Chris Gregg and Wil Kautz
CS 106A

Handout #7
June 30, 2020

Assignment #2: Khan-sole Academy and Computing Interest

Due: 10:30am (Pacific Daylight Time) on Tuesday, July 7th

Based on problems by Jerry Cain, Eric Roberts, Nick Bowman, Sonja Johnson-Yu, Kylie Jue, and the current CS106A staff.

This assignment consists of multiple short problems to give you practice with several different concepts in Python (e.g., variables, control structures, etc.) as well as two longer programs. You can download the starter code for this project under the "Assignments" tab on the CS106A website. The starter project will provide Python files for you to write your programs in.

The assignment is broken up into two parts. The first part of the assignment will give you practice writing short programs (we call them "sandcastles"). In the second part of the assignment you'll get the chance to create two larger programs. In the spirit of learning how to code, in this assignment we're going to use code to help others learn! The second part of this assignment includes a mini version of Khan(-sole) Academy as well as a program to show users how to compute interest in a bank account.

Part 1: Sandcastles

"Sandcastle" problems are meant to be fairly straightforward and make sure you're solid on particular concepts (like control flow, variables, and functions) before moving on to writing larger programs. They're kind of like building sandcastles in a sandbox – they're meant to be fun to do and no one gets hurt. Please make sure you do these six problems before moving on to other parts of the assignment.

1. For this problem, you will be tracing some code (below) by hand. **Please try doing this by hand rather than plugging it into PyCharm (or another Python interpreter).** What is printed to the console when the following function call is made? Put your answer in the file `trace.txt` provided in the Assignment 2 folder.

```
"""
This program is just meant to test your understanding
of variables, control flow, and functions.
"""

def mystery():
    x = 3
    x = 5 - x / 2
    print(x)

def main():
    x = 1
    while x < 20:
        print(f"x = {x}")
        mystery()
        x *= 2
    print(f"x = {x}")

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

2. Write a program in the file **`subtract_numbers.py`** that reads two ***real numbers*** from the user and prints the first number minus the second number. You can assume the user will always enter valid real numbers as input (negative values are fine). Yes, we know this problem is really similar to a problem we did in class – that's why this problem is a sandcastle! A sample run of the program is shown below (user input is in ***italics***):

```
This program subtracts one number from another.
Enter first number: 5.5
Enter second number: 2.1
The result is 3.4.
```

3. Write a program in the file **`liftoff.py`** that prints out the calls for a spaceship that is about to launch. Countdown the numbers from 10 to 1 and then write "Liftoff!" Your program should include a **`for`** loop using **`range`**. A sample run of the program is below.

```
10
9
8
7
6
5
4
3
2
1
Liftoff!
```

4. Write a program in the file **`random_numbers.py`** that prints 10 random integers (each random integer should have a value between 0 and 100, inclusive). Your program should use a constant named **`NUM_RANDOM`**, which determines the number of random numbers to print (with a value of 10). It should also use constants named **`MIN_RANDOM`** and **`MAX_RANDOM`** to determine the minimal and maximal values of the random numbers generated (with respective values 0 and 100). To generate random numbers, you should use the function **`random.randint()`** from Python's **`random`** library (which we discussed in class). A sample run of the program is shown below.

```
35
10
45
59
45
100
8
31
48
6
```

5. Humans first landed on the moon on July 20, 1969. People have often wondered how much they would weigh if they were on the moon. It turns out that on the moon, you would weigh 16.5% of your weight on Earth. Write a program **`moon_weight.py`** that asks the user for their weight (you can assume is a real-valued input is given by the user) and prints out their weight on the moon. If the user enters a negative value (yes, they're being mean and trying to break your program), you should just print out that weights can't be negative. Remember when writing your solution, that it's good programming style to use <u>constants</u> where appropriate. Two sample runs of the program are shown below (user input is in ***italics***).

Sample run 1 (note the imprecision of floating-point numbers in the answer)

```
Enter your weight: 165.3
Your weight on the moon is 27.274500000000003
```

Sample run 2

```
Enter your weight: -13
Sorry, you can't have a negative weight.
```

6. In geometry, you learned the Pythagorean theorem for the relationship among the lengths of the three sides of a right triangle:

$$a^2 + b^2 = c^2$$

which can alternatively be written as:

$$\mathrm{c} = \sqrt{a^2 + b^2}$$

Write a program in the file **`pythagorean.py`** that gets two values for **`a`** and **`b`** as floats (you can assume that **`a`** and **`b`** will be positive real values) from the user and then calculates the solution of **`c`** and prints it. Recall that to compute square roots, you can use the function **`math.sqrt()`** from Python's **`math`** library (which we discussed in class). A sample run of the program is shown below (user input is in ***italics***):

```
Enter values to compute the Pythagorean theorem.
a: 9.7
b: 3.2
c = 10.21420579389313
```

Part 2: Longer programs

1. Khan-sole Academy

Now that you've seen how programming can help us in a number of different areas, it's time for you to implement Khan-sole Academy—a program that helps other people learn! In this problem, you'll write a program in the file `khansole_academy.py` that randomly generates simple addition problems for the user, reads in the answer from the user, and then checks to see if they got it right or wrong, until the user appears to have mastered the material.

More specifically, your program should be able to generate simple addition problems that involve adding two 2-digit integers (i.e., the numbers 10 through 99). The user should be asked for an answer to each problem. Your program should determine if the answer was correct or not, and give the user an appropriate message to let them know. Your program should keep giving the user problems until the user has gotten 3 problems **correct *in a row***. (Note: the number of problems the user needs to get correctly in a row to complete the program is just one example of a good place to specify a constant in your program).

A sample run of the program is shown below (user input is in ***italics***).

```
What is 51 + 79?
Your answer: 120
Incorrect.  The expected answer is 130
What is 33 + 19?
Your answer: 42
Incorrect.  The expected answer is 52
What is 55 + 11?
Your answer: 66
Correct!  You've gotten 1 correct in a row.
What is 84 + 25?
Your answer: 109
Correct!  You've gotten 2 correct in a row.
What is 26 + 58?
Your answer: 74
Incorrect.  The expected answer is 84
What is 98 + 85?
Your answer: 183
Correct!  You've gotten 1 correct in a row.
What is 79 + 66?
Your answer: 145
Correct!  You've gotten 2 correct in a row.
What is 97 + 20?
Your answer: 117
Correct!  You've gotten 3 correct in a row.
Congratulations!  You mastered addition.
```

When Chris was learning how to program, he wrote a number of similar programs to practice math and language. He wrote the programs in the BASIC and Pascal languages (go look them up -- that's how old he is!), and now that he thinks about it, they were terribly decomposed (he would do much better these days!)

2. Computing Interest

We want to write a program `compute_interest.py` that helps a user compute how much interest their bank account will accrue over time. The program starts by asking the user for an initial account balance, which is entered as a float (you can assume a positive real-value is entered). The program then asks the user for a starting year and month as well as an ending year and month, all entered as integers. The program then asks the user for a monthly interest rate. For example, a 2% interest rate would be entered by the user as the value 0.02.

The program should print out the monthly balance in the account from the starting year/month up to and including the ending year/month with interest accruing monthly. The amount of interest earned each month is simply the amount in the account in that month multiplied by the interest rate entered by the user. So (using a 2% interest rate) an account with $1000.00 at the start of the month would earn $20.00 that month and have a total of $1020.00 at the start of the next month.

After printing the monthly balance projection, your program should then repeat the process of asking the user for a new interest rate and printing the monthly balance in the account using the same starting account balance, starting year/month, and ending year/month as the user *initially* entered in the program. The program should end if the user specifies a monthly interest rate of 0%.

You can assume that the user will always enter a valid (i.e., positive) value for the start/end year and an integer from 1 to 12 for the start/end month. If the starting year/month is the *same as or after* the ending year/month, your program should not ask for an interest rate, but rather simply print "Starting date needs to be before the ending date." and then end without an error.

The values for the initial balance, the starting year/month, and the ending year/month are entered by the user. The monthly balance is simply printed as a float, so you don't have to worry about rounding to the nearest cent.

Remember that it's a good idea to decompose your program and use parameters with functions as needed to pass information around your program. That's really part of good programming style.

Two sample runs of the program are shown below (user input is in ***italics***).

Sample run 1 (showing what happens if the user enters a starts year/month that is *after* the ending year/month. Note the program simply ends without asking for an interest rate.)

```
Initial balance: 500.00
Start year: 2020
Start month: 9
End year: 2019
End month: 10
Starting date needs to be before the ending date.
```

Sample run 2

```
Initial balance: 1000.00
Start year: 2020
Start month: 5
End year: 2022
End month: 3
Interest rate (0 to quit): 0.02
Year 2020, month 5 balance: 1000.0
Year 2020, month 6 balance: 1020.0
Year 2020, month 7 balance: 1040.4
Year 2020, month 8 balance: 1061.208
Year 2020, month 9 balance: 1082.43216
Year 2020, month 10 balance: 1104.0808032
Year 2020, month 11 balance: 1126.162419264
Year 2020, month 12 balance: 1148.6856676492798
Year 2021, month 1 balance: 1171.6593810022655
Year 2021, month 2 balance: 1195.0925686223109
Year 2021, month 3 balance: 1218.994419994757
Year 2021, month 4 balance: 1243.3743083946522
Year 2021, month 5 balance: 1268.2417945625452
Year 2021, month 6 balance: 1293.6066304537962
Year 2021, month 7 balance: 1319.478763062872
Year 2021, month 8 balance: 1345.8683383241296
Year 2021, month 9 balance: 1372.7857050906123
Year 2021, month 10 balance: 1400.2414191924245
Year 2021, month 11 balance: 1428.246247576273
Year 2021, month 12 balance: 1456.8111725277984
Year 2022, month 1 balance: 1485.9473959783543
Year 2022, month 2 balance: 1515.6663438979213
Year 2022, month 3 balance: 1545.9796707758796
Interest rate (0 to quit): 0.05
Year 2020, month 5 balance: 1000.0
Year 2020, month 6 balance: 1050.0
Year 2020, month 7 balance: 1102.5
Year 2020, month 8 balance: 1157.625
Year 2020, month 9 balance: 1215.50625
Year 2020, month 10 balance: 1276.2815624999998
Year 2020, month 11 balance: 1340.0956406249998
Year 2020, month 12 balance: 1407.1004226562497
Year 2021, month 1 balance: 1477.4554437890622
Year 2021, month 2 balance: 1551.3282159785153
Year 2021, month 3 balance: 1628.894626777441
Year 2021, month 4 balance: 1710.3393581163132
Year 2021, month 5 balance: 1795.8563260221288
Year 2021, month 6 balance: 1885.6491423232353
Year 2021, month 7 balance: 1979.9315994393971
Year 2021, month 8 balance: 2078.928179411367
Year 2021, month 9 balance: 2182.8745883819356
Year 2021, month 10 balance: 2292.0183178010325
Year 2021, month 11 balance: 2406.619233691084
Year 2021, month 12 balance: 2526.950195375638
Year 2022, month 1 balance: 2653.29770514442
Year 2022, month 2 balance: 2785.962590401641
Year 2022, month 3 balance: 2925.260719921723
Interest rate (0 to quit): 0
```

Possible Extensions

Once you've completed all the required parts of the assignment, you might want to consider adding some extensions.

Extend Khansole Academy

You could consider extending your Khansole Academy program to, for example, add more problem types (subtraction, multiplication, division, and more). You could also consider problems beyond arithmetic. If you could build your own version of Khansole Academy, what would you use it to help people learn? Be creative and enjoy.

Hailstones

A separate (optional) problem you could consider writing is based on a problem in Douglas Hofstadter's Pulitzer-prize-winning book *Gödel, Escher, Bach*. That book contains many interesting mathematical puzzles, many of which can be expressed in the form of computer programs. In Chapter XII, Hofstadter mentions a wonderful problem that is well within the scope of what you know. The problem can be expressed as follows:

> Pick some positive integer and call it *n*.
> If *n* is even, divide it by two.
> If *n* is odd, multiply it by three and add one.
> Continue this process until *n* is equal to one.

On page 401 of the Vintage edition of his book, Hofstadter illustrates this process with thfollowing example, starting with the number 15:

15	is odd, so I make $3n+1$:	46
46	is even, so I take half:	23
23	is odd, so I make $3n+1$:	70
70	is even, so I take half:	35
35	is odd, so I make $3n+1$:	106
106	is even, so I take half:	53
53	is odd, so I make $3n+1$:	160
160	is even, so I take half:	80
80	is even, so I take half:	40
40	is even, so I take half:	20
20	is even, so I take half:	10
10	is even, so I take half:	5
5	is odd, so I make $3n+1$:	16
16	is even, so I take half:	8
8	is even, so I take half:	4
4	is even, so I take half:	2
2	is even, so I take half:	1

As you can see from this example, the numbers go up and down, but eventually—at least for all numbers that have ever been tried—comes down to end in 1. In some respects, this process is reminiscent of the formation of hailstones, which get carried upward by the winds over and over again before they finally descend to the ground. Because of this analogy, this sequence of numbers is usually called the **Hailstone sequence,** although it goes by many other names as well.

You might want to write a Python program that reads in a number from the user and then displays the Hailstone sequence for that number, just as in Hofstadter's book, followed by a line showing the number of steps taken to reach 1. For example, here's a sample run of what such a program might look like (user input is in ***italics***):

```
Enter a number: 17
17 is odd, so I make 3n + 1: 52
52 is even, so I take half: 26
26 is even, so I take half: 13
13 is odd, so I make 3n + 1: 40
40 is even, so I take half: 20
20 is even, so I take half: 10
10 is even, so I take half: 5
5 is odd, so I make 3n + 1: 16
16 is even, so I take half: 8
8 is even, so I take half: 4
4 is even, so I take half: 2
2 is even, so I take half: 1
The process took 12 steps to reach 1
```