Fall 2020 COP 2930 Final Exam Part A Solution

1) (6 pts) A juice costs 75 cents and a soda cost 90 cents. Complete the program below so that it asks the user how many juices to buy and how many sodas to buy and prints out the number of dollars and cents (in between 0 and 99, inclusive) those juices and sodas will cost. (There is no tax.) Note that the way the given code is, you must use variables named dollars and cents in your solution. (You may use other variables as well, if you would like to.)

Sample Solution

```
numJuice = int(input("How many juices to buy?\n"))
numSoda =  int(input("How many sodas to buy?\n"))

totalCents = 75*numJuice + 90*numSoda

dollars = totalCents//100
cents = totalCents%100

print("Total cost =",dollars,"dollars and",cents,"cents.")
```

Grading: 2 pts for total cents calculation, 2 pts for dollar calculation (must keep as int for full credit), 2 pts for cents calculation (must use ints only for full credit)

2) (8 pts) In other programming languages, there is a "matching-else" problem, where if there are two nested if's but only one corresponding else, there is some perceived ambiguity as to which if the else matches to. In Python, this problem does not exist. What rule is used in Python to determine which else a particular if matches to? Show an example where there are two nested if's and one else, explain what that example does and (a) how one can tell which if the one else matches and (b) why you designed the else to match there.

Sample Solution

Python relies on indenting to determine which if a particular else matches. Indenting is both visually and electronically unambiguous, so students rarely have a problem in Python determining which else matches an if. In the following code sample, a student only gets a scholarship if they maintain a 3.5 gpa or greater. If they ALSO have a 1300 SAT score, their scholarship value is $4000. If they don't the value of the scholarship is $3000. Otherwise, there is no scholarship. Assume the values have been read into the variables and the variables store what their names indicate:

```
scholarship = 0
if gpa >= 3.5:
    if sat >= 1300:
        scholarship = 4000
    else:
        scholarship = 3000
```

In this example, the else matches the inner if because the indentation matches the inner if. The reason I designed it this way is that the scholarship should ONLY be set to anything but 0 if the first if is true and that specifically, if the first if is true and the second is false, then the scholarship variable should be set to 3000, and therefore, the proper place to do this is inside the first if, in the else clause of the second, nested if. A student without the requisite GPA but with a high SAT score would earn no scholarship, since the GPA requirement is the "gatekeeper" for any scholarship.

Grading: 4 pts for saying indentation, 2 pts for their example containing nested ifs with one else, 2 pts for the explanation as to the example set up. Be lenient on the last 2 points. My example is FAR more detailed than I expect students to give under timed pressure.

3) (8 pts) A simple way to find the largest integer less than or equal to the cube root of a given positive integer, *n*, is to start with the integer 1, cube it, and compare it to *n*. If this cube is less than or equal to *n*, then we add one to our integer and repeat the process. If this cube is greater than *n*, we stop our loop and know the answer is one less than the number we last tried. Here is a quick example of the algorithm running for $n = 400$

$1^3 = 1 \leq 400$, so go to the next integer
$2^3 = 8 \leq 400$, so go to the next integer
$3^3 = 27 \leq 400$, so go to the next integer
$4^3 = 64 \leq 400$, so go to the next integer
$5^3 = 125 \leq 400$, so go to the next integer
$6^3 = 216 \leq 400$, so go to the next integer
$7^3 = 343 \leq 400$, so go to the next integer
$8^3 = 512 > 400$, so we STOP!
Since 8 was the first number that was too big, our correct answer is 7.

Complete the code below so that it simulates this process and finds the largest integer less than or equal to the cube root of n (entered by the user) and prints this value out to the screen. You must use a variable res as shown below, but you may use any additional variables, if you like.

Sample Solution

```
n = int(input("Please enter a positive integer.\n"))

x = 1
while x*x*x <= n:
    x += 1
res = x-1

print("Largest integer <= cuberoot of",n,"is",res)
```

Grading: 2 pts for looping mechanism, 2 pts for calculating x*x*x for each appropriate x, 2 pts for handling x starting at 1 and incrementing, 2 pts for properly handing the potential off by 1 error. (can give 1 of 2 pts if say they do < n instead of <= n, for example)

4) (8 pts) Complete the segment of code below so that it prints out a Manhattan Distance Grid, showing the Manhattan Distances from the top left corner of the grid. Let the user enter the number of rows and columns for the grid. The Manhattan distance is defined as the sum of the zero based row and column number of a location. Here is the grid that would print out for 3 rows and 4 columns:

```
0    1    2    3
1    2    3    4
2    3    4    5
```

Separate out each item in a row with a tab. Please fill in the code segment started below:

Sample Solution

```
numRows = int(input("How many rows for your Manhattan Grid?\n"))
numCols = int(input("How many columns for your Manhattan Grid?\n"))

for row in range(numRows):
    for col in range(numCols):
        print(row+col,end="\t")
    print()
```

Grading Criteria: 2 pts row loop, 2 pts col loop, 2 pts print, 1 pt tab, 1 pt newline print