Thicccque
October 14, 2018
Dr. Parham-Mocello

Assignment 2 Part 1

Orientation:

1. Interesting Special Interest Groups (SIGs):
   - SIGAI is the special interest group that focuses on the research and development of artificial intelligence. It is mainly the study of intelligence and its implementation in computer systems, and support AI-related conferences [1].

     [1] SIGAI - Artificial Intelligence. (2018). Retrieved October 11, 2018 from https://www.acm.org/special-interest-groups/sigs/sigai

   - SIGCSE is the special interest group that focuses on the methods and curricula of computer science education at every level. It provides a forum for educators around the world to interact and collaborate to further computing education [1].

     [1] SIGCSE - Computer Science Education. (2018). Retrieved October 11, 2018 from https://www.acm.org/special-interest-groups/sigs/sigcse

2. A paper from each of the interesting Groups: (summary + bibliography + questions).
   - Deep Neural Networks for YouTube Recommendations:

     This paper discusses in great detail the large effect neural networks and general deep learning has had on the performance and efficiency of YouTube's recommendation algorithm [1]. They first discuss some of the problems that such a system might face, such as the scale, freshness, and noise. Regarding scale as a potential issue they point out that, due to just how massive YouTube is, the training data stored on some disk is overwhelmingly large and the latency for serving these recommendations could build up if not circumvented. Freshness, as they call it, might be an issue in the specific context of YouTube because the corpus, or pool of content the neural network must sort through and recommend items of, is constantly changing and growing at a pretty ridiculous rate as hours of video are uploaded every second [2]. Noise, in this case, refers to how much 'junk' the algorithms have to sort through, seeing as the data being worked on is all user-generated and not all usable. Paul Covington also dives into how their deep learning recommendation system operates in general terms. First, in

a process labelled candidate generation, the millions of videos making up the corpus are whittled down to just hundreds of candidates generated for the specific user, by referencing the context of the recommendations (date, time, what video they're coming from, etc) as well as the history and viewing habits of the specific user. This is described as the course-level representation that should contain five to ten good recommendations [2].This pool of hundreds of candidates is then put through a ranking process, which uses data on the videos and their specific features, again the user's history, and other sources like the virality of videos in conjunction to order and then whittle down this pool of recommendations to just dozens of candidates, which are then rendered on the user's device.

Stepping back, within the candidate generation step, there are many layers of inputs and computations that reduce the pool of millions to a pool of hundreds. The first layer is the embedding layer, where ninety-nine percent of all the learnable parameters are located in the process [2]. This is where user history in the form of embedded video watches and tokenized search queries are processed with hundreds of dimensions. Importantly, these are learned jointly with the entire rest of the network, through back-propagation [1]. Next up is the combiner which, because feedforward networks require fixed-sized inputs, combines the vectors resultant from processing the search tokens and watch history. The benefit to this step is that it allows for further tailoring to the specific user through adding in geographic specific factors and specifics such as age and gender. The final layer is a ReLU stack, or a stack of fully-connected Rectified Linear Units, with each layer halving in size, where all the computing occurs and results in a user vector or user "embedding" [2]. After this vector is produced, a softmax is computed to generate the systems best guess as to the likelihood the user will watch each of the resulting videos, leading into that ranking process.

The ability to consider the features of the user is important in the ranking process, as each feature included makes this ranking much more accurate and efficient. The ranking structure is very similar to the candidate selection structure of layering, with a few differences. Firstly, the features and specifics of each candidate are processed with multivalent categorical features (those features that could have multiple values in a training example [2]) being averaged, the language of the video, and the prior experiences the user has had with this or adjacent videos are all processed and combined into a single vector through the same combination process. Again, these combined vectors are placed in a ReLU stack, and out pops a weighted logistic, which is used for ranking. Covington describes the job of the engineers working on this algorithm to be continually engineering more features to increase

the accuracy of the overall network. They also analyzed the effect of depth and width of these stacks, and found that both, when increased, increased the quality of the recommendations, but also that increasing both simultaneously is important to limit the chances of gradient "bottlenecks" [2].

○ Bibliography:

[1] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16). ACM, New York, NY, USA, 191-198. DOI: https://doi.org/10.1145/2959100.2959190

➢ This is the official article compiled by the researchers to showcase the construction of their deep learning system for YouTube recommendations and some data they've collected on the efficacy and efficiency of the system. It is rigorous and well backed by experience of the writers and the long list of reputable references.

[2] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. Video. In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16). ACM, New York, NY, USA. DOI: https://youtu.be/WK_Nr4tUtl8

➢ This is the presentation that delivered the content of this article during the proceedings of the 10th ACM Conference on Recommender Systems (otherwise called RecSys 2016). The information conveyed is identical to the content of the article, but phrased in more comprehensive ways, allowing for further summary to be drawn. The presenter was a primary author of the article and is trustworthy. This video also contains some questions posed of the author by the audience of the presentation, which allows further insight into the process and system described.

○ Questions:

This question was partially posed by an audience member in the presentation the video shows, but I believe it to be valid, so how does this system directly compare to the prior

system used for recommendations? What is the cost increase of implementing this kind of system as opposed to a simpler system that picks recommendations in a less tailored way?

Are the majority of the people in the interest group actively working on some project involving deep learning or AI for a larger company, or is the membership more diversified to people interested in the field, even if not directly involved in the heavy hitters developing in it?

- The Persistent Effect of Pre-College Computing Experience on College CS Course Grades:

  Although there is much research that points at pre-college computer science experience having a large positive impact on students in introductory computer science courses and continued study in the program, there hadn't been much research on how or if that trend continues later on into computer science programs in American colleges. The authors conducted a study encompassing all stages of a CS program at a large public university in America to determine whether grade differences exist between students with and without pre-college experience, and if so, for what types of experiences [1]. They were able to find large grade differences continuing through all courses in the program, not just in the introductory courses. They also, interestingly enough, found that AP Computer Science participation correlated to significantly higher average grades, up to a half grade [1]. Much of this is due to the fact that a student who has had pre-college computer science experience is also much more likely than another student to have had significant exposure to more opportunities when it comes to computing, and especially computing education of any kind. Some students enter college being able to program and having completed multiple projects, and some come in having never touched a computer or a command-line interface or terminal. The article interestingly dives into the specific socio-economic and identity backgrounds of these students, and draws attention to the fact that women, members of some racial and ethnic groups, and those from low socioeconomic backgrounds tend to have less experience than White and Asian middle and upper class men [1]. Comparing a child who studied in a wealthy school districts that give every student access to computing and AP classes, and who probably had access to programs like Saturday Academy to a child who studied in an underfunded district with very few opportunities in this area, this difference in college performance is easily understood. Some colleges have adopted strategies for leveling the playing-field in this area, by teaching separate introductory courses for extra support,

allowing students to choose a focus for their first few courses, and teaching courses that start with material students were all unlikely to have been exposed to, among many other strategies, but it hasn't resolved the issue entirely, and these accommodations are not present in every program [1].

Because of this background, the researchers hypothesized that the advantages of pre-college experience would diminish as study continued and that there would exist some experiences that are more impactful than others when it comes to this prior experience. To test for these hypotheses, the researchers studied four quarters of the undergraduate program at their university. They acquired data from upper and lower division courses, including courses that had been separated by experienced and non-experienced students. For each term, they administered pre-course surveys that included questions on qualitative pre-college computer science experience amount, as well as on which specific classes or programs they had partaken of before their first computer science course at the university. The possible answers of this second question were: "Took AP CS / Took another (non-AP) CS course in high school or middle school / Took one or more CS courses at a college or university / Self-taught one or more programming languages (not including HTML) / Participated in a CS club or extracurricular activity / Learned HTML (either in a class or on your own) / Participated in a summer CS experience (e.g. summer academic program or camp) / No experience" [1]. In these answers, AP CS refers to AP Computer Science A, which is a common AP class offered in high schools that often correlates with concepts covered in introductory computer science courses. AP CS did not refer to AP Computer Science Principles, which was not offered until two years after the study was conducted [1]. The survey also inquired into whether a student came straight out of high school or was a transfer student, to aid in future research.

To quantify the students performance in these courses, the researchers used final course grades, continued study in the computer science program (whether participants in lower division courses later completed Advanced Data Structures, the first required upper-division computer science course), and pre-test scores from two courses. These pre-tests, were given early in two terms each of a Computer Science 2 and Advanced Data Structures, and were designed to test the prerequisite knowledge for each course. The instructors of these courses modeled the pretest questions on quiz questions that would have been given toward the end of the preceding courses. Students were graded on completion [1]. Overall, the study found that the average grade differences between students with a little experience and a lot of experience

before college narrowed as study progressed further, but still that having pre-college experience continued to give a sizable advantage to students throughout their studies. The difference on the 4.33 grading scale between the highest scoring group (usually a lot of experience but sometimes some experience) and the lowest scoring group (exclusively those with no experience prior to college) ranged from .1 to over .5, which is over half a grade-point. This partially proved and partially disproved their first hypothesis, as the performance disparity did exist in the lower division courses, but, counter to the researchers' hypothesis, this disparity did not diminish much as studies continued.

Then, in order to compare the different specific experiences students may have had before entering the college program, the researchers compared grades between a few specific sets of experience and the remaining students respectively. They categorized the different kinds of experience as either formal or informal, where formal experiences included AP Computer Science, other high school courses, and college experiences; and informal experiences included computer science clubs, summer programs, self-teaching, and learning HTML. Within and without these categorizations, some comparisons had no specific merit or were inconclusive, but they specifically isolated those with informal experience and those with formal training each against those with no prior experience, as well as those who had taken AP Computer Science as opposed to those who had not. Although they did not find a significant average grade difference between those with exclusively informal experience and those without. The results showed significant differences in grade when comparing those with formal experience to those without and slightly higher differences when it came to whether a student had taken AP Computer Science in all courses but the Computer Networks course, in which formal experiences of any kind had a .6 average grade-point difference. These trends continued somewhat into retention in the computer science program, although differences were much reduced or eliminated in some cases. The researchers admit a few possible sources of error, including that these conclusions are drawn from correlation and do not imply causation, and that the study did not inquire into whether students had the option of engaging in prior experiences and accepted or rejected that option or if they just never had the opportunity. They conclude action should be taken to reduce these gaps and to level the playing field and also that further research should be done on the subject.

○ Bibliography:

[1]Christine Alvarado, Gustavo Umbelino, and Mia Minnes. 2018. The Persistent Effect of Pre-College Computing Experience on College CS Course Grades. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). ACM, New York, NY, USA, 876-881. DOI: https://doi.org/10.1145/3159450.3159508

➢ This is the official article compiled by the researchers to discuss their study on the effects of prior experience on college students in the computer science program. The article is extremely rigorous, covering every base, and is well backed with extensive references. The section on possible obstacles to the validity of the study is also rigorous and examines the shortcomings of the study well, which enforces this article as a reputable document and source of inspiration for further study. There is no element that seemingly disappeared without further notice and every aspect of the researchers thought-process regarding what information to include and compare is expressed and valid. Highly reputable document.

○ Questions:

How big of an impact would the researchers consider the mere presence of many laptops or computing opportunities in early education to be? Is this something worth researching, because I definitely have some ideas for some research on this idea. In conjunction, they mentioned it, but would they think the percentage of students who take-up opportunities for formal or informal experiences offered through k-12 programs correlates with the exposure these students faced to computing early in education? I'm curious there are primary and middle schools with ipads or laptops in every room and those with a single lap of computers or less. Do these circumstances influence the students path in highschool given the same opportunities? In college?

3. Definitions:
    - Open-source software:
        - Open-source software is generally software that anyone can modify and share at their discretion, because the source-code or fundamental design is publically available.
    - Proprietary software:
        - Proprietary software is software that is specifically owned by an individual or company (typically the developer, although proprietary software can be traded for or sold) and whose source-code is not publically available for manipulation or even view.
    - Shareware software:
        - Shareware software is software that is owned by a company or individual that allows for users to use it for free or on a trial-basis with the understanding that at some future point, the user may need or want to pay for its use.
    - System software:
        - System software is software that is designed and intended to provide a platform for other software, whether for development or dissemination. Notable examples of system software include operating systems and game engines. It is also described as the interface between user / application software and the hardware.
    - Application software:
        - Application software is software that is designed to complete some tasks or provide some service for the express benefit or ease of life of the user.

Computation:

4. Using George Polya's problem-solving steps to find the largest number out of 5 random numbers given to you from a friend.
    a. Understanding the Problem:

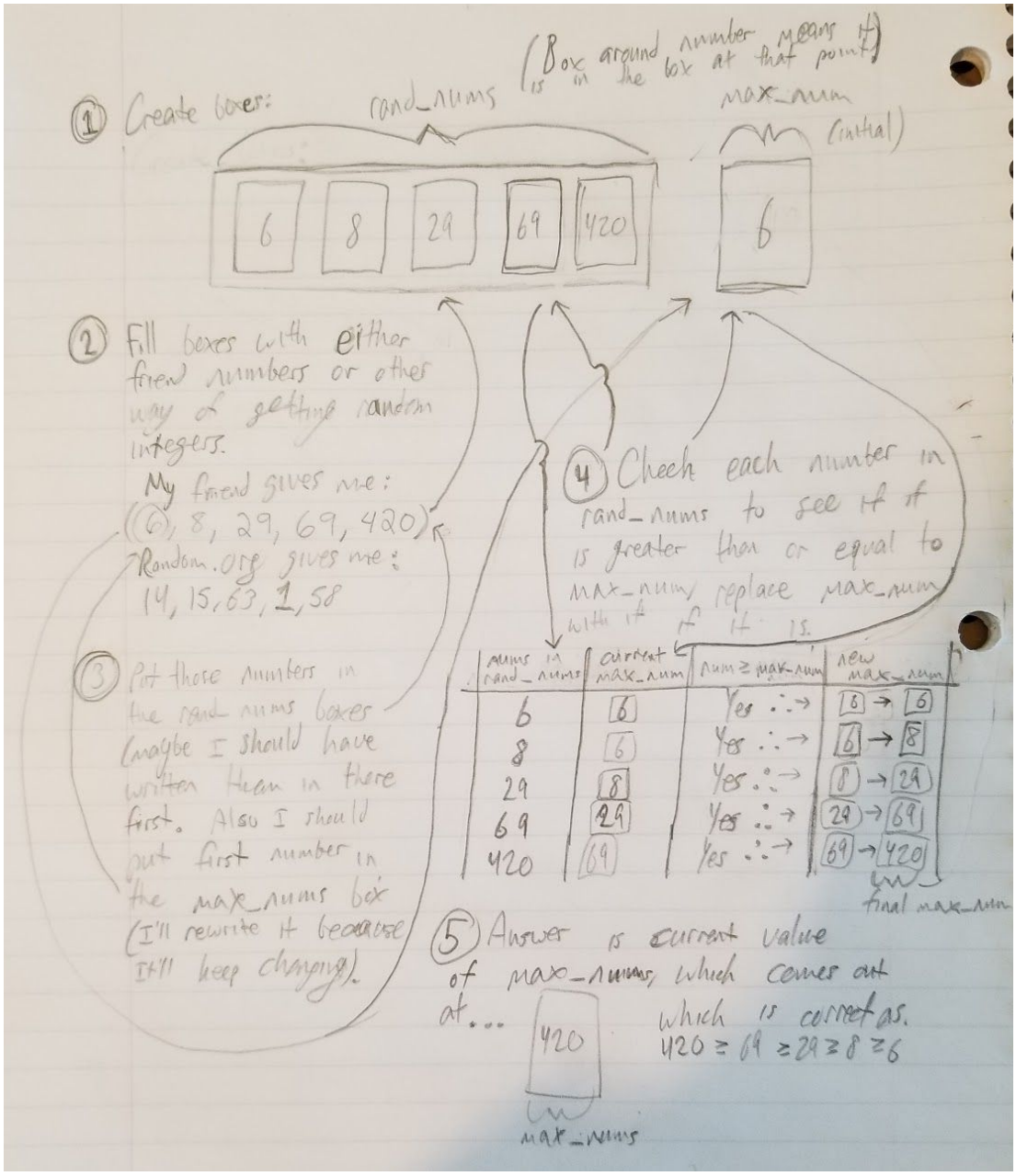
        I will have to be able to compare the numerical values of 5 numbers (must assume a type, as not specified, so I'm assuming type of integer) using code. I must first store them for ease of access in some variables for comparison. I must have a way of keeping track of either the value of the biggest number to give back the largest value or exactly which number was the biggest, if I want to give back the exact instance of the number I was working with. The numbers are 'random' and 'from a friend.' Because of this, I must have some way of getting 5 random numbers, either by generation or by having another person choose 5. I have no friends so this will be an issue to overcome.

b. Devise a Plan:

To solve the problem I need to overcome each of the problem's specifics, listed above. Starting with storing the numbers, I need to have some variables to store the values. I could use individual variables for each, but I see that leading to future issues, so I'll store them in some kind of list, which has many boxes contained in one larger, easier to reference box that I can sort through in an algorithmic way (I wonder how I could implement this… list). I'll call this bigger box rand_nums for the random numbers contained within it. Because I might not have friends, I'll have two ways of getting the five numbers. First, I could have a friend give me 5 numbers to put in these 5 boxes. If I do not have a friend, I will give myself the option of getting 5 random numbers using some other tool, like rolling dice, using a random number generator, or tapping into the entropic nature of the universe to conjure randomness. I'll also put in these two different options because I've solved intro to computer-science questions hundreds of times and if I don't do something different or practice documentation then I'll go insane. To figure out which number I've obtained is the largest, I can use the fact that their in an ordered list to simplify my process. Instead of remembering exactly which instance of a number is the largest, it'd be more efficient to just store the biggest value in another box I'll call max_num for the maximum value number found in the numbers. I'll sort through the numbers in my rand_nums listed order, so I don't accidentally skip one of them. For each of the numbers (huh, for each... Sounds vaguely programm-y.) I'll compare it to the biggest number I've found, which is conveniently stored in my max_num box. But wait a minute, if I'm looking at the first number, I can't compare it to an empty box, so I guess I'll have to put something in there. If I put the smallest number I can think of in there, (because the real world and some programming languages allow the integers to be unbounded and so there is always a lower number than I name), a random number or someone with malicious intent could break my process by giving me exclusively numbers smaller than my assumed smallest possible number, making my max_num inaccurate. Because I can't put in a really small number that will always be beaten by at least one number, I'll just say that the biggest number (stored in max_num) starts as the first number in my rand_nums box, so that every other number has a chance to beat it, and it beats itself. That's another thing. I'll let numbers beat numbers of the same value just in case. I'll be checking to see if my given number is greater than or equal to the max_num found so far. So at this point, for each number in my rand_nums box, I'll see if it's greater than or equal to the current value stored in the max_num box, and if it is, I'll just replace that max_num value with the number that beat it

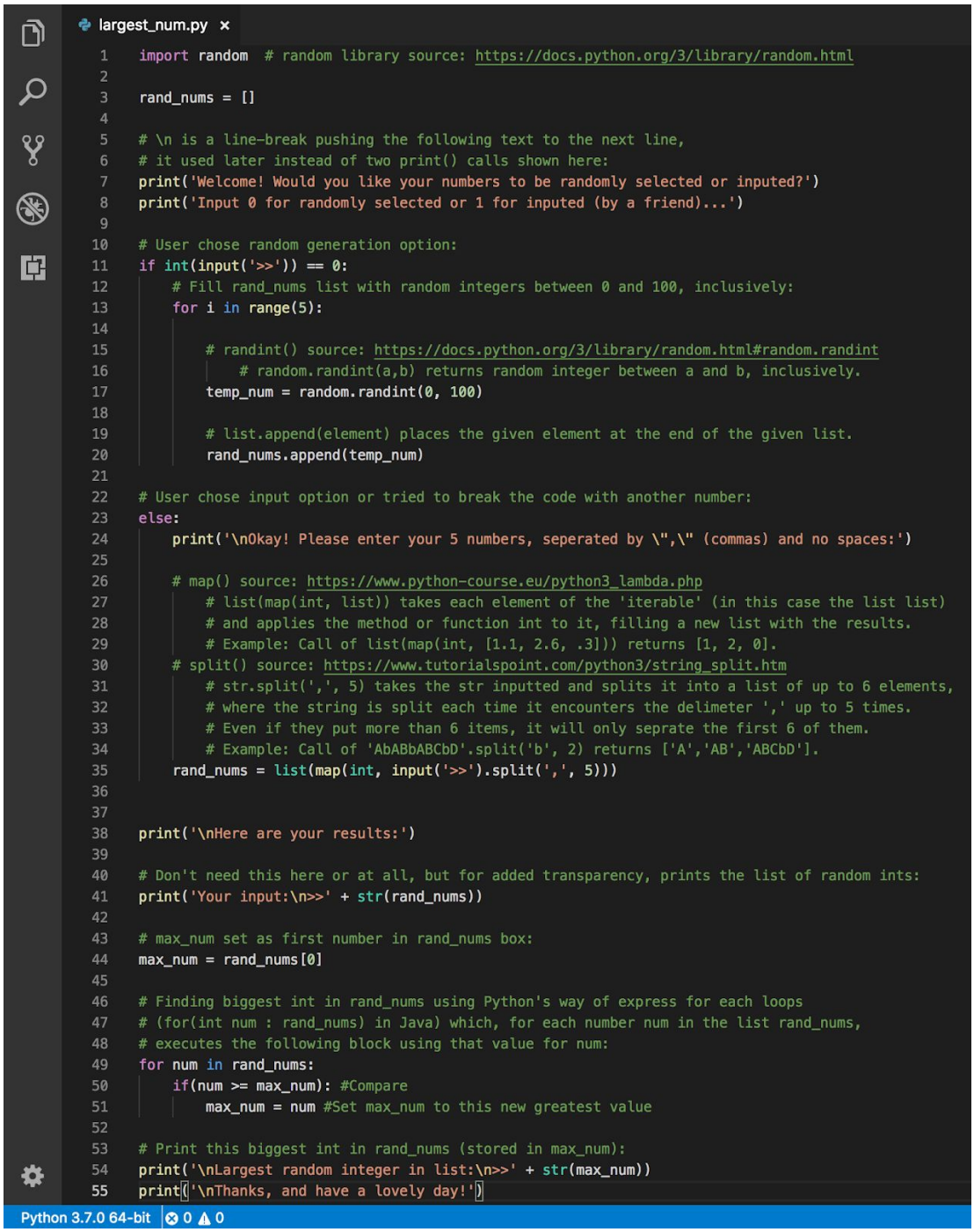
and keep checking the rest of the numbers. After I've checked each number in rand_nums, max_nums should contain the value of the biggest number. At this point I should have found the largest number of the 5 random numbers.

c. Looking Back



d. Carry out the Plan:

I carried the plan out exactly but also added a few things to make the terminal look nicer as well as implementing the “harnessing the entropic nature of the universe” joke through the random library. I used way too many comments so I won’t be docked points and hopefully so that anyone and their mother can read it:



e. Looking Back:

My results do match exactly those from the on-paper trial (although I couldn't have tested my random method with exactly the same values while maintaining its randomness).

In case I can't upload my code as a python file, here is the source code:

```
import random  # random library source:
https://docs.python.org/3/library/random.html

rand_nums = []

# \n is a line-break pushing the following text to the next line,
# it used later instead of two print() calls shown here:
print('Welcome! Would you like your numbers to be randomly selected or inputed?')
print('Input 0 for randomly selected or 1 for inputted (by a friend)...')

# User chose random generation option:
if int(input('>>')) == 0:
   # Fill rand_nums list with random integers between 0 and 100, inclusively:
   for i in range(5):

        # randint() source:
https://docs.python.org/3/library/random.html#random.randint
            # random.randint(a,b) returns random integer between a and b,
inclusively.
       temp_num = random.randint(0, 100)
```

```
        # list.append(element) places the given element at the end of the given
list.
        rand_nums.append(temp_num)

# User chose input option or tried to break the code with another number:
else:
    print('\nOkay! Please enter your 5 numbers, separated by \",\" (commas) and no
spaces:')

     # map() source: https://www.python-course.eu/python3_lambda.php
        # list(map(int, list)) takes each element of the 'iterable' (in this case
the list list)
         # and applies the method or function int to it, filling a new list with the
results.
         # Example: Call of list(map(int, [1.1, 2.6, .3])) returns [1, 2, 0].
    # split() source: https://www.tutorialspoint.com/python3/string_split.htm
        # str.split(',', 5) takes the str inputted and splits it into a list of up
to 6 elements,
         # where the string is split each time it encounters the delimiter ',' up to
5 times.
         # Even if they put more than 6 items, it will only separate the first 6 of
them.
         # Example: Call of 'AbABbABCbD'.split('b', 2) returns ['A','AB','ABCbD'].
    rand_nums = list(map(int, input('>>').split(',', 5)))



print('\nHere are your results:')

# Don't need this here or at all, but for added transparency, prints the list of
random ints:
print('Your input:\n>>' + str(rand_nums))

# max_num set as first number in rand_nums box:
max_num = rand_nums[0]

# Finding biggest int in rand_nums using Python's way of express for each loops
# (for(int num : rand_nums) in Java) which, for each number num in the list
rand_nums,
# executes the following block using that value for num:
```

```
for num in rand_nums:
    if(num >= max_num): #Compare
        max_num = num #Set max_num to this new greatest value

# Print this biggest int in rand_nums (stored in max_num):
print('\nLargest random integer in list:\n>>' + str(max_num))
print('\nThanks, and have a lovely day!')
```