

CSCI 1133, Fall 2019
Programming Assignment 4
Due: 11:55pm, Wednesday October 2, 2019

Due Date: Submit your solutions to GitHub by 11:55 p.m., Wednesday, October 2nd. We will do a pull from this time point. Do not upload anything to Canvas and PLEASE be sure to use proper naming conventions for the file, classes, and functions. We will NOT change anything to run it using our scripts.

Unlike the computer lab exercises, this is not a collaborative assignment. You must design, implement, and test your code on your own without the assistance of anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class (so you are ONLY allowed to reuse examples from the textbook, lectures, or code you and your partner write to solve lab problems). Otherwise obtaining or providing solutions to any homework problem for this class is considered Academic Misconduct. See the syllabus and read section “Academic Dishonesty” for information concerning cheating. Always feel free to ask the instructor or the TAs if you are unsure of something. They will be more than glad to answer any questions that you have. We want you to be successful and learn so give us the chance to help you.

Instructions: This assignment consists of 3 problems, worth a total of 40 points. Solve the problems below by yourself, and put all functions in a single file called `hw4.py`. Use the signatures given for each class and function. We will be calling your functions with our test cases so you must use the information provided. If you have questions, ask!

Because homework files are submitted and tested electronically, the following are very important:

- You follow all naming conventions mentioned in this homework description.
- You submit the correct file, `hw4.py`, through Github by the due date deadline.
- You follow the example input and output formats shown.
- Regardless of how or where you develop your solutions, your programs should execute using the `python3` command on CSELabs computers running the Linux operating system.

Push your work into Github under your own repo. The specific hosting directory should be: `repo-<username>/hw4`, where you replace `<username>` with your U of M user name. For instance, if your email address is `bondx007@umn.edu`, you should push your `hw4` to this directory: `repo-bondx007/hw4`, meaning that the path to your file is `repo-bondx007/hw4/hw4.py`

The following will result in a score reduction equal to a percentage of the total possible points:

- Incorrectly named/submitted source file, functions, or classes (20%)
- Constraints not followed (40%)
- Failure to execute due to syntax errors (30%)
- Not filling out the documentation template for every function, other bad code style (10%)

Problem A. (20 points) Brute Force Password Cracking

You have intercepted some important documents from the Illuminati. The bad news is that they're password-protected. The good news is that they're encrypted with a four letter password, with only lowercase letters (a-z) allowed, so there's only $26^4 = 456976$ possibilities. Trying all of them is a bit unreasonable for a human, but should be easy for a computer.

Download the template file `hw4.py` from Canvas, along with all of the example encrypted .txt files

Since we haven't covered reading from files yet, and most of you probably haven't taken a course on cryptography, we have written the `decrypt` function for you: this takes the text from one of the encrypted files (as a string), and a four letter password as arguments. If the password works, then it prints out the decrypted text and returns the boolean value `True`; if not then it prints out nothing and returns `False`. (If you're interested, the encryption used is a combination of the [Vigenère cipher](#) and [modular exponentiation](#)).

You must edit the function `find_password(filename)`, which is partially written for you already. The function takes in a single string argument `filename`, which is a string representing the name of the file to be decrypted. Currently, it tries to decrypt the data in the file with the password `'pwnd'` (you happen to know that this is the password to `encrypted1.txt`, but it does not work on any of the others). Change the function so that it tries every possible four letter lowercase alphabetic password, and then returns the password once it finds one that works.

Hints:

- The `chr()` built-in function can be used to turn an integer into its ascii character. The ascii values for lowercase a through z are 97 through 122, inclusive.
- Think about how many nested loops you need to hit every possible four-letter password.

Constraints:

- Do not import/use any Python modules.
- Do not use the `input()` function.
- Your submission should have no code outside of function definitions (comments are fine).
- Don't edit the `decrypt`, `vigenere`, or `encode` functions in the template.
- 456976 is a lot of possibilities to check, but `find_password` still should run in under a minute on any lab machine.
- We will test your program on files other than the example files given, so it must try all lowercase 4 letter passwords, not just the ones that work for the test files.

Examples (assumes that you are running this in the same folder as the .txt files; **bold** text indicates the password returned by `find_password`, *italic* text is the decrypted text printed by the `decrypt` function when it finds a match):

```
>>> find_password('encrypted1.txt')  
all your base are belong to us  
'pwnd'
```

```
>>> find_password('encrypted2.txt')  
stan is not what he seems  
'ford'
```

The zipped folder you downloaded should have two more examples, `encrypted3.txt` and `encrypted4.txt`. I won't post the passwords and deciphered text for those two here, but if you get a decrypted phrase out that remotely resembles English, it probably worked.

Problem B. (10 *points*) Counting Primes

Many [better cryptosystems](#) than the one used in the previous problem rely on finding very large prime numbers. In this problem, we'll find all of the primes within a given range.

Recall that a positive integer x is prime if it is divisible by exactly two positive integers: itself and 1. This means that 1 is NOT prime: it is divisible by only one positive integer (1). To determine whether x is prime, you can check whether x is divisible by any integer between 2 and the square root of x (rounded down), inclusive. If not, then it is prime. Think about why you don't have to check potential divisors above the square root of x .

Write a function `count_primes(low, high)` that takes in two positive integers, `low` and `high`. `count_primes` should **return** the number of primes between `low` and `high`, inclusive. It should also **print** out any such primes as it counts, one per line. If `low > high`, print nothing and return 0.

Hints:

- You should consider breaking this problem into two parts. Write a function that determines whether a single integer is prime, and returns True or False. Test that function to ensure that it works. Then, use that function inside `count_primes`.
- If `a % b == 0`, then `a` is divisible by `b`.

Constraints:

- Do not import/use any Python modules.
- Do not use the `input()` function.
- Your submission should have no code outside of function definitions (comments are fine).
- None of the examples below should take longer than a few seconds on a lab computer (if they do, then you might be checking more divisors than necessary).

Examples (text in **bold** is returned, text in *italics* is printed):

```
>>> count_primes(1, 20)
```

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
```

```
8
```

```
>>> count_primes(547120100, 547120200)
```

```
547120117 is prime
547120141 is prime
547120193 is prime
```

3

```
>>> count_primes(79, 97)
```

```
79 is prime
```

```
83 is prime
```

```
89 is prime
```

```
97 is prime
```

4

```
>>> count_primes(3201814, 200)
```

0

```
>>> count_primes(37, 37)
```

```
37 is prime
```

1

Problem C. (10 points) **There's Always a Bigger Fish**

One simple model of predator-prey populations is the [Lotka–Volterra equations](#). This model has three basic tenants, which have at least some basis in reality:

- Without the influence of predators, the prey's population experiences exponential growth (we assume here that the prey always have enough food).
- Without the influence of prey, the predator's population experiences exponential decay (we assume here that the prey is the predator's primary food source).
- Interactions between predators and prey cause the prey's population to go down (because they are eaten), and the predator's to go up (because they are able to feed themselves and their young). Interactions are proportional to both the number of prey in the area and the number of predators.

In this problem, we'll be using a similar model to simulate the populations of three types of fish living in an isolated lake. We'll call these bigfish, middlefish, and smallfish. The bigfish primarily consume middlefish, the middlefish primarily consume smallfish, and the smallfish do not require sustenance because they are magic. Each week, the following occurs:

- The smallfish population increases by 10%
- The middlefish population decreases by 5%
- The bigfish population decreases by 10%

However, the above effects only consider the fish in isolation. In addition, the following predator-prey interactions occur each week:

- Middlefish eat smallfish at a rate of $0.0002 * (\text{\# of smallfish}) * (\text{\# of middlefish})$. For every smallfish eaten, 0.5 middlefish are produced.
- Bigfish eat middlefish at a rate of $0.00025 * (\text{\# of middlefish}) * (\text{\# of bigfish})$. For every middlefish eaten, 0.8 bigfish are produced.

Note that the above calculations should be applied based on the populations of the fish at the **beginning** of the week, before any population growth, decay, or predation has occurred (so you'll probably need to keep track of two sets of population numbers: this week's which you are in the process of updating, and last week's). You may notice that these calculations will produce non-integer numbers of fish: that's fine, since this is just an approximation anyway: keep everything in decimals.

Write a function called `population(small, middle, big)`, which takes three integers as arguments, representing the initial numbers of smallfish, middlefish, and bigfish in the lake, respectively. The function should simulate the change in population each week using the procedure above, and **print** out the populations truncated down to the nearest whole number (continue to store the populations as a decimal, but truncate them for printing purposes). You can use the `int()` built-in function for this. The function should **return** the number of weeks it takes for one of the populations to be essentially wiped out (less than 10 members), or 100 in the case that all three populations are still greater than or equal to 10 after 100 weeks.

Constraints:

- Do not import/use any Python modules.

- Do not use the `input()` function.
- Your submission should have no code outside of function definitions (comments are fine).

Examples (text in **bold** is returned, everything else is printed):

```
>>> population(800, 600, 1000)
Week 1 - Small: 784   Middle: 468   Big: 1020
Week 2 - Small: 789   Middle: 361   Big: 1013
Week 3 - Small: 810   Middle: 280   Big: 985
Week 4 - Small: 846   Middle: 220   Big: 942
Week 5 - Small: 893   Middle: 176   Big: 889
Week 6 - Small: 951   Middle: 143   Big: 831
Week 7 - Small: 1019  Middle: 120   Big: 772
Week 8 - Small: 1096  Middle: 103   Big: 713
Week 9 - Small: 1183  Middle: 91    Big: 657
Week 10 - Small: 1280 Middle: 82    Big: 603
Week 11 - Small: 1387 Middle: 76    Big: 553
Week 12 - Small: 1505 Middle: 72    Big: 506
Week 13 - Small: 1633 Middle: 70    Big: 463
Week 14 - Small: 1774 Middle: 70    Big: 423
Week 15 - Small: 1926 Middle: 72    Big: 386
Week 16 - Small: 2091 Middle: 75    Big: 353
Week 17 - Small: 2269 Middle: 80    Big: 323
Week 18 - Small: 2459 Middle: 88    Big: 296
Week 19 - Small: 2661 Middle: 99    Big: 272
Week 20 - Small: 2875 Middle: 113   Big: 250
Week 21 - Small: 3097 Middle: 133   Big: 231
Week 22 - Small: 3323 Middle: 160   Big: 214
Week 23 - Small: 3549 Middle: 197   Big: 199
Week 24 - Small: 3764 Middle: 248   Big: 187
Week 25 - Small: 3953 Middle: 317   Big: 178
Week 26 - Small: 4098 Middle: 412   Big: 171
Week 27 - Small: 4169 Middle: 543   Big: 168
Week 28 - Small: 4133 Middle: 720   Big: 170
Week 29 - Small: 3951 Middle: 951   Big: 177
Week 30 - Small: 3594 Middle: 1237  Big: 193
Week 31 - Small: 3064 Middle: 1560  Big: 222
Week 32 - Small: 2414 Middle: 1874  Big: 269
Week 33 - Small: 1750 Middle: 2106  Big: 343
Week 34 - Small: 1188 Middle: 2189  Big: 453
Week 35 - Small: 786   Middle: 2091  Big: 606
Week 36 - Small: 536   Middle: 1834  Big: 799
Week 37 - Small: 393   Middle: 1474  Big: 1013
Week 38 - Small: 316   Middle: 1085  Big: 1210
```

Week 39 - Small: 279 Middle: 736 Big: 1352
Week 40 - Small: 266 Middle: 471 Big: 1416
Week 41 - Small: 267 Middle: 293 Big: 1408
Week 42 - Small: 278 Middle: 183 Big: 1350
Week 43 - Small: 296 Middle: 117 Big: 1264
Week 44 - Small: 319 Middle: 77 Big: 1167
Week 45 - Small: 346 Middle: 53 Big: 1069
Week 46 - Small: 377 Middle: 38 Big: 973
Week 47 - Small: 411 Middle: 28 Big: 884
Week 48 - Small: 450 Middle: 22 Big: 800
Week 49 - Small: 493 Middle: 17 Big: 724
Week 50 - Small: 541 Middle: 14 Big: 654
Week 51 - Small: 593 Middle: 12 Big: 590
Week 52 - Small: 651 Middle: 10 Big: 533
Week 53 - Small: 715 Middle: 9 Big: 480

53

>>> population(20,30000,10)

Week 1 - Small: -98 Middle: 28485 Big: 69

1

>>> population(400, 1000, 9)

0

>>> population(1200,400,300)

Week 1 - Small: 1224 Middle: 398 Big: 294
Week 2 - Small: 1248 Middle: 397 Big: 288
Week 3 - Small: 1274 Middle: 398 Big: 282
Week 4 - Small: 1300 Middle: 401 Big: 276
Week 5 - Small: 1326 Middle: 405 Big: 270
Week 6 - Small: 1350 Middle: 411 Big: 265
Week 7 - Small: 1374 Middle: 419 Big: 261
Week 8 - Small: 1396 Middle: 428 Big: 256
Week 9 - Small: 1416 Middle: 439 Big: 253
Week 10 - Small: 1433 Middle: 452 Big: 250
Week 11 - Small: 1447 Middle: 466 Big: 247
Week 12 - Small: 1457 Middle: 481 Big: 246
Week 13 - Small: 1462 Middle: 497 Big: 245
Week 14 - Small: 1463 Middle: 515 Big: 245
Week 15 - Small: 1458 Middle: 533 Big: 245
Week 16 - Small: 1449 Middle: 551 Big: 247
Week 17 - Small: 1434 Middle: 569 Big: 250
Week 18 - Small: 1413 Middle: 587 Big: 253

Week 19	-	Small: 1389	Middle: 604	Big: 258
Week 20	-	Small: 1360	Middle: 618	Big: 263
Week 21	-	Small: 1327	Middle: 631	Big: 269
Week 22	-	Small: 1293	Middle: 640	Big: 276
Week 23	-	Small: 1256	Middle: 647	Big: 284
Week 24	-	Small: 1219	Middle: 650	Big: 292
Week 25	-	Small: 1182	Middle: 649	Big: 301
Week 26	-	Small: 1147	Middle: 644	Big: 310
Week 27	-	Small: 1114	Middle: 636	Big: 319
Week 28	-	Small: 1083	Middle: 624	Big: 328
Week 29	-	Small: 1056	Middle: 609	Big: 336
Week 30	-	Small: 1033	Middle: 592	Big: 344
Week 31	-	Small: 1014	Middle: 573	Big: 350
Week 32	-	Small: 999	Middle: 552	Big: 355
Week 33	-	Small: 989	Middle: 530	Big: 359
Week 34	-	Small: 983	Middle: 509	Big: 361
Week 35	-	Small: 981	Middle: 487	Big: 362
Week 36	-	Small: 983	Middle: 467	Big: 361
Week 37	-	Small: 990	Middle: 447	Big: 358
Week 38	-	Small: 1000	Middle: 429	Big: 355
Week 39	-	Small: 1014	Middle: 412	Big: 350
Week 40	-	Small: 1032	Middle: 397	Big: 344
Week 41	-	Small: 1053	Middle: 384	Big: 337
Week 42	-	Small: 1077	Middle: 373	Big: 329
Week 43	-	Small: 1105	Middle: 364	Big: 320
Week 44	-	Small: 1135	Middle: 357	Big: 312
Week 45	-	Small: 1167	Middle: 352	Big: 303
Week 46	-	Small: 1202	Middle: 348	Big: 294
Week 47	-	Small: 1238	Middle: 347	Big: 285
Week 48	-	Small: 1276	Middle: 348	Big: 276
Week 49	-	Small: 1314	Middle: 351	Big: 268
Week 50	-	Small: 1353	Middle: 356	Big: 260
Week 51	-	Small: 1392	Middle: 363	Big: 252
Week 52	-	Small: 1430	Middle: 373	Big: 246
Week 53	-	Small: 1466	Middle: 384	Big: 239
Week 54	-	Small: 1500	Middle: 399	Big: 234
Week 55	-	Small: 1530	Middle: 415	Big: 229
Week 56	-	Small: 1556	Middle: 434	Big: 225
Week 57	-	Small: 1577	Middle: 456	Big: 222
Week 58	-	Small: 1590	Middle: 479	Big: 220
Week 59	-	Small: 1597	Middle: 505	Big: 219
Week 60	-	Small: 1595	Middle: 533	Big: 220
Week 61	-	Small: 1584	Middle: 562	Big: 221

Week 62	-	Small: 1565	Middle: 592	Big: 224
Week 63	-	Small: 1536	Middle: 622	Big: 228
Week 64	-	Small: 1498	Middle: 651	Big: 234
Week 65	-	Small: 1453	Middle: 678	Big: 241
Week 66	-	Small: 1401	Middle: 701	Big: 249
Week 67	-	Small: 1344	Middle: 721	Big: 259
Week 68	-	Small: 1285	Middle: 735	Big: 271
Week 69	-	Small: 1224	Middle: 743	Big: 284
Week 70	-	Small: 1165	Middle: 744	Big: 297
Week 71	-	Small: 1108	Middle: 738	Big: 312
Week 72	-	Small: 1055	Middle: 725	Big: 327
Week 73	-	Small: 1007	Middle: 706	Big: 342
Week 74	-	Small: 966	Middle: 681	Big: 356
Week 75	-	Small: 931	Middle: 653	Big: 369
Week 76	-	Small: 902	Middle: 620	Big: 380
Week 77	-	Small: 880	Middle: 586	Big: 389
Week 78	-	Small: 865	Middle: 551	Big: 396
Week 79	-	Small: 856	Middle: 517	Big: 400
Week 80	-	Small: 853	Middle: 484	Big: 402
Week 81	-	Small: 856	Middle: 452	Big: 400
Week 82	-	Small: 864	Middle: 423	Big: 396
Week 83	-	Small: 877	Middle: 396	Big: 390
Week 84	-	Small: 895	Middle: 372	Big: 382
Week 85	-	Small: 918	Middle: 351	Big: 373
Week 86	-	Small: 945	Middle: 333	Big: 362
Week 87	-	Small: 977	Middle: 318	Big: 350
Week 88	-	Small: 1012	Middle: 305	Big: 337
Week 89	-	Small: 1051	Middle: 295	Big: 324
Week 90	-	Small: 1094	Middle: 288	Big: 310
Week 91	-	Small: 1141	Middle: 282	Big: 297
Week 92	-	Small: 1190	Middle: 279	Big: 284
Week 93	-	Small: 1243	Middle: 279	Big: 272
Week 94	-	Small: 1298	Middle: 281	Big: 260
Week 95	-	Small: 1354	Middle: 285	Big: 248
Week 96	-	Small: 1413	Middle: 291	Big: 238
Week 97	-	Small: 1471	Middle: 301	Big: 228
Week 98	-	Small: 1530	Middle: 313	Big: 219
Week 99	-	Small: 1587	Middle: 328	Big: 211
Week 100	-	Small: 1642	Middle: 346	Big: 203