Notes about Homework #4

Professor Hugh C. Lauer

CS-1004 — Introduction to Programming for Non-Majors

(Slides include materials from *Python Programming: An Introduction to Computer Science*, 2nd edition, by John Zelle and copyright notes by Prof. George Heineman of Worcester Polytechnic Institute)

Multi-module program

- Big jump in complexity
- Big jump in stress!
- More planning needed …
- … just to get started!

Homework #4

- **Three separate modules required**

1. **Playing the game**
   - I.e., rolling dice
   - Figuring out whether win or lose
   - Continuing with multiple rolls
   - Reporting win/loss to calling function

2. **Keeping and reporting the statistics**
   - Collecting the win/loss data
   - Creating the plot in **`matplotlib`**
   - Inferring probabilities from simulation

3. **Managing the other two**
   - Starting and finishing the program
   - Calling the right functions at the right time
   - Seeding random number generator!
   - Vehicle for test code — i.e., code that exists only to help you develop other code

Homework #4

- **Maximum function length = 25 lines**
- **However, good practice ⇒ < 10–15 lines!**
- **Make your modules into logical units — e.g.,**
  - Playing the game
  - Recording and keeping statistics
  - Control (and testing)
- **Within each module, make functions into logical units**
  - Loop for calling **`PlayOneGame()`** is one function
    - Returns *two* values — win/loss, # of rolls
  - Recording result of one game is another function
  - Analyzing results is a third function
- **Testing requires additional code that is not part of solution!**

Yeah! — I understand all that, but I just cannot get my head around it.

Let's look at the pieces

- **Start *near* the beginning …**
- **… but not *at* the beginning**

- ***One* module, *one* function**
  - Play one game of craps
  - Roll dice multiple times
    - Decide whether win, lose, or continue!
  - Return a pair of results!!

- **Test it!** **How?**
  - Write a separate module called *Wrapper, Control, Test,* or something like that
  - Add another parameter to **`PlayOneGame()`** called **`verbose`**
    - Defaults to **`False`**
    - Prints out every roll if **`True`**
  - Manually check, convince yourself that it is correct

Questions so far?

Digression 1

- Returning more than one value from a function

```
count  = …
if win:
   return True, count
else:
   return False, count
```

- Caller:–

```
winLoss, nRolls = PlayOneGame()
```

Digression 2

- **Default values of parameters**
  - **`def F(n, d=someValue):`**
- **When calling F,**
  - Second argument is optional!
  - Defaults to **`someValue`** if not specified
- **Example uses**
  - In **`PlayOneGame()`** to control verbose printing!
  - As a seed random number generator
    - During development only!

Let's look at the pieces

- **Start *near* the beginning …**
- **… but not *at* the beginning**

- ***One* module, *one* function**
  - Play one game of craps
  - Roll dice multiple times
    - Decide whether win, lose, or continue!
  - Return a pair of results!!

- **Test it!** **How?**
  - Write a separate module called *Wrapper, Control, Test,* or something like that
  - Add another parameter to **`PlayOneGame()`** called **`verbose`**
    - Defaults to **`False`**
    - Prints out every roll if **`True`**
  - Manually check, convince yourself that it is correct

Next Step

- **In a new module, write one or more functions …**
- **… repeatedly call `PlayOneGame( )`**
  - i.e., within a for-loop
- **Build up a data structure to record wins and losses**
  - Allows plot of wins *vs.* count
  - Separate plot of losses *vs.* count
- **Test again**
  - Modify wrapper to prompt for number of games, collect results in data structure
  - Inspect result manually
- **Finally, plot**

Testing Step #3

- Beef up your wrapper (i.e., control module or test module, or whatever you call it)
- …
- Print out and inspect result manually

Still not done!

- **Need to ask and answer questions in Homework assignment!**
  - What is the probability that player wins eventually wins?
  - What percentage of games are decided on 1st roll? 2nd roll? 3rd roll? etc.?
  - What is average number of rolls per game?
  - …
- **You need to invent code to get these answers**
  - In the wrapper!

Questions?

Additional note

- **`random.randint(a, b)` is same as `random.randrange(a, b+1)`**
  - I.e., **`random.randint(1,6)`** simulates a normal six-sided die with faces 1, 2, 3, 4, 5, 6
    - See Python docs §9.6

Brainstorm about Data Structure

- **What is required output?**
  - A: Graphs of #s of games versus #s of throws
- **What function can produce that output?**
  - A: `pyplot.plot()`
- **What arguments does `pyplot.plot()` need to produce that output?**
  - A: Lists
  - Specifically, y-value list contains numbers of games indexed by numbers of throws
  - What about x-value list?
    - Is it needed at all? If not, why? If so, how organized?
  - List value indicates number of games ending in that number of throws
    - Separate lists for wins and losses

But wait!

- **How big should the list be if number of throws is unknown and/or unbounded?**

- **Answer:–**
  - Make it grow to accommodate the results of experiments!
  - Add more entries as needed
    - Zeros for intervening values

Homework #4

Easy parts:–

- **Playing one game**
  - Multiple rolls of dice
  - Returning a pair of results
- **Rolling the dice**
  - Two independent dice
    - Values 1 – 6 each
  - Use `random.randint()`
    - Variant of `random.randrange()`
    - See Python docs §9.6

Harder parts:–

- **Setting up the lists**
- **Growing lists to accommodate long games**

Questions?