**CS5001**

**Spring 2019**

**HW2 (Functions and conditionals)**

**Assigned:** January 17, 2019

**Deadline:** January 24, 2019 at 9:00am

To submit your solution, compress all files together into one .zip file. Login to handins.ccs.neu.edu with your CCIS account, click on the appropriate assignment, and upload that single zip file. You may submit multiple times right up until the deadline; we will grade only the most recent submission.

Your solution will be evaluated according to our [CS5001 grading rubric](http://bit.ly/cs5001rubric). The written component accounts for 20% of your overall HW2 score; the programming component for 80%. We’ll give written feedback when appropriate as well; please come to Laney’s office hours with any questions about grading (email to set up a time if no current office hours work for you).

Review both of this week’s handouts before you begin the assignment: [Writing Functions](https://docs.google.com/document/d/1hnQgjJAdh4plYdgLt_3Xu7sLAoHcwwOYacPaNTSEaV0/edit) and [Testing and Debugging](https://docs.google.com/document/d/1-TNlzh03hchSfOihfj74gsmVLmuPdQZmTqja4lGCTOo/edit).

This is an introductory course, and we want to make sure that everyone has a solid understanding of the fundamentals, so we’ll often rule some Python tools in or out. For this homework, do not use lists or loops -- we haven’t gotten there yet, and you don’t need them!

If you apply your late token, your new deadline is **January 28, 2018 at 9:00am**. Email laneys@northeastern.edu to notify us you’ll be cashing it in.

**Written Component**

Please open a plaintext file (you can do this in IDLE or any plaintext editor you like, such as TextEdit or NotePad) and type out your answers to the questions below.

**Written #1**

What does each of the following Python programs print to the terminal?

**1A**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **def spam():**  **word = "yabba"**  **def main():**  **word = "dabba"**  **print(word)**  **spam()**  **print(word)**  **main()** |

**1B**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14** | **def eggs(word):**  **print(word)**  **def spam():**  **word = "yabba"**  **eggs(word)**  **def main():**  **word = "dabba"**  **print(word)**  **spam()**  **print(word)**  **main()** |

**1C**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14** | **def eggs(scrambled):**  **print(scrambled)**  **def spam():**  **canned = "yabba"**  **eggs(canned)**  **def main():**  **word = "dabba"**  **print(word)**  **spam()**  **print(word)**  **main()** |

**Written #2**

For each of the functions below -- identified by name, parameters, and return type -- write a line of Python code that demonstrates how you would call that function. Pass literals as your arguments to the function (no need to define variables); these literals can have any values you like. If the function returns something, save the result in a variable.

**2A**

Name: print\_formatted

Parameters: one string

Return type: nothing (void)

**2B**

Name: get\_max

Parameters: **two integers** <=== !~~list of integers~~

Return type: int

**2C**

Name: log\_base\_two

Parameter: int

Return type: int

**Written #3**

For each Python snippet below, what would be printed to the screen?

**3A**

|  |  |
| --- | --- |
| **1**  **2**  **3** | **if 6 > 2 \* 5:**  **print("Hello")**  **print("World")** |

**3B**

|  |  |
| --- | --- |
| **1**  **2**  **3** | **if 6 - 2 < 5:**  **print("Hello")**  **print("World")** |

**3C**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **x = 3**  **y = 2**  **if x > 2:**  **if y > 2:**  **z = x + y**  **print("z =", z)**  **x = x + y**  **print("x =", x)** |

**Programming Component**

There are three programming components for this homework, and you’ll be graded on each one separately, but they all plug in together.

There are lots of ways to measure the distance between any two points. In computer science, we use the “distance” between two things in lots of ways, such as:

* Google Maps calculates the distance from point A to point b
* Lyft calculates the distance from you to the nearest driver
* A recommendation engine calculates the distance between things you already like and a new thing. Less distance means more similarity. (*Oh, you like dramas,* Hulu says. *Here, try The First.*  <=== it recommended this new show to me recently, apparently because I like the Handmaid’s Tale. Sean Penn is in it. It’s OK.)

For this homework, we’re going to use the idea of distances for a machine-learning algorithm called *nearest neighbors.*

You’ve likely heard of Machine Learning. It’s a field within computer science where we teach computers to think so they can eventually become our overlords. A typical machine-learning algorithm provides a computer with a set of “training” data to provide the computer with known answers. Then an unknown “testing” set is provided, and the program figures out the answer to this new material on its own.

This particular machine-learning algorithm, nearest neighbors, works as follows:

* There is a community of training data: a bunch of data points.
* A new data point is added.
* The nearest neighbors algorithm uses a distance function to determine which of the known points in the community it’s closest to.

**Programming #1 -- Defining Functions**

* Filename: distance.py

Do not write a *main* function in your distance.py file. Here, you’ll just define functions. We’ll be doing a lot of “rolling our own,” so you **may not use** any of the following:

* Python’s abs function
* Python’s pow function
* Python’s sqrt function

For this part of the assignment, your functions must match the spec exactly -- same name, parameters, and return type. This is strictly so we can test your code with testing code we’ve already written, which will contribute to the correctness part of your grade. (In Programming #3, you’ll be asked to come up with your own functions, because doing that well is an important part of computer science, too.)

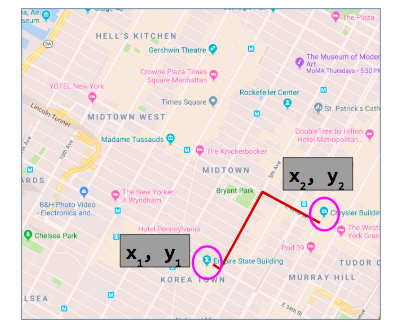
*Function #1*

* **Name**: **absolute**
* **Input**: one number (float or int)
* **Returns**: a number (float or int), the absolute value of the given input.

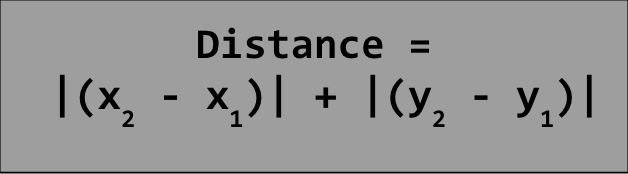
The absolute value of a number is its magnitude without regard for its sign.

*Function #2*

* **Name**: **manhattan**
* **Input**: four numbers (ints or floats), representing two coordinates (x1, y1, x2, y2)
* **Returns**: a float, the manhattan distance between the two coordinates

You’re in Manhattan, in the part where the streets are laid out like a grid. How far is it from point A to point B? It’s the number of blocks you need to walk, using only right angles, to get there. 

With an *x* and a *y* coordinate for each point, we calculate the absolute value of the difference between the *x* coordinates and the *y* coordinates, and then add them. The formula:



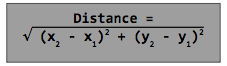
*Function #3*

* **Name**: **euclidean**
* **Input**: four numbers (ints or floats), representing two coordinates (x1, y1, x2, y2)
* **Returns**: a float, the euclidean distance between the two coordinates

Euclidean distance is more like the distance measure you might be familiar with if you measure points on a plane in geometry.



With an *x* and a *y* coordinate for each point, we calculate the absolute value of the difference between the *x* coordinates and the *y* coordinates, and then add them. In general, the Euclidean distance between two points is given by:



AMAZING points for this programming assignment:

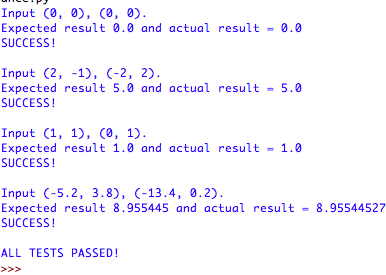
* Each function is 4-6 lines long, not including comments.
* Comments for each function specify the name, parameters, and return type and do so *incredibly* clearly, like a person who has never heard of that function before could understand what it does.

**Programming #2 -- Testing Your Functions**

* Filename: **test\_distance.py**
* Starter code for reference: [**test\_euclidean.py**](http://www.ccs.neu.edu/home/laney/cs5001/spring19/code/hw2/test_euclidean.py)

Now you’ve written your distance functions, and you want to be sure they’re working as expected before you plug them into a larger program. If there’s a problem with any of them, much better to find it now than when you’re in the middle of a big, complicated program that they’re smaller parts of. In the starter code linked above, we’ve provided code that tests the euclidean function. (The absolute function is also called, but not specifically tested, so make sure it’s working before your run our test code.)

To make sure your Euclidean code is working, download and save the above file in the same directory as your distance.py and then execute it like you would any other Python program. If you see a message like this, your Euclidean code is great!



How did I know what result each test should generate? I just wrote them out by hand and used a calculator. (I did **not** write my code and then just run it to see what the output was. Let’s none of us get into that habit, even now while our assignments are relatively straightforward.)

Now your job is to write similar test code for **absolute** and **manhattan**. Follow the example above and as outlined in [this week’s handout](https://docs.google.com/document/d/1-TNlzh03hchSfOihfj74gsmVLmuPdQZmTqja4lGCTOo/edit) to structure your test cases. In particular, you are required to:

* Test each function with at least 4 different inputs
* Test edge cases (no distance between two points, no change from original number to absolute value, that kind of thing)
* Print the input to be tested, expected output, and outcome (SUCCESS/FAILURE) from every test you run, and the total number of tests passed/failed.

AMAZING points for this programming assignment:

* Tests are so thorough that we’re convinced no stone is left unturned
* Include in the block comment at the top of your code either (1) an error that your testing turned up, or (2) an error that *might have* occurred, but writing the test code reminded you to fix something you overlooked. If neither of those things happened, then use this block comment to describe your approach to writing the functions and avoiding possible errors.

**Programming #3 -- Putting it All Together**

* Filename: **nearest\_neighbor.py**

Ok, now we’ve got a couple of distance functions and we know for sure that they work. Now let’s put this into a little Machine Learning program to calculate a data point’s nearest neighbor.

We’ll apply the Nearest Neighbor algorithm in this program to find the geographically-nearest neighbor based on longitude and latitude. We’ll have two vital starting points, locations of a Starbucks and a Dunkin’ Donuts. (The actual data points are completely made up, though; I have no idea if there are coffee shops there at all.)

* Starbucks: (67.35, 71.18)
* Dunkin’: (22.33, -173.09)

You’ll write a program, a regular one with a *main* and everything, that uses Python’s turtle function to draw those two initial points on the screen, and then:

* Prompt the user for their location (*x* and *y* coordinates).
* Prompt the user for which distance function to use, Euclidean or Manhattan.
* Draw their new location on the screen.
* Report which is their nearest neighbor, based on their chosen distance function.

If you put all of this code into main, it’ll get messy fast. You’ll need to write functions for this part of the assignment; you can put them in the same file, just define them above main. And the job of each function, its name, parameters, and return type are all up to you.

*Requirements*:

* Include three test cases in your block comment at the top: Given a particular *x, y* input, which is closer, Starbucks or Dunkin?
* Draw the initial points in one color, and the new point in another. We used red and blue, but you can choose any colors you like.
* Make good use of functions.
  + Every function should be small, and should have ONE job.
  + Function names are similar to variable names, but are usually verbs.
  + Parameter names, like variable names, should be short and descriptive.
  + Put a block comment inside each function describing its parameters and return type, and optionally its name and short description.
  + As a rule of thumb, a function should be no more than 30 lines long.
* You may assume that the user provides good input (floats/ints for the coordinates, and ‘*E’* or ‘*M’* for the distance function choice).

*Helpful Hints:*

* We used Python’s **circle** function to draw the initial locations, but **dot** would probably work, too.
* The turtle’s **penup** function is useful to move the turtle without drawing a trail.
* Depending on how you set up your code, the graphics-based functions can be challenging to test programmatically. We won’t require you to write a test suite for this part, but you do still need to make sure it all works -- so write your test cases before you begin.

AMAZING points:

* Each of your functions, including main, is less than 25 lines of code.
* Vertical space between chunks of related code is absolutely perfect and easy to follow.
* The output is pretty, obviously that’d need to be prettier than the example.
* Use constants for the initial data points.
* Use descriptive variable names (not just x1, x2, x3, etc.)

Example of running this program (your wording can vary, and use any colors/sizes you like in the graphics):

