Assignment-3

By: Alexander
UMSL

Coding Journey Summary:

- ⏱️ 2 hours 21 minutes 17 seconds of active typing
- 📅 3 days 1 hour 2 minutes 31 seconds total duration
- 🔢 9,463 characters | 247 lines
- 📈 Typing speed: 2 character(s) per second
- 🗄️ Process Data: Showing 267 changes made every 5+ seconds (of 800 changes total)
- 📊 Interactive version of this report is AVAILABLE HERE.

Final Code

If the code contains more than one file, they are concatenated and shown.

```
==+==
FILE: Main.py
==+==

# Imports
from random import randrange
from guess import number_guessing_game
from cows import cows
from stack import stack
from cubes import cubes
from invoice import invoice
from order import order

# constants
BDAY = 22
d = 2 + (BDAY %2)
m = 1 +(BDAY // 2)
random_ints = 2 * d + m

number_guessing_game(d)
cows(d)
stack()
cubes(random_ints)
invoice()
order()


==+==
FILE: guess.py
==+==

from random import randrange
import math

def number_guessing_game(d):
        """ Game asks user to enter a range and then guess a number within that range.
        """
        # variables to keep track when a user wins or computer wins
        user_wins = 0
        computer_wins = 0
        # keep track which round the user is in
        games_played = 0
        max_guesses_to_win = 0
```

```
        min_guesses_to_win = 1000
        average_guesses_to_win = 0
        average_guesses_per_round = 0
        # append amount of times a user guessed to the list after each round is over
        total_guesses_per_round = []
        total_guesses_per_winning_round = []
        # ask user if they want to play
        is_game_active = input("Would you like to play a number guessing game?\n(y/n) Enter -999 to
end once playing.").lower()
        # begin game
        while is_game_active == 'y':
                games_played += 1
                # collect user input
                user_input = input("Enter a lower bound (inclusive) and upper bound (exclusive)
separated by a space. -999 to end\n").split()
                if has_user_ended_game(int(user_input[0])):
                        break
                # get bounds from split string delimmiter is space
                lower_bound = int(user_input[0])
                upper_bound = int(user_input[1])
                # check if lower bound is -999
                # if true break and end game
                if has_user_ended_game(lower_bound, upper_bound):
                        break

                # generate number for user to guess
                number_to_guess = randrange(lower_bound, upper_bound)
                # number of times the user is allowed to guess before they lose
                times_user_can_guess = math.ceil(math.log(upper_bound - lower_bound + 1, d))
                # how many guesses they have left
                guesses_left = times_user_can_guess
                guesses_made = 0

                while guesses_left > 0:
                        user_guess = int(input(f"Enter a number between your range to guess. Guesses
left {guesses_left}\n"))
                        guesses_made += 1
                        # check to see if they've ended the game here
                        if has_user_ended_game(user_guess):
                                is_game_active = 'n'
                                break

                        guesses_left -= 1
                        if user_guess == number_to_guess:
                                print("you win this round\n")
                                if guesses_made > max_guesses_to_win:
                                        max_guesses_to_win = guesses_made
                                if guesses_made < min_guesses_to_win:
                                        min_guesses_to_win = guesses_made
                                user_wins += 1
                                total_guesses_per_winning_round.append(guesses_made)
                                guesses_left = 0
                        elif guesses_left == 0:
                                print("Out of guesses. Pay me for more. JK. Computer wins\n")
                                total_guesses_per_round.append(guesses_made)
                                computer_wins += 1
                                guesses_left = 0

        print(f'amount of games played {games_played}')
        print(f'computer wins {computer_wins}')
        print(f'user wins: {user_wins}')
        print(f'max guesses for user win: {max_guesses_to_win}')
        print(f'min guesses for user win: {min_guesses_to_win}')
        print(f'Average guesses in user win: {calculate_average(total_guesses_per_winning_round)}')
        print(f'Average guesses per game {calculate_average(total_guesses_per_round)}')



def has_user_ended_game(*args):
        """Checks if user input is -999 to end the game
        args -> int
        """
        for arg in args:
                print(arg)
```

```
                if arg == -999:
                        return True
        return False

def calculate_average(l):
        """checks average of the list passed in
        l - list of int
        """
        total = 0
        for num in l:
                total += num
        return total / len(l) if len(l) > 0 else 0

==+==
FILE: cows.py
==+==

from random import randrange
import math

user_prompt_text = "Enter a number for cows and bulls. -999 to end\n"

def cows(d):
        user_guess = 1000
        # variables to keep track when a user wins or computer wins
        user_wins = 0
        computer_wins = 0
        games_played = 0
        average_guesses_to_win = 0
        average_guesses_per_round = 0
        max_guesses_to_win = 0
        min_guesses_to_win = 1000
        total_guesses_per_round = []
        total_guesses_per_winning_round = []
        while user_guess != -999:
                # append amount of times a user guessed to the list after each round is over

                times_user_can_guess = math.ceil(math.log(10 + 1, d))
                guesses_made = 0
                guesses_left = times_user_can_guess
                random_number = generate_random_number()

                while guesses_made < times_user_can_guess:
                        print(f"Guesses remaining: {guesses_left}")
                        user_guess = int(input(user_prompt_text))
                        if user_guess == -999:
                                break

                        guesses_left -= 1
                        guesses_made += 1
                        user_number_list = convert_user_number_to_list(user_guess)
                        print(user_number_list)
                        print(random_number)
                        bulls_and_cows = compare_lists(user_number_list, random_number)
                        print_bulls_and_cows(bulls_and_cows)
                        if check_for_win(bulls_and_cows):
                                print("User wins")
                                user_wins += 1
                                total_guesses_per_winning_round.append(guesses_made)
                                # track guesses per overall round regardless of win or lose
                                total_guesses_per_round.append(guesses_made)
                                games_played += 1
                                if guesses_made > max_guesses_to_win:
                                        max_guesses_to_win = guesses_made
                                if guesses_made < min_guesses_to_win:
                                        min_guesses_to_win = guesses_made
                                break
                                # verify user lost
                        if guesses_made == times_user_can_guess:
                                print("Computer wins")
                                computer_wins += 1
                                # track guesses per overall round regardless of win or lose
                                total_guesses_per_round.append(guesses_made)
                                games_played += 1
```

```
        print(total_guesses_per_round)
        print(f'amount of games played {games_played}')
        print(f'computer wins {computer_wins}')
        print(f'user wins: {user_wins}')
        print(f'max guesses for user win: {max_guesses_to_win}')
        print(f'min guesses for user win: {min_guesses_to_win}')
        print(f'Average guesses in user win: {calculate_average(total_guesses_per_winning_round)}')
        print(f'Average guesses per game {calculate_average(total_guesses_per_round)}')




def generate_random_number():
        """return a list to track position of random 4 digit number"""
        number_list = []
        while len(number_list) < 4:
                r = randrange(0, 10)
                if r not in number_list:
                        number_list.append(r)
        return number_list

def convert_user_number_to_list(user_n):
        number_list = [0, 0, 0, 0]
        for x in range(3, -1, -1):
                remainder = user_n % 10
                user_n = user_n // 10
                number_list[x] = remainder
        return number_list

def compare_lists(user_list, computer_list):
        """Compare the two lists for position and digit"""
        bulls_and_cows = []
        for idx, item in enumerate(user_list):
                if item == computer_list[idx]:
                        bulls_and_cows.insert(idx, "Bull")
                elif item in computer_list:
                        bulls_and_cows.insert(idx, "Cow")
        return bulls_and_cows

def check_for_win(l):
        return l.count("Bull") == 4

def print_bulls_and_cows(b_and_c):
        for item in b_and_c:
                print(item)

def calculate_average(l):
        """checks average of the list passed in
        l - list of int
        """
        total = 0
        for num in l:
                total += num
        return total / len(l) if len(l) > 0 else 0

==+==
FILE: stack.py
==+==

from statistics import median, mean, mode
def stack():
        user_input = 0
        stack = []
        while user_input != -999:
                user_input = int(input("Enter numbers for the stack\n"))
                if user_input != -999:
                        stack.append(user_input)

        print(stack)
        print(f'mean is {mean(stack)}')
        print(f'median is {median(stack)}')
        print(f'mode is {mode(stack)}')

--+--
```

```
==+==
FILE: cubes.py
==+==

from random import randint
def cubes(rand_ints):
        rands = [randint(0, 9) for _ in range(0, rand_ints)]
        for num, cube_num in get_cubes(rands):
                print(f'{num}:{cube_num}')

def get_cubes(l):
        cubes_list = [x ** 3 for x in l]
        return zip(l, cubes_list)


==+==
FILE: invoice.py
==+==

from decimal import Decimal
from operator import getitem

def invoice():
        invoice_list = [(83, 'Electric Sander', 7, Decimal(57.98)), (24, 'Power Saw', 18,
Decimal(99.99)), (7, 'Sledge Hammer', 11, Decimal(21.50)), (77, 'Hammer', 76, Decimal(11.99)), (39,
"Jig Saw", 3, Decimal(79.50))]
        print(f'sorted by product description {sorted(invoice_list, key=lambda item: item[1])}')
        print(f'sorted by price {sorted(invoice_list, key=lambda item: item[3])}')
        print(f'tuple map to tuple part desc and quantity sorted {sorted(list(map(lambda x: (x[1],
x[2]), invoice_list)), key=lambda item: item[1])}')
        print(f'tuple map tuple part desc and value sorted {sorted(list(map(lambda x: (x[1], x[2] *
x[3]), invoice_list)), key=lambda item: item[1])}')
        # readability is ZERO but... this creates the mapped tuples then filters them then sorts it
based on invoice value. That is all. Power to the lambda
        print(f'filtered {sorted(list(filter(lambda x: x[1] >= 200 and x[1] <= 500, list(map(lambda
x: (x[1], x[2] * x[3]), invoice_list)))), key=lambda item: item[1])}')
        total = 0
        for x in invoice_list:
                total += x[3]
        print(f'Total of all invoices {total:.2f}')


==+==
FILE: order.py
==+==

def order():
        """Explanation
        filter first filters items out so it will change the amount of items in each list that are
making it to the square map lambda
        map first squares all the elements which creates more elements that will be divisble evenly
by 4 resulting in a bigger final list
        """
        primary_list = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]
        mapped_first = list(filter(lambda x: x % 4 == 0, list(map(lambda x: x**2, primary_list))))
        filter_first = list(map(lambda x: x**2, list(filter(lambda x: x % 4 == 0, primary_list))))
        print(f'primary list {primary_list}')
        print(f'Mapped first {mapped_first}')
        print(f'Filter first {filter_first}')
```

Final Execution Output

Output of the most recent code execution.

```
primary list [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]
Mapped first [100, 16, 4, 64, 36]
Filter first [16, 64]
```

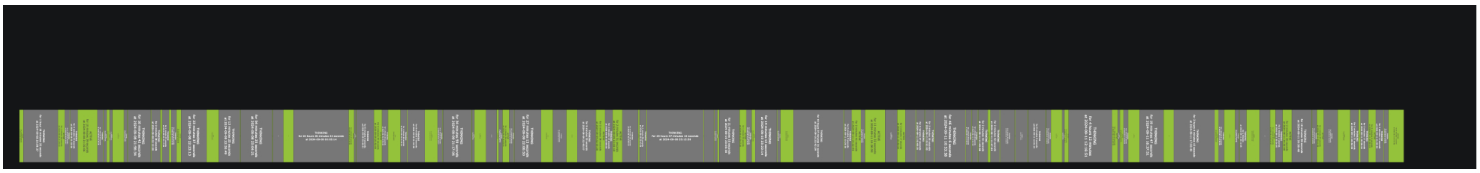
Most Common Error

The most common error that occured during the coding process; "null" (if any) indicates no errors.

```
Traceback (most recent call last):
  File "Main.py", line 10, in <module>
    number_guessing_game(d)
  File "/usr/src/app/guess.py", line 30, in number_guessing_game
    upper_bound = int(user_input[1])
                      ~~~~~~~~~~~^^^
IndexError: list index out of range
```

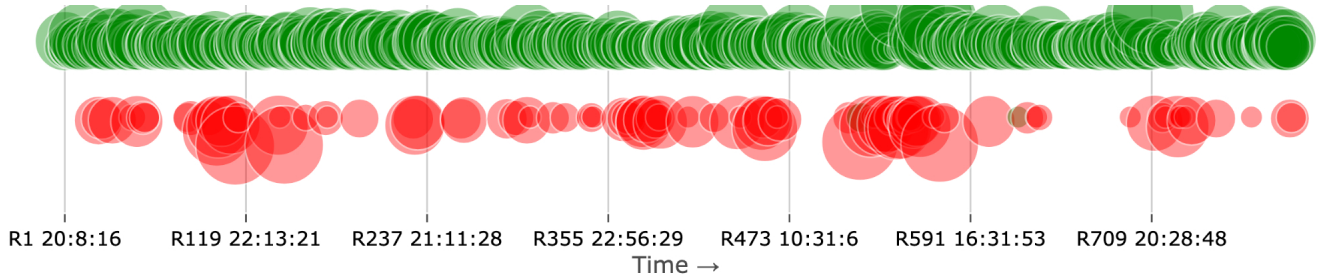
Active Sessions

Active typing sessions while working on the task, shown as green bars.



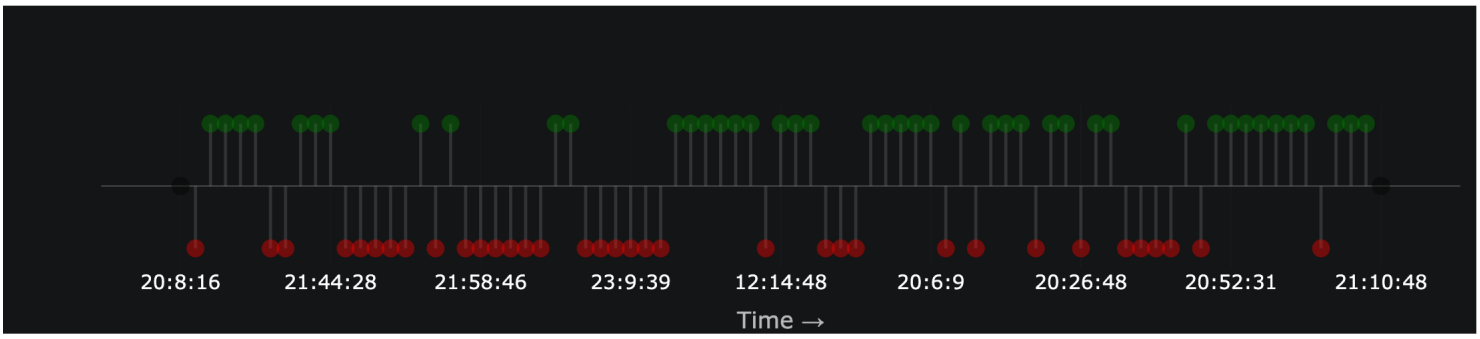
Text Added or Removed Over Time

Green and red circles show text add and removed; size of the circle shows change in the number of characters.



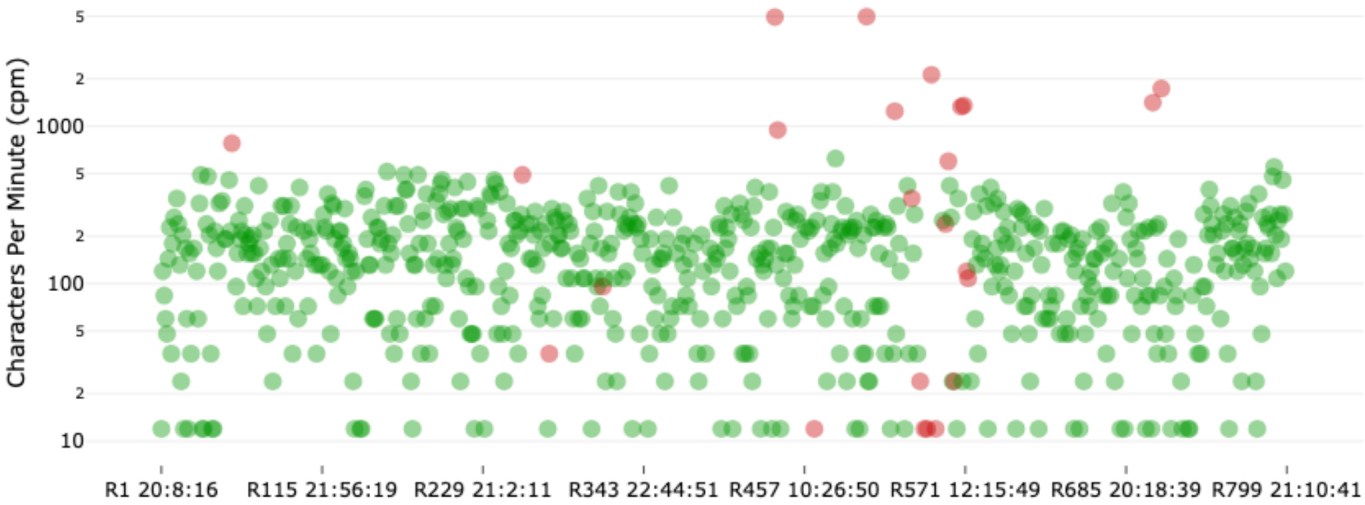
Code Executions

Green and red dots correspond to successful and failed executions.

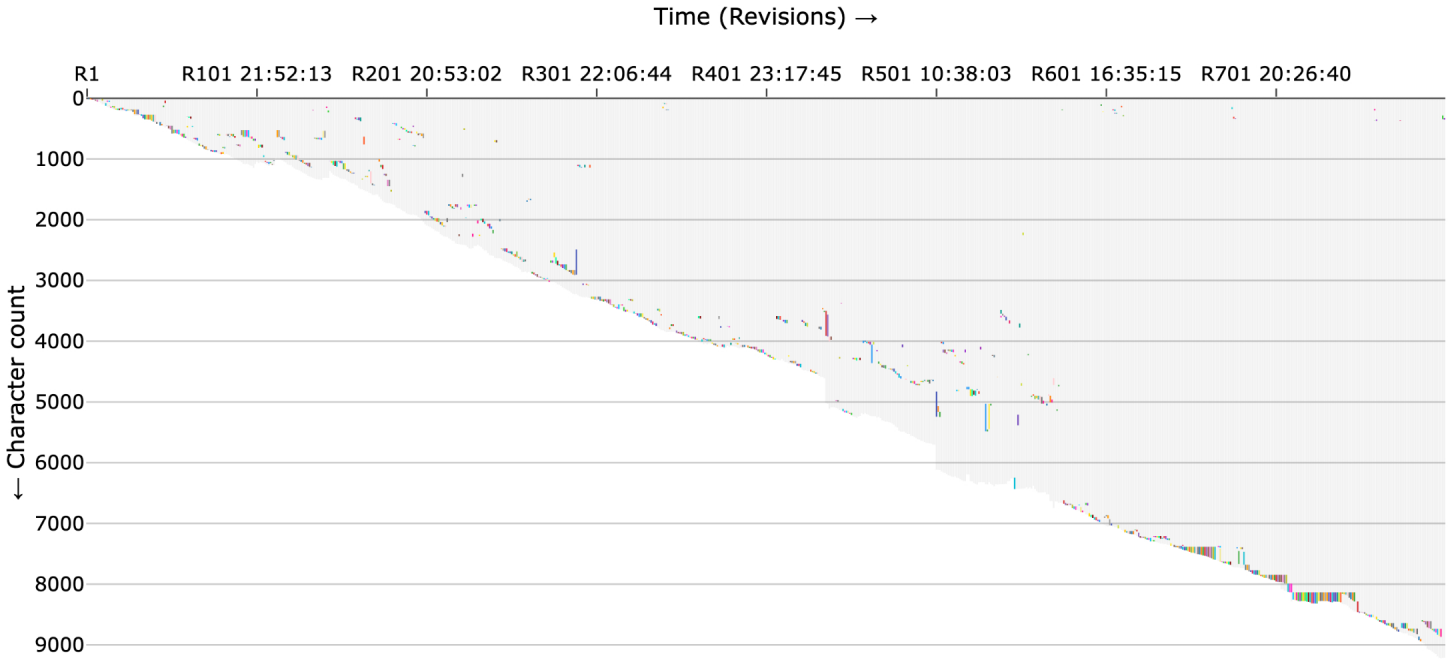


Typing Speed

Typing speed can be negative when text is deleted.



Time →

Edited Line Locations

Each edited line block is assigned a color; highlighted bars show the line(s) edited at that revision.