



CMSC 201 Section 60

Homework 4 - Looping Again

Due Date: Monday, October 7th, 2024 by 11:59:59 PM

Value: 40 points

This assignment falls under the standard cmssc201 academic integrity policy. This means you should not discuss/show/copy/distribute your solutions, your code or your main ideas for the solutions to any other student. Also, you should not post these problems on any forum, internet solutions website, etc.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
"""
File:      FILENAME.py
Author:    YOUR NAME
Date:      THE DATE
Section:   YOUR DISCUSSION SECTION NUMBER
E-mail:    YOUR_EMAIL@umbc.edu
Description:
    DESCRIPTION OF WHAT THE PROGRAM DOES
"""
```

Creating Your HW4 Directory

```
linux3[1]% cd cmsc201/Homeworks
linux3[2]% mkdir hw4
linux3[3]% cd hw4
linux3[4]% █
```

Submission Details

Submit the files under the following titles:

(These are case sensitive as usual.)

submit cmsc201 HW4 {files go here}

Problem 1 - Fibonacci	reach_fib.py
Problem 2 - Grocery List	grocery_list.py
Problem 3 - Number Guesser	number_guesser.py
Problem 4 - Two Bins	two_bins.py
Problem 5 - Reverse without Numbers	list_reverse.py

For example you would do:

```
linux1[4]% submit cmsc201 HW4 reach_fib.py grocery_list.py
number_guesser.py two_bins.py list_reverse.py
Submitting reach_fib.py...OK
Submitting grocery_list.py...OK
Submitting number_guesser.py...OK
Submitting two_bins.py...OK
Submitting list_reverse.py...OK
linux1[5]% █
```

From here, you can use **emacs** to start creating and writing your different Homework 4 Python programs.

You don't need to and should not make a separate folder for each file. You should store all of the Homework 4 files in the same **hw4** folder.

Coding Standards

Coding standards for CMSC 201 can be [found here](#).

For now, you should pay special attention to the sections about:

- Naming Conventions
- Use of Whitespace
- Comments (specifically, File Header Comments)
- Variable names.

Use `if __name__ == '__main__':`

Make sure to include `if __name__ == '__main__':` under your header comment of each file and indent your code inside it. (This is a code cleanliness practice, it means that people using your Python file will only run your code if they intend to. It will make more sense when we talk about functions.)

Input Validation

For this assignment, you do **not** need to worry about **SOME** “input validation.”

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

Unlike in HW1 you didn't have to worry about any input validation since you didn't have if statements, but now you do, so you can worry a little about it. You can try to prevent invalid input, but only when that invalid input is of the correct type.

For example, if your program asks the user to enter a whole number, it is acceptable if your program crashes if they enter something else like “dog” or “twenty” or “88.2” instead.

But it's a good idea to try to catch a zero division error for instance, or entering negative numbers when only positive numbers are allowed.

Here is what that error might look like:

```
Please enter a number: twenty
Traceback (most recent call last):
  File "test_file.py", line 10, in <module>
    num = int(input("Please enter a number: "))
ValueError: invalid literal for int() with base 10: 'twenty'
```

Allowed Built-ins/Methods/etc

- Declaring and assigning variables, ints, floats, bools, strings.
- Casting `int(x)`, `str(x)`, `float(x)`, (technically `bool(x)`)
- Using `+`, `-`, `*`, `/`, `//`, `%`, `**`; `+=`, `-=`, `*=`, `/=`, `//=`, `%=`, `**=` where appropriate
- Print, with string formatting, with `end=` or `sep=`:
 - `'{}'.format(var)`, `'%d' % some_int`, f-strings
 - Really the point is that we don't care how you format strings in Python
 - `Ord`, `chr`, but you won't need them this time.
- Input, again with string formatting in the prompt, casting the returned value.
- Using the functions provided to you in the starter code.
- Comparisons `==`, `<=`, `>=`, `>`, `<`, `!=`, `in`
- Logical `and`, `or`, `not`
- `if/elif/else`, nested if statements
- Using `import` with libraries and specific functions **as allowed** by the project/homework.
- String Operations:
 - `upper()`, `lower()`, `join()`, `strip()`, `split()`
 - concatenation `+`, `+=`
- For loops, both *for i* and *for each* type.
- Lists, `list()`, indexing, i.e. `my_list[i]` or `my_list[3]`
 - 2d-lists if you want them/need them `my_2d[i][j]`
 - Append, remove
- `len()`
- While loops
 - sentinel values, boolean flags to terminate while loops

Forbidden Built-ins/Methods/etc

This is not a complete listing, but it includes:

- list slicing
- Declaring your own Functions
- Dictionaries
 - creation using `dict()`, or `{}`, copying using `dict(other_dict)`
 - `.get(value, not_found_value)` method
 - accessing, inserting elements, removing elements.
- `break`, `continue`
- methods outside those permitted within allowed types
 - for instance `str.endswith`
 - `list.index`, `list.count`, etc.
- Keywords you definitely don't need: `await`, `as`, `assert`, `async`, `class`, `except`, `finally`, `global`, `lambda`, `nonlocal`, `raise`, `try`, `yield`
- The `is` keyword is forbidden, not because it's necessarily bad, but because it doesn't behave as you might expect (it's not the same as `==`).
- built in functions: `any`, `all`, `breakpoint`, `callable`, `classmethod`, `compile`, `exec`, `delattr`, `divmod`, `enumerate`, `filter`, `map`, `max`, `min`, `isinstance`, `issubclass`, `iter`, `locals`, `oct`, `next`, `memoryview`, `property`, `repr`, `reversed`, `round`, `set`, `setattr`, `sorted`, `staticmethod`, `sum`, `super`, `type`, `vars`, `zip`
- If you have read this section, then you know the secret word is: `argumentative`.
- `exit()` or `quit()`
- If something is not on the allowed list, not on this list, then it is probably forbidden.
- The forbidden list can always be overridden by a particular problem, so if a problem allows something on this list, then it is allowed for that problem.

Problem 1 - Fibonacci

The Fibonacci sequence is defined in the following way:

Start out with $F_0 = 1$ and $F_1 = 1$. To calculate the n^{th} Fibonacci number, where $n > 1$, add up the two previous numbers. Mathematically we'd write:

$$F_n = F_{n-1} + F_{n-2}$$

For some examples,

$$F_2 = 1 + 1 = 2, F_3 = 2 + 1 = 3, F_4 = 3 + 2 = 5, F_5 = 5 + 3 = 8$$

Write a program which:

- 1) Inputs an integer (N).
- 2) Find the Fibonacci number that is greater than or equal to the entered number.

For instance, when $N = 31$, $F_8 = 34$, so we should output:

$$F_8 = 34 \text{ which is greater than or equal to } 31$$

When $N = 233$, we should print:

$$F_{12} = 233 \text{ which is greater than or equal to } 233$$

For instance when $N = 1$, we should print:

$$F_0 = 1 \text{ which is greater than or equal to } 1$$

Sample Output:

```
linux[34]$ python3 reach_fib.py
What number should we exceed? 1
F0 = 1 which is greater than or equal to 1

linux[35]$ python3 reach_fib.py
What number should we exceed? 123
F11 = 144 which is greater than or equal to 123

linux[36]$ python3 reach_fib.py
What number should we exceed? 71526
F24 = 75025 which is greater than or equal to 71526

linux[37]$ python3 reach_fib.py
What number should we exceed? 3
F3 = 3 which is greater than or equal to 3

linux[38]$ python3 reach_fib.py
What number should we exceed? 531
F14 = 610 which is greater than or equal to 531

linux[38]$ python3 reach_fib.py
What number should we exceed? 12345
F21 = 17711 which is greater than or equal to 12345
```




Problem 2 - Grocery List

For this problem you are going to go grocery shopping. You need eggs, bread, butter, and milk, of course. So, keep a list of all of the things that you pick up in the grocery store.

At the end, you should print out what you bought and also print out if you bought all of the required items, or if you still need something.

The required items are eggs, bread, butter, and milk.

Call the file `grocery_list.py`.

```
linux[132]$ python3 grocery_list.py
What do you pick up in the store? (quit to exit) milk
What do you pick up in the store? (quit to exit) cake
What do you pick up in the store? (quit to exit) pie
What do you pick up in the store? (quit to exit) eggs
What do you pick up in the store? (quit to exit) panko
What do you pick up in the store? (quit to exit) quit
You still need bread, butter
You bought milk, cake, pie, eggs, panko

linux[133]$ python3 grocery_list.py
What do you pick up in the store? (quit to exit) quit
You still need eggs, milk, bread, butter
You bought

linux[134]$ python3 grocery_list.py
What do you pick up in the store? (quit to exit) bread
What do you pick up in the store? (quit to exit) milk
What do you pick up in the store? (quit to exit) eggs
What do you pick up in the store? (quit to exit) butter
What do you pick up in the store? (quit to exit) tomato
sauce
What do you pick up in the store? (quit to exit) ramen
What do you pick up in the store? (quit to exit) chicken
What do you pick up in the store? (quit to exit) quit
You have everything you need.
You bought bread, milk, eggs, butter, tomato sauce, ramen,
chicken
```

Problem 3 - Number Guesser

You must implement a number guesser. Then you should ask the user for guesses until they guess the number.

If the number is too big, tell them that.

If the number is too small, tell them that.

You should count the number of steps it took, and tell the user at the end.

You will need to import the following modules under your header comment:

```
import sys
from random import randint, seed
```

For testing purposes, you must have this bit of code above your if `__name__ == '__main__'` block:

```
if len(sys.argv) >= 2:
    seed(sys.argv[1])
```

To use `randint`, you should specify the lower and upper bound (inclusive) in this case:

```
randint(1, 100)
```

Here is some sample output for rock_paper.py, with the user input in **blue**.

```
linux[0]$ python3 number_guesser.py
Guess a number between 1 and 100: 50
Your guess is too low.
Guess a number between 1 and 100: 72
Your guess is too high.
Guess a number between 1 and 100: 65
Your guess is too high.
Guess a number between 1 and 100: 64
Your guess is too high.
Guess a number between 1 and 100: 62
Your guess is too high.
Guess a number between 1 and 100: 57
You guessed the value! It took you 6 steps.
```

```
linux[1]$ python3 number_guesser.py
Guess a number between 1 and 100: 75
Your guess is too low.
Guess a number between 1 and 100: 80
Your guess is too low.
Guess a number between 1 and 100: 85
Your guess is too low.
Guess a number between 1 and 100: 90
You guessed the value! It took you 4 steps.
```

```
linux[2]$ python3 number_guesser.py
Guess a number between 1 and 100: 30
Your guess is too low.
Guess a number between 1 and 100: 70
Your guess is too high.
Guess a number between 1 and 100: 50
Your guess is too low.
Guess a number between 1 and 100: 60
Your guess is too high.
Guess a number between 1 and 100: 55
You guessed the value! It took you 5 steps.
```

Problem 4 - Two Bins

For this problem you want to keep track of the items in two bins under the following operations:

- 1) add [A or B] item

Add an item to either bin A or bin B.

- 2) remove [A or B] item

From A or B remove the item named "item". If there are multiple, only remove the first.

If the item doesn't exist in the list make sure to tell the user rather than generating an IndexError.

- 3) display [A or B]

Display all the elements in bin A or B.

- 4) transfer [A or B]

Transfer the first element of A to B or B to A depending on which list is selected first. The first element would be whatever is in the index 0 position of the list. "transfer A" would transfer from A to B.

Ensure the list is non-empty before attempting a transfer.

Whenever I write [A or B] I mean the string "A" or "B" but no brackets and no word or. Look at the sample output for clarity on that.

You'll have to use string methods like split and join to make the sample output as shown.

Sample output:

User input is generally colored **blue** to help distinguish it from the rest of the text.

```
linux4[116]% python3 two_bins.py
```

```
>>> add A 1
>>> add A 2
>>> add A 3
>>> add B 1
>>> transfer A
>>> display B
Bin B Contents: 1, 1
>>> remove A 2
>>> display A
Bin A Contents: 3
>>> quit
```

```
linux4[117]% python3 two_bins.py
```

```
>>> add A hello
>>> add B goodbye
>>> add A robot
>>> add B something
>>> add A somethingelse
>>> display A
Bin A Contents: hello, robot, somethingelse
>>> display B
Bin B Contents: goodbye, something
>>> transfer B
>>> transfer B
>>> display B
Bin B Contents:
>>> display A
Bin A Contents: hello, robot, somethingelse, goodbye,
something
>>> quit
```

Problem 5 - Reverse without Numbers

Here your objective will be to take in a string list separated by commas, and then convert it into a list using `split(',')` which will split on the commas.

After doing that, you should reverse the list, with the following exception: throw out any strings that contain a digit ('0' through '9').

So for example, if we input "apple, 2banana, chestnut, 3dog4, epsilon5, fast, 123456" we expect the output "fast, chestnut, apple".

To output the new list, use the join function on your newly created list. Join takes the separator first, so if you want to put commas and spacing back, you would do `",[space]".join(my_new_list)`.

Restrictions

- 1) Do not use `reverse()`
- 2) Do not use `[::-1]`
- 3) Do not use negative indices, i.e. accessing any index `my_list[x]` where $x < 0$. You are permitted to use subtraction within the indices as long as it stays positive.

If you end up with some extra spaces, we won't judge that. For instance if you enter "apple, kitten" and the reversed list comes out as `"[space]kitten,[space][space]apple"` with an extra space up in the front, that's ok, don't worry about it.

Call the file `list_reverse.py`.

```
linux[1729]$ python3 list_reverse.py
Enter a list separated by commas:
apple,orange,banana,pear, mango
mango, pear, banana, orange, apple

linux[1730]$ python3 list_reverse.py
Enter a list separated by commas: taco2, banana7, song,
cat, sword4, tumeric
tumeric, cat, song

linux[1731]$ python3 list_reverse.py
Enter a list separated by commas: a, b, c, d, e, 2
e, d, c, b, a

linux[1732]$ python3 list_reverse.py
Enter a list separated by commas: 1, 2, 3
The new list was empty
```