An Introduction To

python

by Mark Clarkson

Version 2.1, July 2012

Model Answers

Introduction

The programs below are sample solutions to the challenges and tasks provided in the main text book. These solutions work and demonstrate how to use the techniques covered in the book, however the nature of programming in particular and problem solving in general means that there are many possible strategies and solutions to some of the problems.

A program that works differently, but meets the requirements of the task is still a successful solution, and it is quite possible that your solution is better than that listed here.

One suggestion is to try to solve each problem yourself and then compare the methods and techniques used below with your own. If you see a good idea then consider using it, but please don't assume that the method used here is better than yours as it may not be.

At the time of writing, only tasks up to and including section 6.1 have been completed. Problems from sections 6.2 and 6.3 will be solved and model answers provided at some point in the future. Keep an eye on the Mukoku site (Google it) for an updated version.

--

Mark Clarkson

July, 2012

3.1 Selection

3.1e - Selection Challenge

Try making a calculator.

The program will ask for two numbers and then give you a menu, so you can add, subtract, multiply, divide, square either number or find out the power.

```
"""
Filename: calculator.py
Author: Mark Clarkson
Date Created: July, 2012
Notes: \n means 'start a new line', \ means \ 'continue string on next line'
"""
menu = "Welcome to the Python Calculator\n\ #Store menu text in a string
What calculation would you like to perform?\n\
1. Add\n\
2. Subtract \n\
3. Multiply \n\
4. Divide \n\
5. Square first number \n\
6. Square second number \n\
7. Calculate the power\n"

num1 = int(input("Enter the first number: ")) #important to cast as int
num2 = int(input("Enter the second number: "))
print(menu)
choice = int(input("Enter your choice: ")) #get user choice

if choice == 1:
    print(num1 + num2)
elif choice == 2:
    print(num1 - num2)
elif choice == 3:
    print(num1 * num2)
elif choice == 4:
    print(num1 / num2)
elif choice == 5:
    print(num1**2)
elif choice == 6:
    print(num2**2)
elif choice == 7:
    print(num1**num2)
```

3.2 Iteration

3.2c - FOR Loop Challenge

1. Look closely at the three times table example on the last page. Write a similar program to calculate the four times table.
2. Write a program that will calculate the 5 times table.
3. Try writing a program that will prompt for an integer (whole number) and print the correct times table (up to 12x). Test with all of the tables up to 12.

```
"""
Filename: forLoop.py
Author: Mark Clarkson
Date Created: July, 2012
Notes: FOR loop format: (start, stop, step)
"""

#4x table
for loopCounter in range(4,49,4):
    print(loopCounter)

#5x table
for loopCounter in range(5,61,5):
    print(loopCounter)

#n x table
table = int(input("Which times table do you want? "))
for loopCounter in range(table, 12 * table + 1, table):
    print(loopCounter)
```

3.2f - Loop Challenge

A factorial means the number and all of the preceding numbers multiplied together.

So 4 factorial (written 4!) = 1 x 2 x 3 x 4

5! = 1 x 2 x 3 x 4 x 5

And so on...

Write a program that will prompt for an integer and then calculate the factorial of that number.

To test it works, 8! > 40,320

```
"""
Filename: factorial.py
Author: Mark Clarkson
Date Created: July, 2012
Notes: FOR loop format: (start, stop, step) OR (start, stop) [assume step = 1]
"""

fact = int(input("Enter your factorial number: ")) #fact short for factorial
runningTotal = 1

for loopCounter in range (1,fact+1):
    runningTotal = runningTotal * loopCounter

print(fact,"! =",runningTotal)
```

4.1 Turtle

4.1c - Creating shapes challenge

A circle has 360°. For each shape the turtle has to do one full circuit (i.e. turn 360°).

A square has 4 turns, and 360° ÷ 4 = 90°.

A hexagon has 6 turns, and 360° ÷ 4 = 60°.

A triangle has 3 turns, and 360°÷ 3 = ???

Try drawing a triangle. And a pentagon (5 sides). And an octagon (8 sides). And a decagon (10 sides).

```
"""
Filename: turtleShapes.py
Author: Mark Clarkson
Date Created: July, 2012
Notes: 360 / noOfSides = angleInDegrees
"""

import turtle

window = turtle.Screen()
timmy = turtle.Turtle()

#pentagon
for loopCounter in range(5):
    timmy.forward(50) #small steps so there is still room
    timmy.right(72)

#octagon
for loopCounter in range(8):
    timmy.forward(60) #bigger steps so it is easier to see
    timmy.right(45)

#decagon
for loopCounter in range(10):
    timmy.forward(70) #bigger steps again
    timmy.right(36)

window.exitonclick()
```

5.1 Regular Expressions

5.1b - Regular Expression challenge

Create a program that asks the user to type in a URL (website address) and validates it. A valid URL should have "some characters, a full stop, some characters, a full stop and some characters".

```
"""
Filename: regularExpressions.py
Author: Mark Clarkson
Date Created: July, 2012
Notes: '.' (full stop) means any character. '\.' means a full stop
"""

import re

pattern = "^.+\..+\..+$"

url = input("Enter a web address: ")

if re.search(pattern,url):
    print("Address is valid")
else:
    print("Invalid address")
```

5.3 File Handling

5.3d - Reading Challenge

Write a program that will read each character and append it to a string until the first space is found, then print that string.

```
 """
Filename: findASpace.py
Author: Mark Clarkson
Date Created: July, 2012
Notes: This program creates, as well as reading the file
"""

#create a file
file = open("temp","w")
string = "He's pining for the fjords"
file.write(string)
file.close #remember to close the file

newString = "" #holder for the first word

#read the file
file = open("temp","r")
char = file.read(1) #read the first character
while char != " ": #repeat until you get to a space
    newString = newString + char #add the character to the newString
    char = file.read(1)
file.close #once the loop has finished, remember to close the file!

print("The first word is:",newString)
```

6.1 Straightforward Exercises

6.1a - Pythagoras' Calculator

Write a program that will do the following:

- Print a menu:

  Pythagoras' Calculator

  1 - Find a hypotenuse

  2 - Find another side

  9 - Exit

  Enter an option:

- If '1' is entered, prompt for the two other sides, calculate the hypotenuse and print the answer. Reprint the menu.
- If '2' is entered, prompt for the hypotenuse and the other side, calculate the third side and print the answer. Reprint the menu.
- If '9' is entered, print a goodbye message and exit (`break`)
- If another value is entered, print an error message and print the menu again.

NB: Remember you will need to import the math module (`import math`) and use the `sqrt()` function.

```
"""
Program Name: pythag.py
Author: Mark Clarkson
Date Created: July, 2012
Description: Small program to solve pythagoras problems
"""

import math #needed for the sqrt (square root) function

#procedure to display the menu
def printMenu():
    menu = "\n====================\n\
Pythagoras' Calculator\n\
1 - Find a hypotenuse\n\
2 - Find another side\n\
9 - Exit"
    print(menu)

#procedure to display a goodbye message
def sayGoodbye():
    print("\n\nThank you for using Pythagoras' Calculator")

#procedure to display an error message
def sayError():
    print("Sorry, that repsonse was not recognised. Please try again.")
```

```
#procedure to calculate and display the hypotenuse
def findHyp():
    A = int(input("Enter the length of a short side: "))
    B = int(input("Enter the length of the other short side: "))
    C = math.sqrt(A**2 + B**2) #A squared + B squared = C squared
    print("\n\nThe hypotenuse is", C)

#procedure to calculate and display a missing short side
def findOther():
    C = int(input("Enter the length of the hypotenuse: "))
    B = int(input("Enter the length of the short side you know: "))
    A = math.sqrt(C**2 - B**2) #C squared - B squared = A squared
    print("\n\nThe other short side is", A)

#MAIN METHOD
option = 0 #set option to something, otherwise the while statement will fail
while option != 9:
    printMenu()
    option = int(input("Enter an option: "))
    if option == 1:
        findHyp()
    elif option == 2:
        findOther()
    elif option == 9:
        sayGoodbye()
    else:
        sayError()
"""
Alternative loop:
while True: #keep looping forever [break statement needed to jump out]
    printMenu()
    option = int(input("Enter an option: "))
    if option == 1:
        findHyp()
    elif option == 2:
        findOther()
    elif option == 9:
        sayGoodbye()
        break
    else:
        sayError()
"""
```

6.1b - Primary Division

Write a program that will do the following:

- Prompt the user for two numbers.
- Calculate, and print the result of the division in the format x remainder y (e.g. 17 / 4 = 4 remainder 1).
- Ask the user if they would like to run the program again or quit.

```
"""
Program Name: divAndMod.py
Author: Mark Clarkson
Date Created: July, 2012
Description: Small program to divide using div and mod
"""

num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

div = int(num1 / num2) #get the whole number part of the division [div]
mod = num1 - (num2*div) #work out the remainder [mod]

print(num1,"/",num2,"=",div,"r",mod)

"""
The following solution is quicker, but hides how to get just the integer part

div = num1 // num2
mod = num1 % num2

This solution is even quicker, but hides even more of the workings

print(divmod(num1,num2))
"""
```

6.1c - Random Arithmetic

Write a program that will do the following:

- Generate two random numbers
- Calculate and print the results if you add, subtract, divide and multiply the two numbers verbosely (e.g. "2 + 3 = 5")
- Ask the user if they would like to run the program again or quit.

```
"""
Program Name: random.py
Author: Mark Clarkson
Date Created: July, 2012
Description: Small program to generate and use random numbers
"""

import random #import the random module

#function to generate and return a random integer from 1 to 100
def getRandomNumber():
    return(random.randint(1,100))

#procedure to get two random numbers, perform calculations and print results
def doAllTheWork():
    rand1 = getRandomNumber()
    rand2 = getRandomNumber()

    print(rand1,"+",rand2,"=",rand1+rand2)
    print(rand1,"-",rand2,"=",rand1-rand2)
    print(rand1,"x",rand2,"=",rand1*rand2)
    print(rand1,"/",rand2,"=",rand1/rand2)

#MAIN METHOD
#Loop forever. Only break if capital Q is entered
while True:
    doAllTheWork()
    choice = input("Type Q to quit or press the return key to repeat. ")
    if choice == "Q":
        break
```

6.1d - Code Maker (and Breaker) - Lite Version

Write a program that will do the following:

- Print a menu:

  Code Maker (and Breaker)

  1 - Encode a letter

  2 - Decode a letter

  9 - Exit

  Enter an option:

- If '1' is entered, prompt for a letter and use a selection statement to convert the letter into an integer (a = 1, b = 2, etc.).
- If '2' is entered, prompt for an integer, check it is valid (prompt again if invalid) and convert the integer into a letter.
- If '9' is entered, print a goodbye message and exit (`break`)
- If another value is entered, print an error message and print the menu again.

My solution is quite long (128 lines, the next 4 pages). As well as the next few pages of code, there is also a downloadable solution available at the mukoku resource site (search "mukoku").

It is worth noting the number of sub routines used throughout the program to make the control flow easier to follow.

```
"""
Filename: CodeLite.py
Author: Mark Clarkson
Date: July, 2012
Description: A simple encryption program
NB: There are two versions of each conversion procedure
"""

# print a menu and get a user's choice. Some basic validation inside a loop
def menu():
    menu = "Code Maker (and Breaker)\n\
1 - Encode a letter\n\
2 - Decode a letter\n\
9 - Exit\n"

    while True:
        option = int(input(menu))
        if option == 1 or option == 2 or option == 9:
            return option
        print("Error, response not recognised\n\n")
```

```
"""
These two procedures use simple logic, but a lot of code
"""

# Simple logic but a long-winded solution
def convertCharIntSimple():
    char = input("Enter a single letter to convert: ")
    num = 0
    if char == "a" or char == "A":
        num = 1
    elif char == "b" or char == "B":
        num = 2
    elif char == "c" or char == "C":
        num = 3
    elif char == "d" or char == "D":
        num = 4
    elif char == "e" or char == "E":
        num = 5
    elif char == "f" or char == "F":
        num = 6
    elif char == "g" or char == "G":
        num = 7
    elif char == "h" or char == "H":
        num = 8
    elif char == "i" or char == "I":
        num = 9
    elif char == "j" or char == "J":
        num = 10
    elif char == "k" or char == "K":
        num = 11
# repeat up to z

    print(num)
```

```
# Simple logic but  alogn winded solution
def convertIntCharSimple():
    num = 0
    while num < 1 or num > 26:
        num = int(input("Enter the code number: "))
        char = ""
        if num == 1:
            char = "a"
        elif num == 2:
            char = "b"
        elif num == 3:
            char = "c"
        elif num == 4:
            char = "d"
        elif num == 5:
            char = "e"
        elif num == 6:
            char = "f"
        elif num == 7:
            char = "g"
        elif num == 8:
            char = "h"
        elif num == 9:
            char = "i"
        elif num == 10:
            char = "j"
        elif num == 11:
            char = "k"
# repeat up to 26

        print(char)
```

```
"""
These two procedures use MUCH less code, but slightly more complex logic
A basic understanding of ASCII is required
"""


# Less code, but a slightly more complex logic
def convertCharIntASCII():
    char = input("Enter a single letter to convert: ")
    num = ord(char) #gives the ASCII value of that character
    if num < 97: # 97 is the ASCII code for 'a', anything lower is likely to be
upper case
        num = num + 32 # 65 is the ASCII code for 'A'. Adding 32 gives the ASCII
code for 'a'


    num = num - 96 # numbering now starts from 1


    print(num)


# Less code, but a slightly more complex logic
def convertIntCharASCII():
    num = 0
    while num < 1 or num > 26:
        num = int(input("Enter the code number: "))
        numASCII = num + 96 # convert to ASCII number
        char = chr(numASCII) #convert from ASCII number to character
        print(char)




while True:
    choice = menu()
    if choice == 1:
        result = convertCharIntSimple()
        #result = convertCharIntASCII()
    elif choice == 2:
        result = convertIntCharSimple()
        #result = convertIntCharASCII()
    else:
        print("Goodbye!")
        break


#END OF PROGRAM
```

6.1e - Time Calculator

Write a program that will do the following:

- Print a menu:

  Time Calculator - Arithmetic Mode

  1 - Add 2 times

  2 - Find the difference between 2 times

  8 - Conversion mode

  9 - Exit

  Enter an option:

- If '8' is entered, print a menu:

  Time Calculator - Conversion Mode

  1 - Convert Time to Days

  2 - Convert Time to Hours

  3 - Convert Time to Minutes

  4 - Convert Minutes to Time

  5 - Convert Hours to Time

  6 - Convert Days to Time

  8 - Arightmetic mode

  9 - Exit

  Enter an option:

- If '8' is entered, return to the first menu.
- Complete all of the required procedures.
- Times should be stored as strings in the format DD:HH:MM.
- Days, Hours and Minutes should be stored as integers.

My solution is quite long (225 lines, the next 6 full pages). As well as the next few pages of code, there is also a downloadable solution available at the mukoku resource site (search "mukoku").

It is worth noting the number of sub routines used throughout the program to make the control flow easier to follow.

```
"""
Filename: time.py
Author: Mark Clarkson
Date: July, 2012
Description: A time calculator and converter
"""
#function to print the arithmetic menu and return the choice
def menuOne():
    menu = "\n\n====================\n\
Time Calculator - Arithmetic Mode\n\
1 - Add 2 times\n\
2 - Find the difference between 2 times\n\
8 - Conversion mode\n\
9 - Exit\n"

    while True: #repeat until a valid answer submitted
        print(menu)
        choice = int(input("Enter an option: "))
        if choice == 1 or choice == 2 or choice == 8 or choice == 9:
            return choice
        print("\n\nSorry, your response weas not recognised. Please try again.")

#function to print the conversion menu and return the choice
def menuTwo():
    menu = "\n\n====================\n\
Time Calculator - Conversion Mode\n\
1 - Convert Time to Days\n\
2 - Convert Time to Hours\n\
3 - Convert Time to Minutes\n\
4 - Convert Minutes to Time\n\
5 - Convert Hours to Time\n\
6 - Convert Days to Time\n\
8 - Arithmetic mode\n\
9 - Exit\n"
    while True: #repeat until a valid answer submitted
        print(menu)
        choice = int(input("Enter an option: "))
        if choice > 0 and choice < 7:
            return choice
        elif choice == 8 or choice == 9:
            return choice
        print("\n\nSorry, your response weas not recognised. Please try again.")
```

```
#function to get a time and return it in the format DD:HH:MM. Really this should
have validation
def getTime():
    days = input("Please enter the number of days (leave blank for none) ")
    if days == "":
        days = 0
    else:
        days = int(days)
    hours = input("Please enter the number of hours (leave blank for none) ")
    if hours == "":
        hours = 0
    else:
        hours = int(hours)
    mins = input("Please enter the number of minutes (leave blank for none) ")
    if mins == "":
        mins = 0
    else:
        mins = int(mins)
    time = [days,hours,mins]
    return time


#function to add two times together in DD:HH:MM
def addTwoTimes():
    days = [0]*3 #declare a blank array with 3 values to store the answer
    print("Enter your first time.")
    time1 = getTime()
    print("Enter your second time.")
    time2 = getTime()


    days[2] = time1[2] + time2[2] #add the minutes together
    if days[2] > 60:    #if there are more then 60 minutes then carry over to
hours
        days[1] = days[2] // 60
        days[2] = days[2] % 60


    days[1] = days[1] + time1[1] + time2[1] #add the hours together (including
the carry)
    if days[1] > 24: #if there are more than 24 hours then carry over to days
        days[0] = days[1] // 24
        days[1] = days[1] % 24


    days[0] = days[0] + time1[0] + time2[0] #add the days together (including
the carry)
```

```
    print(time1[0],"days",time1[1],"hours and",time1[2],"minutes\n   plus")
    print(time2[0],"days",time2[1],"hours and",time2[2],"minutes\n   equals")
    print(days[0],"days",days[1],"hours and",days[2],"minutes\n")


#function to subtract two times in DD:HH:MM
def diffTwoTimes():
    neg = False #boolean to state whether the answer is negative (second number
bigger than the first
    days = [0]*3 #declare a blank array with 3 values to store the answer
    print("Enter your first time.")
    time1 = getTime()
    print("Enter your second time.")
    time2 = getTime()


    min1 = time1[0] + 24 * (time1[1] + 60 * time1[2]) #calculate total minutes
in time 1
    min2 = time2[0] + 24 * (time2[1] + 60 * time2[2]) #caluclate total minutes
in time 2
    if min2 > min1: #find out it second number is bigger. If so, swap over so
result will always be positive
        time3 = [0]*3
        neg = True #record whether the subtraction is back to front
        for loopCounter in range(3):
            time3[loopCounter] = time2[loopCounter]
        for loopCounter in range(3):
            time2[loopCounter] = time1[loopCounter]
        for loopCounter in range(3):
            time1[loopCounter] = time3[loopCounter]



    days[2] = time1[2] - time2[2] #subtract minutes
    while days[2] < 0: #if result is negative, borrow (possibly more than once)
        days[1] = days[1] - 1
        days[2] = days[2] + 60


    days[1] = days[1] + time1[1] - time2[1] #subtract hours
    while days[1] < 0: #if result is negative, borrow (possibly more than once)
        days[0] = days[0] - 1
        days[1] = days[1] + 24


    days[0] = days[0] + time1[0] - time2[0] #subtract days. Should never be
positive
```

```
    print(time1[0],"days",time1[1],"hours and",time1[2],"minutes\n   plus")
    print(time2[0],"days",time2[1],"hours and",time2[2],"minutes\n   equals")
    if neg: #if values were swapped, make it clear that the result is negative
        print("minus",days[0],"days",days[1],"hours and",days[2],"minutes\n")
    else:
        print(days[0],"days",days[1],"hours and",days[2],"minutes\n")


#procedure to convert from a time (DD:HH:MM) to a decimal fraction of days
def convertTimeDay():
    print("Enter your time.")
    time = getTime()
    days = time[0] + (time[1] + time[2]/60)/24
    print("\n\n",time[0],"days",time[1],"hours and",time[2],"minutes
=",days,"days.")


#procedure to convert from a time (DD:HH:MM) to a decimal fraction of hours
def convertTimeHour():
    print("Enter your time.")
    time = getTime()
    hours = time[0] *24 + time[1] + time[2]/60
    print("\n\n",time[0],"days",time[1],"hours and",time[2],"minutes
=",hours,"hours.")


#procedure to convert from a time (DD:HH:MM) to a decimal fraction of minutes
def convertTimeMin():
    print("Enter your time.")
    time = getTime()
    mins = (time[0] *24 + time[1]) * 60 + time[2]
    print("\n\n",time[0],"days",time[1],"hours and",time[2],"minutes
=",mins,"minutes.")


#procedure to convert from a decimal fraction of minutes to a time (DD:HH:MM)
def convertMinTime():
    minsOrig = int(input("Enter a number of minutes.")) #record original input
    mins = minsOrig #use a different variable for calculations
    time = [0]*3 #declre a blank array of 3 elements to store the answer
    time[0] = mins // (24*60) #calculate number of days
    mins = mins % (24*60) #update number of minutes remaining
    time[1] = mins // 60 #calculate number of hours
    mins = mins % 60 #update number of minutes remianing
    time[2] = mins #add number of minutes remaining to answer
    print("\n\n",minsOrig,"minutes equals",
time[0],"days",time[1],"hours",time[2],"minuntes.")
```

```
#procedure to convert from a decimal fraction of hours to a time (DD:HH:MM)
#basic method as convertMinTime() above
def convertHourTime():
    hoursOrig = float(input("Enter a number of hours."))
    hours = hoursOrig
    time = [0]*3
    time[0] = hours // 24
    hours = hours % 24
    time[1] = int(hours)
    time[2] = (hours - int(hours))*60
    print("\n\n",hoursOrig,"hours equals",
time[0],"days",time[1],"hours",time[2],"minuntes.")


#procedure to convert from a decimal fraction of days to a time (DD:HH:MM)
#basic method as convertMinTime() above
def convertDayTime():
    daysOrig = float(input("Enter a number of days."))
    days = daysOrig
    time = [0]*3
    time[0] = int(days)
    days = days - int(days)
    days = 24*days #now really hours
    time[1] = int(days)
    days = days - int(days)
    days = days * 60 #now really minutes
    time[2] = days
    print("\n\n",daysOrig,"days equals",
time[0],"days",time[1],"hours",time[2],"minuntes.")
```

```
#MAIN METHOD
mode = 1
while True: #repeat forever (until break)
    if mode == 1: #if in calculation mode
        option = menuOne()
        if option == 1:
            addTwoTimes()
        elif option == 2:
            diffTwoTimes()
        elif option == 8:
           mode = 2 #switch mode
        elif option == 9:
            break
    elif mode == 2: #if in conversion mode
        option = menuTwo()
        if option == 1:
             convertTimeDay()
        elif option == 2:
             convertTimeHour()
        elif option == 3:
            convertTimeMin()
        elif option == 4:
            convertMinTime()
        elif option == 5:
            convertHourTime()
        elif option == 6:
            convertDayTime()
        elif option == 8:
           mode = 1 #switch mode
        elif option == 9:
            break

print("\n\nThank you for using the Time Calculator") #print goodbye message once
loop completed



#END OF PROGRAM
```

6.1f - ATM - Lite Version

Northern Frock needs you to write a program for their new ATMs (or Automatic Teller Machines). Assuming the starting balance is £67.14, Write a program that will do the following:

- Print a menu:

  Welcome to Northern Frock

  1 - Display balance

  2 - Withdraw funds

  3 - Deposit funds

  9 - Return card

  Enter an option:

- If '1' is entered, display the current balance and also the maximum amount available for withdrawal (must be a multiple of £10)
- If '2' is entered, provide another menu with withdrawal amounts of £10, £20, £40, £60, £80, £100, Other amount, Return to main menu & Return card.
- Check that the requested withdrawal is allowed, print a message to show that the money has been withdrawn and calculate the new balance.
- If 'Other amount' is selected then prompt the user for an integer value. Check this number is a multiple of 10 and that the withdrawal is permitted (as above).
- If '3' is entered, provide another menu that will allow the user to enter an amount to deposit(does not need to be a multiple of £10), return to main menu or return card. If funds are deposited, provide appropriate feedback and update the balance.
- If '9' is entered, print a goodbye message and exit (`break`)
- If another value is entered, print an error message and print the menu again.

Again, the code for this program is quite long (168 lines). Again, a python file is available rather than copying and pasting. Please note the comments on local and global variables in particular.

```
"""
Filename: ATMLite.py
Author: Mark Clarkson
Date: July, 2012
Description: A simple ATM simulator
"""

#simple procedure to force-quit the program from any point without
#following returns back through the other functions
def returnCard():
    print("Returning card now, thank you for your custom")
    global keepGoing #see note at bottom of program on global variables
    keepGoing = False
```

```
#simple procedure to display the current and available balance
def displayBalance(balance):
    #cleanup balance for stray 1/10ths of pennies, etc.
    balance = int(balance * 100)
    balance = balance / 100
    maxAvail = (balance // 10) * 10
    print("Current balance, £",balance)
    print("Maximum withdrawal, £",maxAvail)


#subtract a set amount form the balance (checking money available)
#prompt for an 'other' figure, if needed & check it is a multiple of £10
def subtract(balance,amount = 1): #if amount is not passed, set it to 1

    while amount % 10 != 0: #while amount not a multiple of 10
        amount = int(input("Enter an amount to withdraw (multiple of £10): "))
        if amount % 10 != 0:
            print("Error, invalid amount. Try again or enter '0' to return")
            # if user enters 0 then mod operation is satisfied and code below
will run,
            # making no change to the balance

    #Check if there is enough in the account
    if balance > amount:
        balance = balance - amount
        print("£",amount,"withdrawn. Please collect your money")
    else:
        print("Sorry, insufficient funds available")

    return balance
```

```
#display withdrawal menu and handle choice
def withdraw(balance):
    menu = "Enter amount to withdraw.\n\
1 - £10\n\
2 - £20\n\
3 - £40\n\
4 - £60\n\
5 - £80\n\
6 - £100\n\
7 - Other Amount\n\
8 - Return to Main Menu\n\
9 - Return Card\n"

    option = 0

    while option < 1 or option > 9:
        option = int(input(menu))
        if option == 1:
            balance = subtract(balance,10)
            return balance
        elif option == 2:
            balance = subtract(balance,20)
            return balance
        elif option == 3:
            balance = subtract(balance,40)
            return balance
        elif option == 4:
            balance = subtract(balance,60)
            return balance
        elif option == 5:
            balance = subtract(balance,80)
            return balance
        elif option == 6:
            balance = subtract(balance,100)
            return balance
        elif option == 7:
            balance = subtract(balance)
            return balance
        elif option == 8:
            break
        elif option == 9:
            returnCard()
            return balance
```

```
#Handle deposits by asking for an amount.
#0 amount will have no effect (will look like returning to main menu)
#999 will quit (you can't deposit £999, sorry
#Note loack of validation. Negative numbers will work
def deposit(balance):
    menu = "Enter the deposit amount. \n\
Enter 0 to return to main menu.\n\
Enter 999 to return card\n"

    choice = float(input(menu))

    if choice == 999:
        returnCard()
    else:
       balance = balance + choice

    return balance


#END OF PROGRAM
```

```
#Display main menu, get choice, run appropriate function
def mainMenu(balance):
    menu = "\n\n\n==============================\n\
Welcome to Northern Frock\n\
1 - Display Balance\n\
2 - Withdraw Funds\n\
3 - Deposit Funds\n\
9 - Return Card\n\
Enter an option: "

    invalidChoice = True #Assume invalid and repeat until valid choice

    while invalidChoice:
        choice = int(input(menu))
        if choice == 1:
            invalidChoice = False
            displayBalance(balance)
        elif choice == 2:
            invalidChoice = False
            balance = withdraw(balance)
        elif choice == 3:
            invalidChoice = False
            balance = deposit(balance)
        elif choice == 9:
            returnCard()
        else:
            print("\n\nInvalid choice, please try again.")

        return balance


#Main Method
balance = 67.14
keepGoing = True
while keepGoing:
    balance = mainMenu(balance)
```

```
"""
Some notes on global variables:

You will notice that the balance vairable is constantly being passed to
functions
and returned back. This means that balance is always a local variable (i.e. it
exists within that function). This also means that any changes don't affect the
rest of the program until the variable is returned (so we can try doing things
in a non-destructive manner).

You will also notice that the keepGoing variable is not passed, but in the
returnCard() method there are two lines referring to it. The first declares
keepGoing as a global variable (otherwise Python will assume it is a local
variable
and make a new one that only exists within that method). The second assigns a
new
value so that the global variable will be updated immediately without the need
to
pass th evariable around from one method to another.

Global variables are generally frowned upon for many reasons, but in this case
it is
simpler to refer to keepGoing without having to pass it to methods that will
never
need to use it. This isn;t the case with the balance variable which is used by
several
of the functions in the code above.
"""
```

6.2 Advanced Exercises

6.2a - Prime Candidate

Prime numbers are numbers that are only divisible by 1 and itself. The first few prime numbers are 2, 3, 5, 7, 11 & 13.

The easiest way to test for primes is to try dividing it by every number up to the square root of that number.

e.g. tests for 27 and 31. Divide by every number up to the square root of that number (5.196 & 5.57)

27 / 2 = 13 r 1	31 / 2 = 15 r 1
27 / 3 = 9 r 0 (we could stop now, but we won't)	31 / 3 = 10 r 1
27 / 4 = 6 r 3	31 / 4 = 7 r 3
27 / 5 = 5 r 2	31 / 5 = 6 r 1
No point going past 5.196	No point going past 5.57
3 & 9 are factors, so NOT prime	No factors found, so IS prime

The easiest way to do this is to use the % operator (27 % 2 = 1 & 27 % 3 = 0)

Write a program that will do the following:

- Prompt the user for a lower and upper number (limited to a maximum range of 200 and going no higher than 10,000).
- Print all of the prime numbers in that range.
- Ask the user if they would like to run the program again or quit.

6.2b - Binary Search

A binary search is the most efficient way to find a value. It involves splitting the available values into two equal halves and testing which half it is in, and then refining this until the correct value is found.

e.g. A number must be between 1 and 100, in this case it is 53.

Midpoint between 1 and 100 is 50 `[int((1+100)/2)]`. Target number is higher.

Midpoint between 50 and 100 is 75. Target number is lower.

Midpoint between 50 and 75 is 62. Target number is lower.

Midpoint between 50 and 62 is 56. Target number is lower.

Midpoint between 50 and 56 is 53. Target number is found.

Write a program that will do the following:

- Prompt the user for an integer between 1 and 100 (validating and prompting if invalid)
- Use a selection statement to test the value 50 (below, equal, above) and print this guess.
- Repeat the above until the correct number is found.
- Ask the user if they would like to run the program again or quit.

Extension 1: Count and output the number of steps required to find the number.

Extension 2: Generate a random number as the target.

Extension 3: Print all of the values that require exactly 7 steps to find them.

Extension 4: Find the mean average number of steps required to find every number frpm 1 to 100.

Extension 5: Calculate the maximum and mean number of steps required if the range of numbers is 200, 1,000, 2,000, 10,000 & 100,000. Are you surprised by the results?

6.2c - Advanced Strings

You can use the following functions to do interesting things with strings:

```
string = "Strings are sequences of letters"
x = len(string)     #Returns the length of the string                    x = 32
x = string[0]       #Returns the 1st character of the string             x = S
x = string[3]       #Returns the 4th character of the string             x = i
x = string[3:5]     #Returns the 4th and 5th characters of the string     x = in
x = string[3:6]     #Returns the 4th - 6th characters of the string       x = ing
x = str.lower(string[0])        #Returns a lower case string             x = s
x = str.upper(string[3])        #Returns an upper case string            x = I


x = str.islower(string[3])
#Returns a bool depending on whether the character is lower case      x = True

x = str.isupper(string[0])
#Returns a bool depending on whether the character is upper case      x = True

x = str.isdigit(string[5])
#Returns a bool depending on whether the character is a digit         x = False

x = str.isspace(string[7])
#Returns a bool depending on whether the character is a space         x = True
```

Write a program that will do the following:

- Prompt the user for a string
- Calculate and print:
  - The number of characters (with spaces)
  - The number of characters (without spaces)
  - The number of upper case characters
  - The number of lower case characters
  - An upper case version of the string
  - A lower case version of the string
  - Whether the string is a palindrome (the same backwards as forwards)
  - Print the string in reverse

6.2d - Code Maker (and Breaker) - Full Version

Remind yourself of the lite version of this program (4.1d). This program encoded (or decoded) one character at a time.

Write a program that will do the following:

- Print a menu:

  Code Maker (and Breaker)

  1 - Encode a sentence

  2 - Decode a sentence

  9 - Exit

  Enter an option:

- If '1' is entered, prompt for a string and convert the string into a series of string of digits (a = 1, b = 2, etc.) separated by 1 space per character (space = 27). Print this string.
- If '2' is entered, prompt for a string of numbers separated by 1 space per character and convert into a string. Print this string.
- If '9' is entered, print a goodbye message and exit (`break`)
- If another value is entered, print an error message and print the menu again.

6.2e - Bubble Sort

A bubble sort is a simple (but not particularly efficient) method of putting a list in order.

A bubble sort looks at the first two elements in a list (or array) to see if the first is bigger than the second. If it is, they get swapped round. Either way, the 2nd and 3rd are checked, then the 3rd and 4th, etc...

At the end, the whole thing is done again.

Only when a complete run through happens without any swaps is the list correct.

e.g.

x = [3,9,5,2]

Compare x[0] and x[1], 3 < 9 so no swap

Compare x[1] and x[2], 9 !< 5 so swap [3,5,9,2]

Compare x[2] and x[3], 9 !< 2 so swap [3,5,2,9]

Run again:

Compare x[0] and x[1], 3 < 5 so no swap

Compare x[1] and x[2], 5 !< 2 so swap [3,2,5,9]

Compare x[2] and x[3], 5 < 9 so no swap

Run again:

Compare x[0] and x[1], 3 !< 2 so swap [2,3,5,9]

Compare x[1] and x[2], 3 < 5 so no swap

Compare x[2] and x[3], 5 < 9 so no swap

Run again:

Compare x[0] and x[1], 2 < 3 so no swap

Compare x[1] and x[2], 3 < 5 so no swap

Compare x[2] and x[3], 5 < 9 so no swap

No swaps, so sorting is complete!

Write a program that will do the following:

- Generate an array of 6 random integers.
- Print the list, unsorted.
- Perform a bubble sort.
- Print the sorted list.

6.3 File Handling Exercises

6.3a - Simple Database

Your task is to use a simple database that uses 5 fields - ID, shade, red, green & blue (to represent the RGB colour values for that particular shade, in the range 0-255) with a space between each value. The data file would typically look like this:

1 black 0 0 0

2 white 255 255 255

3 red 255 0 0

4 green 0 255 0

5 blue 0 0 255

- Create a program with a procedure that will create a new file called "colours" and populate it with the data above.
- Create a procedure that will output the entire contents of the file.
- Call both procedures and check that they work.
- Create a function that will print a menu to the screen asking which colour is to be looked up, prompt for a value and return it as an integer.
- Create a function that will look up a given colour (given its ID) and return 4 strings for the shade and RGB values.
- Write a main program that will populate the file, print the menu and print the RGB values for the chosen shade.

Extension 1: Write the menu procedure so that the shade will automatically be shown (rather than being hard-coded)

Extension 2: Improve the menu further so that it will produce as many menu items as there are lines in the data file (it might be useful to use the string "EOF" to signify the end of the file.

Extension 3: Include an extra menu item to allow the user to add their own shades with associated RGB colours. This should include suitable validation to prevent impossible or incomplete entries.

6.3b - ATM - Full Version

Northern Frock needs you to write a program for their new ATMs (or Automatic Teller Machines).

In this version you will need to use a customer database that will initially look like this (ignore the top row):

ID	Title	First Name	Surname	Balance
1057	Mr.	Jeremy	Clarkson	£172.16
2736	Miss	Suzanne	Perry	£15.62
4659	Mrs.	Vicki	Butler-Henderson	£23.91
5691	Mr.	Jason	Plato	£62.17

Write a program that will do the following:

- Generate the above data file.
- Prompt the user for their ID number, validating against the list of known users.
- If the ID number 99999 (5x 9s) is given the machine should shut down.
- Print a menu:

    Welcome to Northern Frock

    1 - Display balance

    2 - Withdraw funds

    3 - Deposit funds

    9 - Return card

    Enter an option:

- If '1' is entered, display the current balance and also the maximum amount available for withdrawal (must be a multiple of £10)
- If '2' is entered, provide another menu with withdrawal amounts of £10, £20, £40, £60, £80, £100, Other amount, Return to main menu & Return card.
- Check that the requested withdrawal is allowed, print a message to show that the money has been withdrawn and calculate the new balance.
- If 'Other amount' is selected then prompt the user for an integer value. Check this number is a multiple of 10 and that the withdrawal is permitted (as above).
- If '3' is entered, provide another menu that will allow the user to enter an amount to deposit(does not need to be a multiple of £10), return to main menu or return card. If funds are deposited, provide appropriate feedback and update the balance.
- If '9' is entered, print a goodbye message and return to the initial prompt (for the next customer).
- If another value is entered, print an error message and print the menu again.

Extension 1: Add an extra field to the database for a 4 digit PIN which should be prompted for and checked following the entry of the ID number. The user should also have the option to change their PIN.

Extension 2: Add another field to record whether the card is blocked. Three incorrect PIN attempts should permanently lock the card. PIN attempts should only be reset by correctly entering the PIN. Simply removing the card and trying again should not work.

Extension 3: Create an encoding algorithm that will store the PIN in a format that cannot be read by just looking t the file. Only by running the PIN attempt through the same algorithm can the match be found.