## Assignment #5

For this assignment you will be writing 3 programs which should be saved independently as their own “.py” file. The filename you should use for each program is outlined in the sections below. When you’re finished you should submit your programs to the Assignment #5 category inside of Brightspace.

You may work by yourself or with a partner. Remember, collaborating just entails discussing approaches to the problems; you should not be writing actual code together. If you choose to work with someone else please be sure to give them credit in your source code and on Brightspace when you submit your work. Each member of a team must submit their own unique copy of the assignment to get credit for the assignment.

[Problem 1 |](https://emilydidthis.github.io/CSCI-UA.0002-Fall22/assignments/05.html#problem-1-pizza-party)[Problem 2 |](https://emilydidthis.github.io/CSCI-UA.0002-Fall22/assignments/05.html#problem-2a-prime-number-finder)[Problem 3](https://emilydidthis.github.io/CSCI-UA.0002-Fall22/assignments/05.html#problem-3a-addition-table)

### Problem 1: Pizza Party

You are arranging a pizza party for all of your friends. You want to figure out how much pizza you can order without exceeding your budget. Write a program that accepts the number of people attending your party, the budget for your party, the price per slice of pizza, and the price per pie of pizza. Your program should then identify how many slices each person wants and determine how many individual slices and pies to order without exceeding your budget. If you exceed your budget, the program should warn the user they can’t place their order. Note that for the purpose of this program a pizza pie contains 8 slices. And note that the restaurant you are ordering from will not sell you more than 7 individual slices at a time (if you need to purchase 8 slices you will have to buy a whole pie).

You do not need to validate the first four input statements (total budget, cost per slice, cost per pie and # of people attending party). You do have to validate the user’s input when prompting them for the number of slices for each person coming to the party (i.e. Enter number of slices for person #1).

Example input/output:

Enter budget for your party: 100

Cost per slice of pizza: 2.50

Cost per whole pizza pie (8 slices): 12.50

How many people will be attending your party? 10

Enter number of slices for person #1: -2

Not a valid entry, try again!

Enter number of slices for person #1: 2

Enter number of slices for person #2: 3

Enter number of slices for person #3: 4

Enter number of slices for person #4: 2

Enter number of slices for person #5: 3

Enter number of slices for person #6: 4

Enter number of slices for person #7: 2

Enter number of slices for person #8: 1

Enter number of slices for person #9: 5

Enter number of slices for person #10: 4

You should purchase 3 pies and 6 slices

Your total cost will be: 52.50

You will still have 47.50 left after your order

Enter budget for your party: 10

Cost per slice of pizza: 2.50

Cost per whole pizza pie (8 slices): 12.50

How many people will be attending your party? 4

Enter number of slices for person #1: 1

Enter number of slices for person #2: 1

Enter number of slices for person #3: 1

Enter number of slices for person #4: 1

You should purchase 0 pies and 4 slices

Your total cost will be: 10.00

You will have no money left after your order.

Enter budget for your party: 10

Cost per slice of pizza: 2.50

Cost per whole pizza pie (8 slices): 12.50

How many people will be attending your party? 4

Enter number of slices for person #1: 2

Enter number of slices for person #2: 2

Enter number of slices for person #3: 2

Enter number of slices for person #4: 2

Your order cannot be completed.

You would need to purchase 1 pies and 0 slices

This would put you over budget by 2.50

### Problem 2a: Prime Number Finder

Write a program that prompts the user to enter in a positive number. Only accept positive numbers - if the user supplies a negative number or zero you should re-prompt them.

Next, determine if the given number is a prime number. A prime number is a number that has no positive divisors other than 1 and itself. For example, 5 is prime because the only numbers that evenly divide into 5 are 1 and 5. 6, however, is not prime because 1, 2, 3 and 6 are all divisors of 6.

Here’s a sample running of the program:

Enter a positive number to test: 5

2 is NOT a divisor of 5 ... continuing

3 is NOT a divisor of 5 ... continuing

4 is NOT a divisor of 5 ... continuing

5 is a prime number!

And here’s another running:

Enter a positive number to test: 9

2 is NOT a divisor of 9 ... continuing

3 is a divisor of 9 ... stopping

9 is not a prime number.

Some notes on your program:

* 1 is technically not a prime number.
* Once you find a number that evenly divides into your test number you do not need to continue testing additional numbers - the number cannot be prime.

### Problem 2b: Find all Prime Numbers between 1 and 1000

Next, make a copy of Part A and update it so that the program now finds all prime numbers between 1 and 1000. Here’s a sample running of your program:

1 is technically not a prime number.

2 is a prime number!

3 is a prime number!

5 is a prime number!

7 is a prime number!

11 is a prime number!

... cut ...

977 is a prime number!

983 is a prime number!

991 is a prime number!

997 is a prime number!

### Problem 2c: Custom Number Range

Make a copy of Part B and update it so that the user can choose to examine a specific range of numbers for prime numbers. Here’s a sample running of your program:

Start number: 5

End number: -5

Start and end must be positive

Start number: 5

End number: 3

End number must be greater than start number

Start number: 5

End number: 23

5

7

11

13

17

19

23

Some notes on your program:

* You need to ensure that the start and end numbers are both positive.
* You also need to ensure that the start number is less than the end number.

### Part 2d: Tabular Formatting

Print the prime numbers that you find within the given range so that only 10 numbers print per line. Align the numbers so that they stack neatly on top of one another in all cases (i.e. the table should line up no matter what number range you are analyzing). The table you generate should use the MINIMUM amount of spaces needed to align itself (i.e. if the largest number if your range is “100” then all of your values should be formatted to be exactly 4 characters in size. If the highest value in your range is “1000” then all values should be formatted to be 5 characters in size). Here’s a sample running of the program:

Start number: 1

End number: 100

2 3 5 7 11 13 17 19 23 29

31 37 41 43 47 53 59 61 67 71

73 79 83 89 97

And here’s another running:

Start number: 1

End number: 1000

2 3 5 7 11 13 17 19 23 29

31 37 41 43 47 53 59 61 67 71

73 79 83 89 97 101 103 107 109 113

127 131 137 139 149 151 157 163 167 173

179 181 191 193 197 199 211 223 227 229

233 239 241 251 257 263 269 271 277 281

283 293 307 311 313 317 331 337 347 349

353 359 367 373 379 383 389 397 401 409

419 421 431 433 439 443 449 457 461 463

467 479 487 491 499 503 509 521 523 541

547 557 563 569 571 577 587 593 599 601

607 613 617 619 631 641 643 647 653 659

661 673 677 683 691 701 709 719 727 733

739 743 751 757 761 769 773 787 797 809

811 821 823 827 829 839 853 857 859 863

877 881 883 887 907 911 919 929 937 941

947 953 967 971 977 983 991 997

Start number: 9900

End number: 10100

9901 9907 9923 9929 9931 9941 9949 9967 9973 10007

10009 10037 10039 10061 10067 10069 10079 10091 10093 10099

### Problem 3a: Addition Table

Write a program that prompts the user for two integers that are greater than or equal to zero. Ensure that the first integer is less than the second integer. Next, write a program that generates an “addition table” using these numbers that computes the sums of all possible values - use the output below as a guide:

Lowest number: -5

Lowest number must be 0 or greater

Lowest number: 0

Highest number: 0

Highest number must be larger than lowest number!

Highest number: 5

+ 0 1 2 3 4 5

----------------------------

0 | 0 1 2 3 4 5

1 | 1 2 3 4 5 6

2 | 2 3 4 5 6 7

3 | 3 4 5 6 7 8

4 | 4 5 6 7 8 9

5 | 5 6 7 8 9 10

Here are some hints to get you started:

* Don’t worry about formatting your table until you have a basic version printing out correctly.
* Start by generating the first line of output (the ‘+’ sign and the column headings)
* Next, generate the left most column (the row headings)
* Big hint: Python prints from top to bottom, left to right. Before moving onto the next row, what do you have to do? Do you need another loop here?
* When you’re ready to format your table you need to ensure that no matter what there will always be room for all of your numbers.

Hint: you might need to count the size of your largest possible value and use that as part of the “format” function. For example, notice how the spacing in the following tables always works out no matter the size of the values being printed:

Lowest number: 9999

Highest number: 10004

+ 9999 10000 10001 10002 10003 10004

-------------------------------------------------

9999 | 19998 19999 20000 20001 20002 20003

10000 | 19999 20000 20001 20002 20003 20004

10001 | 20000 20001 20002 20003 20004 20005

10002 | 20001 20002 20003 20004 20005 20006

10003 | 20002 20003 20004 20005 20006 20007

10004 | 20003 20004 20005 20006 20007 20008

Lowest number: 9999999

Highest number: 10000004

+ 9999999 10000000 10000001 10000002 10000003 10000004

----------------------------------------------------------------------

9999999 | 19999998 19999999 20000000 20000001 20000002 20000003

10000000 | 19999999 20000000 20000001 20000002 20000003 20000004

10000001 | 20000000 20000001 20000002 20000003 20000004 20000005

10000002 | 20000001 20000002 20000003 20000004 20000005 20000006

10000003 | 20000002 20000003 20000004 20000005 20000006 20000007

10000004 | 20000003 20000004 20000005 20000006 20000007 20000008

### Part 3b: Addition Table

Next, add in a feature that asks the user if they want to identify ‘Prime’ numbers in their table. If the user elects to show prime numbers you can print a lowercase ‘p’ character after each prime number. Ensure that your table displays correctly, as described above.

Lowest number: 0

Highest number: 10

Would you like to identify Prime numbers in your table? (y/n): pikachu

Invalid command, try again

Would you like to identify Prime numbers in your table? (y/n): y

+ 0 1 2 3 4 5 6 7 8 9 10

------------------------------------------------

0 | 0 1 2p 3p 4 5p 6 7p 8 9 10

1 | 1 2p 3p 4 5p 6 7p 8 9 10 11p

2 | 2p 3p 4 5p 6 7p 8 9 10 11p 12

3 | 3p 4 5p 6 7p 8 9 10 11p 12 13p

4 | 4 5p 6 7p 8 9 10 11p 12 13p 14

5 | 5p 6 7p 8 9 10 11p 12 13p 14 15

6 | 6 7p 8 9 10 11p 12 13p 14 15 16

7 | 7p 8 9 10 11p 12 13p 14 15 16 17p

8 | 8 9 10 11p 12 13p 14 15 16 17p 18

9 | 9 10 11p 12 13p 14 15 16 17p 18 19p

10 | 10 11p 12 13p 14 15 16 17p 18 19p 20

### Part 4c: Addition Table (Extra Credit)

Expand your program to support ALL of the arithmetic operators (+, -, \*, /, // and %) - prompt the user for an operator to use and then display the desired table. Validate your data accordingly. Ensure that your tables print out using the expected formatting. Note that negative numbers are not considered Prime for the purpose of this part of the assignment.