POLYTECHNIC COMPUTER SCIENCE - SPRING 2012
PROGRAMMING PROJECT #3
INDEPENDENT PROJECT #1

ASSIGNMENT OVERVIEW

In this assignment you'll be creating a program called *independentproject1.py*, the subject of which you will choose from the options listed below.

This assignment is worth 100 points and is due on the crashwhite.polytechnic.org server at midnight (00:00:00) on the date given in class.

BACKGROUND

You've learned how to use *control structures* (looping and branching) and some *data structures* (numbers, strings, and lists). Apply what you've learned to one of the following programs.

PROGRAM SPECIFICATIONS

Create one of the following programs:

a. **Blackjack**
A deck of cards is shuffled (using a module provided by the instructor), the cards are dealt to the player and the computer (the dealer), new cards are dealt to both player and computer as indicated, and the winner is announced.

b. **Card Dealer**
A deck of cards can be shuffled, and hands dealt as required by the user.

c. **Mugwump**
This program allows the user to play a game of Mugwump, in which four "mugwumps" hide on a 10-by-10 grid. Homebase is the lower left position of the grid (0,0), and the user has ten guesses to find the mugwumps. Each guess is entered as a pair of whole numbers 0-9,0-9, separated by commas, with the first number being the *x*-coordinate of the guess, and the second being the *y*-coordinate of the guess. After each guess, the program calculates the distance between the user's guess and each of the mugwumps, and reveals that information to the user so that he/she can eventually track down the location of all of the mugwumps.

d. **Monkeys write Shakespeare**
The *infinite monkey theorem* states that a monkey hitting random keys on a typewriter for an infinite amount of time will eventually produce the works of Shakespeare. Write a program that tests this hypothesis. (Hint: You *don't* have infinite time for the assignment. Perhaps testing for the phrase "Alas, poor Yorick," or even "Alas" would be a good start.)

e. **Calculator**
The user enters numeric values and operations as desired, and the program calculates a result.

DELIVERABLES
independentproject1-lastnamefirstinitial.py

You should keep a copy of this file in your home folder on the server. To submit your assignment for grading, copy your file to the instructor's Public/Dropbox folder at *crashwhite.polytechnic.org* before the deadline.

1. Please be sure to use the specified file-naming convention.
2. Save a copy of your file on your hard drive, flash drive, etc..
3. Your grade will be based on the file you upload to the instructor's Public/Dropbox folder.

ASSIGNMENT NOTES

- Specific challenges to each of the different programs will only arise as you begin working. Keep in mind that programming is an *iterative* process, which has two important implications:
    - You can start big ("top-down design") or start small ("bottom-up design"), but you can expect that there will be a fair amount of writing/re-writing of code as your program slowly approaches completion.
    - "Test early, test often." Writing a whole program and expecting that it's going to work straight away is a strategy doomed to fail. While there's no single "best way" to write a program, everyone agrees that it's folly to "write a bunch of code and see if it works." Write one small function, make sure it's working. Then write another function and see if it works. Write a main program that calls the two functions, and make sure that works. By working incrementally, it will be easier to figure out what went wrong when your program fails. And believe me... it *will* fail. And then you'll fix it! Welcome to programming.

- For the Blackjack program, you can focus on the mechanics of the blackjack game itself without worrying about how to shuffle the cards by importing a **carddeck** module, available on the course website. Download "carddeck.py" and place it in the same directory as your blackjack program, then "import carddeck" to be able to use that module. See the end of this handout for ways that you can use the carddeck module in your program.

- For the Card Dealer program, one of the challenges is figuring out how to represent a card: is it just a number, from 1-52? How does one represent the "pips" (A, 2, 3,...,10, J, Q, K)? How does one identify the suit of each card?
  One approach is to have the cards numbered, 0-51. The pips on each card can be represented by a number 0-12, with 0 being an Ace and 12 being a King. You can identify the pips of a particular card by calculating **cardnumber** % 13, which is the remainder left over after an integer division. To identify the suit of a particular card, calculate **cardnumber** / 4, where "/" represents the whole number of an integer division. (In Python 3, this is //.) This yields a number from 0 to 3, representing the 4 different suits.

- The Mugwump program requires using "a list within a list," which is fairly common in programming. Each row consists of 10 elements indexed 0-9, and each of those rows consists of a column of 10 elements, also indexed 0-9. These rows and columns, officially referred to as *lists* in Python, are more commonly called *arrays*, and the Mugwump program is based on a 2-dimensional array.
  We can create that array as follows:

```
board = []  # initialize empty list
for j in range(0,10):
    board.append([])       # Adds an empty list item to board
    for i in range(0,10):
        board[j].append(0) # Appends a 0 to board[j] 10 times

# First element is board[0][0], next is board[0][1], through to board[9][9]
```

GETTING STARTED

1. With paper and pencil, and perhaps in collaboration with a partner, identify what the main components are that you'll need to include in your program.
2. Sketch out the basic flow of your program using a flowchart, and write some pseudocode that you can use to begin implementing those main components.
3. You can use commented pseudocode to start coding, of course.
4. Keep a copy of your program on the *crashwhite.polytechnic.org* server. Consider keeping several versions of your program as you work on editing it and improving it.
5. Fix old problems before adding new components. You'll repeatedly run through this edit-run, edit-run process to find bugs and fix them.
6. When your program is completed (but before the deadline), copy it to the server.

QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. Do you find that you prefer writing one long program, and scrolling up and down while you work on it? Or do you prefer working with smaller functions, which simpler themselves, but more difficult to get to work with each other?
2. Is your program "bulletproof," i.e. will it continue to function without any runtime errors, regardless of what the user enters? We'll be looking at more sophisticated ways to prevent different types of errors in your programs in the future.

THE CARDDECK *MODULE*

The *carddeck* module, available in the assignments folder, can be downloaded and used for your Blackjack program. Examples of how to use this module are presented here in an interactive Python session. Further questions about the module are available from help(carddeck) and by asking the instructor in class.

```
>>> import carddeck
>>> myDeck = carddeck.Deck()  # creates a new deck called "myDeck"
>>> myDeck.shuffle()  # shuffles the cards!
>>> thisHand = myDeck.deal(2)
>>> print thisHand
[[5, '5', 'Hearts'], [6, '6', 'Clubs']]
>>> myDeck.remaining() # tells how many cards are still left in the deck
50
```

SAMPLE INTERACTIONS
Sample output from running the various programs might look like this:

```
What is your name? Richard
Let's play Blackjack, Richard!
Dealing
Richard, your cards:
10 of Spades
2 of Clubs
Your total is 12.
Would you like another card (y/N)? y
You got a 8 of Diamonds
your total is 20
Would you like another card (y/N)?
You got 20. Now it's MY turn!
```

```
Enter how many cards you'd like dealt to you: 7
Your cards are here in a list, which includes their point value, a string
representing their pips, and the suit:
[[3, '3', 'Spades'], [3, '3', 'Clubs'], [11, 'Jack', 'Clubs'], [1, 'Ace',
'Spades'], [9, '9', 'Clubs'], [4, '4', 'Spades'], [4, '4', 'Hearts']]
There are 45  cards left in the deck now.
```

```
Let's play Mugwump!
Do you want instructions (y/N)?
9 | 0 0 0 0 0 0 0 0 0 0
8 | 0 0 0 0 0 0 0 0 0 0
7 | 0 0 0 0 0 0 0 0 0 0
6 | 0 0 0 0 0 0 4 0 0 0
5 | 0 0 0 0 0 0 0 0 0 0
4 | 0 0 0 0 0 0 2 0 0 0
3 | 0 0 0 0 0 0 0 0 0 0
2 | 1 3 0 0 0 0 0 0 0 0
1 | 0 0 0 0 0 0 0 0 0 0
0 | 0 0 0 0 0 0 0 0 0 0
  +---------------------
    0 1 2 3 4 5 6 7 8 9
Here we go!
It's turn # 1
Where do you want to check for a Mugwump (x,y)?3,4
Distance to Mugwump # 1 is 3.60555127546
Distance to Mugwump # 2 is 3.0
Distance to Mugwump # 3 is 2.82842712475
Distance to Mugwump # 4 is 3.60555127546
It's turn # 2
Where do you want to check for a Mugwump (x,y)?
```

```
# SAMPLE OUTPUT (looking for "yorick")
JulesVerne:~ 12:33 $ python Desktop/monkeys.py
We found it!
Yahoo! We found it after 640649539 letters typed!
JulesVerne:~ 12:52 $
```