died!"
        return
    elif not enemy.isAlive and player.isAlive:
        print "You killed the %s!" % enemy.name
        return
    else:
        print "A bizarre draw has occurred."
        return

def changeWeapons(p):
    for i in range(len(p.inventory)):
        print "%d: %s" % (i, p.inventory[i].name)
    i = raw_input("Enter which one you'd like to set your weapon to: ")
    i = int(i)
    p.equippedWeapon = p.inventory[i]

p = Player(raw_input("What is your name?\t\t"))

# populate list of enemies
enemyList = [Enemy("giant rat", 20), Enemy("land pirahna", 30)]
enemyList[0].attackList = [Attack("Gnaw", 6, 1.00), Attack("Bite Down", 10, 1.00)]
enemyList[1].attackList = [Attack("Gnash", 20, 1.00), Attack("Fin Slap", 4, 1.00)]

run = True
currentEnemy = None

while run:
    if currentEnemy == None or if currentEnemy.isAlive == False:
        currentEnemy = copy.deepcopy(random.choice(enemyList))

    if p.isAlive == True:
        print "\n%s CURRENT HEALTH:\t%d" % (p.name, p.health)
        print "%s CURRENT HEALTH:\t%d\n\n" % (currentEnemy.name, currentEnemy.health)

        for i in range(len(p.attackList)):
            print "Enter %d to use the move: %s" % (i, p.attackList[i].name)

        print "Enter %d to max your HP." % len(p.attackList)
        print "Enter %d to change weapons." % (len(p.attackList) + 1)
        print "Enter %d to exit." % (len(p.attackList) + 2)

        playerIn = raw_input("What do you choose? ")
        playerIn = int(playerIn)
```

```
        if playerIn < len(p.attackList):
            attackTurn(p.attackList[playerIn], p, currentEnemy)
        elif playerIn == len(p.attackList):
            p.health = 50
        elif playerIn == (len(p.attackList) + 1):
            changeWeapons(p)
        elif playerIn == (len(p.attackList) + 2):
            run = False
        else:
            "That's not a valid input.\n\n\n"
    else:
        print "You died! Game over."
        run = False
```

There's quite a bit that you can add to this, but it's a very barebones solution and gives you the tools to expand upon it.

By now, we've covered quite a bit of information about programming games. I really enjoy programming games in my free time, and hopefully you'll share that passion. It's really rewarding to write something fun for other people to play. You can use the lessons about Python and PyGame that you've gotten in this chapter in order to form a jumping-off point for your own personal video game projects, should you have any.

Conclusion

Thank you for making it through to the end of *Python: Tips and Tricks to Programming Code with Python*. Let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever it may be.

The next step is to start applying these tips and tricks to your next programming endeavor. There's no perfect time like the present to start using what you've learned.

Finally, if you found this book useful in anyway, a review on Amazon is always appreciated!

Good luck!

About the Author

Charlie Masterson is a computer programmer and instructor who have developed several applications and computer programs.

As a computer science student, he got interested in programming early but got frustrated learning the highly complex subject matter.

Charlie wanted a teaching method that he could easily learn from and develop his programming skills. He soon discovered a teaching series that made him learn faster and better.

Applying the same approach, Charlie successfully learned different programming languages and is now teaching the subject matter through writing books.

With the books that he writes on computer programming, he hopes to provide great value and help readers interested to learn computer-related topics.