COMP 1900 - Fall 2021

Lab 2: Variables and Expressions

Total Points: 32

Due: Fri., Sept. 17, by 2359 CDT
Please carefully read the submission instructions at the end of the assignment. As with all other assignments, this should be done *individually*.

Grader: The lab TA who supervises your registered lab section will grade your submission. Grades will generally be posted within 1-2 weeks of the assignment due date. Questions about grading? Please contact them first.

1. (7 points) The basic unit of currency in the European Union is the **euro** (€), which is subdivided into 100 cents (c). Euro coins come in 1c, 2c, 5c, 10c, 20c, 50c, €1, and €2 denominations.

   Write a program named `euro_change_maker.py` that allows the user to enter a number of euro cents. The program should then compute and display a combination of the euro coins above that add up to that amount. Use the same change making algorithm that we discussed in class for American coins.

   Here's an example of what your completed program might look like when you run it. Underlined parts indicate what you type in as the program is running.

   Example program run (underlined parts indicate what the user enters)

```
Enter number of euro cents: 2597

2 euro coins: 12
1 euro coins: 1
50 cent coins: 1
20 cent coins: 2
10 cent coins: 0
5 cent coins: 1
2 cent coins: 1
1 cent coins: 0
```

2. (4 points) A common programming operation is to **swap** or exchange the values of two variables. If the value of `x` is currently 19 and the value of `y` is 42, swapping them will make `x` into 42 and `y` into 19.

   Write a program named `swap.py` that allows the user to enter two `int` values. The program should store those values into variables and then swap the variables. Print the values of both variables before and after the swap. You will receive full credit as long as your solution works, but try to do this using as few variables as possible.

   Example program run (underlined parts indicate what the user enters)

```
Enter value for x: 19
Enter value for y: 42

Before swap: x = 19, y = 42
After swap: x = 42, y = 19
```

3. (7 points) Consider the following "magic trick":

   - Think of any integer. Let's say we pick 12.
   - Multiply the number by 2: $12 \times 2 = 24$
   - Add 12 to the previous result: $24 + 12 = 36$
   - Divide the previous result by 2: $36/2 = 18$
   - Subtract the original integer from the previous result: $18 - 12 = 6$

   The final result will *always* be 6, regardless of the original number. Such amaze. So wow.

   Write a program named `questionable_magic_trick.py` that simulates performing this trick. Your program should start by getting user input for the original number and storing that into a variable. Then, change the value of that variable according to each step of the trick. After each step, print the result of that step on the screen. Use an integer division in the division step to avoid seeing any decimal points in the result.

   Do *not* make a new variable for each step of the trick. Instead, have your program change the value of the existing variable. You should need only two variables in all.

   Example program run (underlined parts indicate what the user enters)

```
You stand in the presence of the Magnificent Magic Mademoiselle Millicent.
Prepare to be amazed!
Enter an integer: 12

OK, watch this...
We'll multiply by 2 and get 24
Then we'll add 12 and get 36
Then we'll divide by 2 and get 18
Then we'll subtract the original number and get... 6.  Amazing!!!
```

4. (7 points) You probably know that any point on Earth's surface can be located using latitude (ϕ) and longitude (λ). Given two points with latitude-longitude coordinates (ϕ_1, λ_1) and (ϕ_2, λ_2), the distance d between them can be calculated using the **haversine formula**:

$$d = 2R\ \texttt{atan2}(\sqrt{A}, \sqrt{1-A})$$

where R is the mean radius of Earth (3958.8 mi) and A is defined as

$$A = \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)$$

Note that we can't just use the well-known distance formula, since that doesn't account for the curvature of Earth. (I hope there are no flat-Earthers in this class...) Also note that these calculations assume Earth is perfectly spherical, and they don't consider elevation.

Write a program named `distance_on_globe.py` that allows the user to enter the latitude and longitude of two points (in degrees) and computes the distance (in miles) between them.

Hint: The trig functions in the haversine formula can be computed using Python's built-in math functions (from the `math` module): `math.sin`, `math.cos`, and `math.atan2`. These trig functions require that the angles are expressed in radians, so your program needs to convert the user's input. Remember that π rad = 180°. You can use the built-in constant `math.pi` to get the value of π accurate to about 15 decimal places.

Example program run (underlined parts indicate what the user enters) [1]

```
Enter lat/long of starting point (in degrees):
Lat: 35.0421
Long: -89.9792

Enter lat/long of ending point (in degrees):
Lat: 33.6407
Long: -84.4277

Distance is 331.13020585065516 mi.
```

5. (7 points) When making major purchases such as real estate or vehicles, most people don't pay the entire amount up front. Instead, they **finance** the purchase by taking out a loan at a certain interest rate, and they agree to pay back the loan in monthly installments over a period of time.

[1] If you're curious, the locations in this example are Memphis International Airport and Hartsfield-Jackson Atlanta International Airport.

To compute the amount of a monthly payment, you need the following information:

- P, initial amount of loan (in dollars)
- r, *monthly* interest rate (as a decimal), which is the annual interest rate divided by 12
- n, number of *months* to make payments

Given those values, the monthly payment M can be computed using the formula

$$M = P\frac{r(1+r)^n}{(1+r)^n - 1}$$

For example, suppose you're financing a \$20,000 car over 5 years, at an annual interest rate of 4.8%. Then, $P = 20000$, $r = 0.048 \ / \ 12 = 0.004$, and $n = 5 \times 12 = 60$. Substituting those values into the formula yields a monthly payment of $M = 375.59$.

Note that the total amount you pay is \$375.59 per month over 60 months, which comes out to \$22,535.69. Your loan ultimately costs an extra \$2,535.69 in interest! Higher interest rates and/or longer loan terms will result in paying more over time.[2]

Write a program named `monthly_payments.py` that allows the user to enter the following inputs:

- Initial amount of loan, in dollars
- *Annual* interest rate, as a percentage (e.g., 4.8 for 4.8%)
- Number of *years* to pay back the loan

Your program should then calculate and show the monthly payment, the total amount paid over the entire loan term, and the total interest paid, all rounded to the nearest cent. Note that the user inputs the *annual* interest rate and the number of *years* to pay back the loan. However, the monthly payment formula requires values in terms of *months*. Be sure to convert accordingly.

Example program run (underlined parts indicate what the user enters)

```
Please enter the following information:
Initial amount of loan:      150000
Annual interest rate (in %): 4.25
Number of years:             30

Monthly payment: $737.91
Total paid:      $265647.54
Interest paid:   $115647.54
```

[2]In general, *very* few things are worth financing. If you can't afford to buy it in cash, you probably shouldn't be buying it...

Need Help?

- Attend your weekly lab meeting. Your lab TA can give you live help during this session.
- Contact your lecture instructor, and/or attend office hours.
- The UofM offers free online tutoring through the Educational Support Program (ESP): `https://www.memphis.edu/esp/onlinetutoring.php`

Submission Instructions

- Create a zip file containing your Python source files. You can use any Python development environment you like, as long as you submit the source files.
- Upload your zip file to the appropriate dropbox folder on eCourseware. The dropbox *will* cut off submissions at precisely the stated deadline, so please submit with some time to spare. Late submissions are not accepted. You may submit as many times as you want up to the deadline. Each submission overwrites the previous one.
- Contact your lab TA to schedule a one-on-one code review via Zoom. During this short meeting (15-30 minutes), your TA will ask you to run your code. They may also ask you to explain your code and/or run different test cases. The code review must be completed for you to get a grade for the lab.

  Note that since the code review happens after your submission, its purpose is *not* for you to get help with writing your code! Getting help should be done during the scheduled lab sessions, or by reaching out to your lecture instructor/online tutors *before* the due date.