Assignment-4

By: Alexander
UMSL

Coding Journey Summary:

⏱️ 1 hour 10 minutes 49 seconds of active typing
📅 2 days 4 hours 12 minutes 15 seconds total duration
🔢 5,684 characters | 152 lines
📈 Typing speed: 2 character(s) per second
🗄️ Process Data: Showing 229 changes made every 5+ seconds (of 461 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
==+==

# Sample Python code
import numpy as np
import pandas as pd
import copy
from phonebook import phonebook
from ingredients import ingredients

# constants
BDAY = 22
D = 2 + (BDAY % 2)
M = 1 + (BDAY // 2)

# Q1
"""
mutable objects in python inlcudes things like lists, dicts, and sets
whereas immutable are pythons built-in types like string, int, etc..
"""
some_tuple = (4, 10)
# this will throw TypeError - does not support item assignment
# some_tuple[0] = (5, 9)
some_string = "hello"
# str is also immutable. This line below will result in another TypeError 'str' does not support
item assignment
# some_string[0] = "b"
some_list = [1,2,3]
print(some_list)
# here appending adds / mutates the list some_list
some_list.append(4)
```

```
print(some_list)


# Q2
"""
Data structures incldue lists, sets, dicts, tuples (source -
https://docs.python.org/3/tutorial/datastructures.html)
"""
# lists can contain different data types but usually are homogenous
another_list = ["4", 3]
print(another_list)
# lists can also be used like stacks and queues (although shouldn't be used as a queue since
performance is bad)
# like a stack items are appended and popped from the end in LIFO
another_list.append(4)
# ['4', 3, 4]
print(another_list)
another_list.pop()
# int 4 is popped and list now prints ['4', 3]
print(another_list)

# sets follow the mathematical set in which duplicates are automatically removed
some_set = set('aaabbbbc')
# just prints unique letters 'a', 'b', 'c'
print(some_set)
# one advantage is easily removing duplicates from a list
remove_dups_list = list(set([1,1,2,3,4,4]))
# prints list [1,2,3,4]
print(remove_dups_list)
# dicts are great for key:value associations
some_dict = {"key": "value"}
print(some_dict)
# keys can be any immutable data type
some_dict = {2: "value"}
print(some_dict)

# tuple is good for a collection of heterogenous items. As shown above they are immutable
some_other_tuple = (1, "hello", 13.95)
print(some_other_tuple)
# they can easily be unpacked
quantity, description, price = some_other_tuple
# prints 1, hello, 13.95
print(quantity, description, price)


# Q3
shallow_list1 = [1,2, [4, 5]]
shallow_copy = copy.copy(shallow_list1)

print("Original List items")
print(shallow_list1)

shallow_copy[2][0] = 8
print("original list after modifications to shallow copied list")
print(shallow_list1)
"""
A shallow copy constructs a new compound object and then (to the extent possible) inserts references
into it to the objects found in the original.
```

```
into it to the objects found in the original.
The key here being "references" meaning the copy still has references to the original object so
changing will still affect the original
source: https://docs.python.org/3/library/copy.html
"""

deep_list = [1,2, [4,5]]
deep_copy = copy.deepcopy(deep_list)\

print("Original list items")
print(deep_list)

deep_copy[2][0] = 9
print("Original list after modifications made to deep copy")
print(deep_list)
print("Deep copied list")
print(deep_copy)
"""
A deep copy constructs a new compound object and then, recursively, inserts copies into it of the
objects found in the original.
The key here is that deep copy doesn't insert references but copies of them
source: https://docs.python.org/3/library/copy.html
"""

# Q4
# Queue is FIFO so first in must be first out
user_input = input("Enter items separated by space to place in queue\n").split()
user_input_list = [int(x) for x in user_input]
print(user_input_list)
# popping at the first index of a list implements a FIFO / queue
user_input_list.pop(0)
user_input_list.pop(0)
print(user_input_list)


# Q5
repeated_list = [3,3,4,5,6,4,2,3,4,3,5,6]
# use the key as a value of the list and value should be its frequency
dict_counter = {}
for item in repeated_list:
    if item in dict_counter:
        dict_counter[item] += 1
    else:
        dict_counter[item] = 1

# max_freq will hold the number which occurs the most
max_freq = 0
for k,v in dict_counter.items():
    if v > max_freq:
        max_freq = v
        max_freq_key = k

indicies_of_repeated_value = [idx for idx, val in enumerate(repeated_list) if val == max_freq_key]
print(indicies_of_repeated_value)


# Q7
phonebook(M)
```

```
# 6
ingredients()

==+==
FILE: phonebook.py
==+==

def phonebook(m):
    choice = input("Add a new contact or finish\n").lower()
    amount_of_friends_entered = 0
    user_input = []
    digits_list = []
    while choice == 'add a new contact' or amount_of_friends_entered <= m:
        friend = input("Enter your friends last name\n").lower()
        if friend == 'finish':
            break
        user_input.append(friend)
        digits = int(input("Enter your friends number\n"))
        digits_list.append(digits)
        amount_of_friends_entered += 1

    contacts = {k:v for k,v in zip(user_input, digits_list)}
    dict_keys = list(contacts.keys())
    dict_keys.sort(reverse=True)
    sorted_by_keys = {i: contacts[i] for i in dict_keys}
    sorted_by_values = {k: v for k, v in sorted(contacts.items(), key=lambda item: item[1])}

    print(f'sort descending by keys: {sorted_by_keys}')
    print(f'sort ascendign by values: {sorted_by_values}')


==+==
FILE: ingredients.py
==+==

def ingredients():
    fake_recipe_ingredients = ["flour", "salt", "pepper", "milk", "lettuce", "butter", "popcorn",
"paprika", "cream", "turkey"]
    healthy_stuff = ["not flour", "not salt", "less pepper", "lactose free milk", "kale",
"margerine", "corn", "not paprika", "low fat cream", "healthy turkey"]
    quantity = [1,3,5,6,7,3,4,8,9,2]

    recipe = {k:v for k,v in zip(fake_recipe_ingredients, quantity)}
    healthy_recipe = {k:v for k,v in zip(recipe.keys(), healthy_stuff)}
    print("normal recipe\n")
    for k,v in recipe.items():
        print(f'{k}: {v}')
    print("")
    print("healthy alternative\n")
    for k,v in healthy_recipe.items():
        print(f'{k}: {v}')
```

Final Execution Output

Output of the most recent code execution.

```
[1, 2, 3]
[1, 2, 3, 4]
['4', 3]
['4', 3, 4]
['4', 3]
{'a', 'b', 'c'}
[1, 2, 3, 4]
{'key': 'value'}
{2: 'value'}
(1, 'hello', 13.95)
1 hello 13.95
Original List items
[1, 2, [4, 5]]
original list after modifications to shallow copied list
[1, 2, [8, 5]]
Original list items
[1, 2, [4, 5]]
Original list after modifications made to deep copy
[1, 2, [4, 5]]
Deep copied list
[1, 2, [9, 5]]
Enter items separated by space to place in queue
[1, 2, 5, 48]
[5, 48]
[0, 1, 7, 9]
normal recipe

flour: 1
salt: 3
pepper: 5
milk: 6
lettuce: 7
butter: 3
popcorn: 4
paprika: 8
cream: 9
turkey: 2

healthy alternative

flour: not flour
salt: not salt
pepper: less pepper
milk: lactose free milk
lettuce: kale
butter: margerine
popcorn: corn
paprika: not paprika
cream: low fat cream
turkey: healthy turkey
```

Inputs to the Code

Most recent user-typed inputs to the program (STDIN).

```
1 2 5 48
```

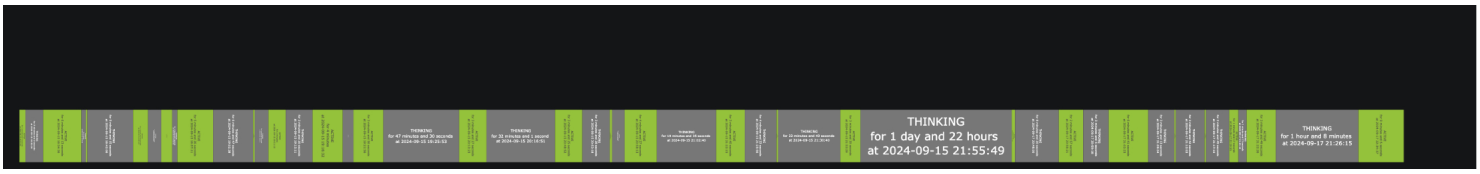
Most Common Error

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 136, in <module>
    phonebook(M)
  File "/usr/src/app/phonebook.py", line 17, in phonebook
    sorted_by_keys = {k: contacts[i] for i in dict_keys}
                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/src/app/phonebook.py", line 17, in <dictcomp>
    sorted_by_keys = {k: contacts[i] for i in dict_keys}
                      ^
NameError: name 'k' is not defined
```

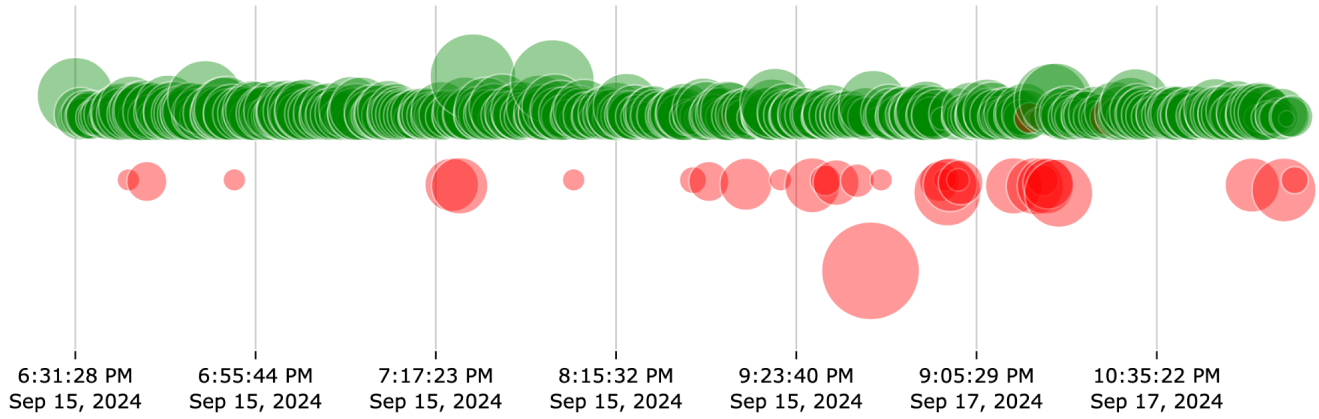
Active Sessions

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



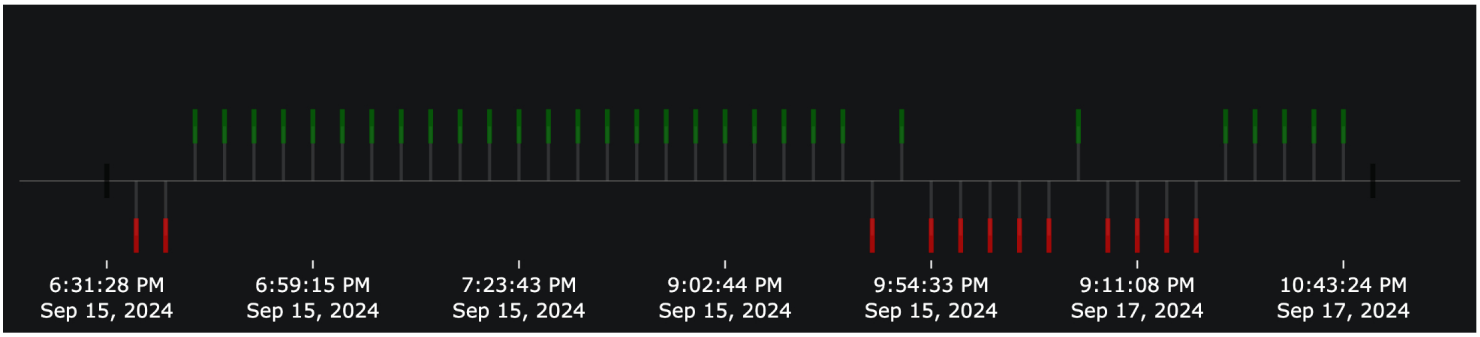
Characters Added and Removed Timeline

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



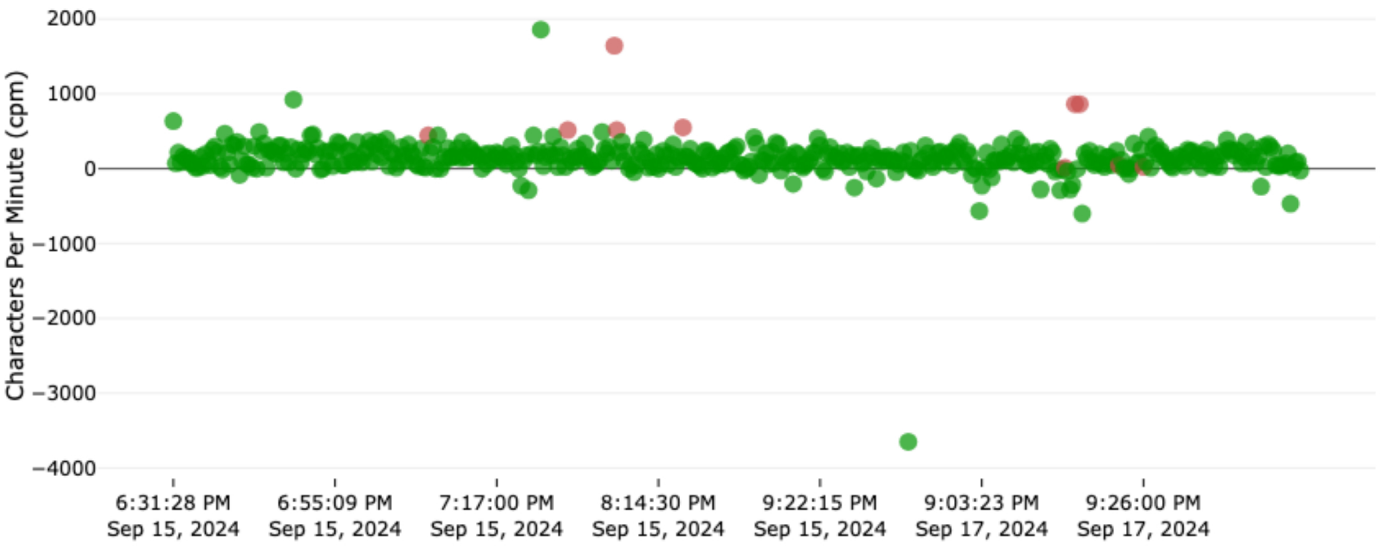
Code Execution Timeline

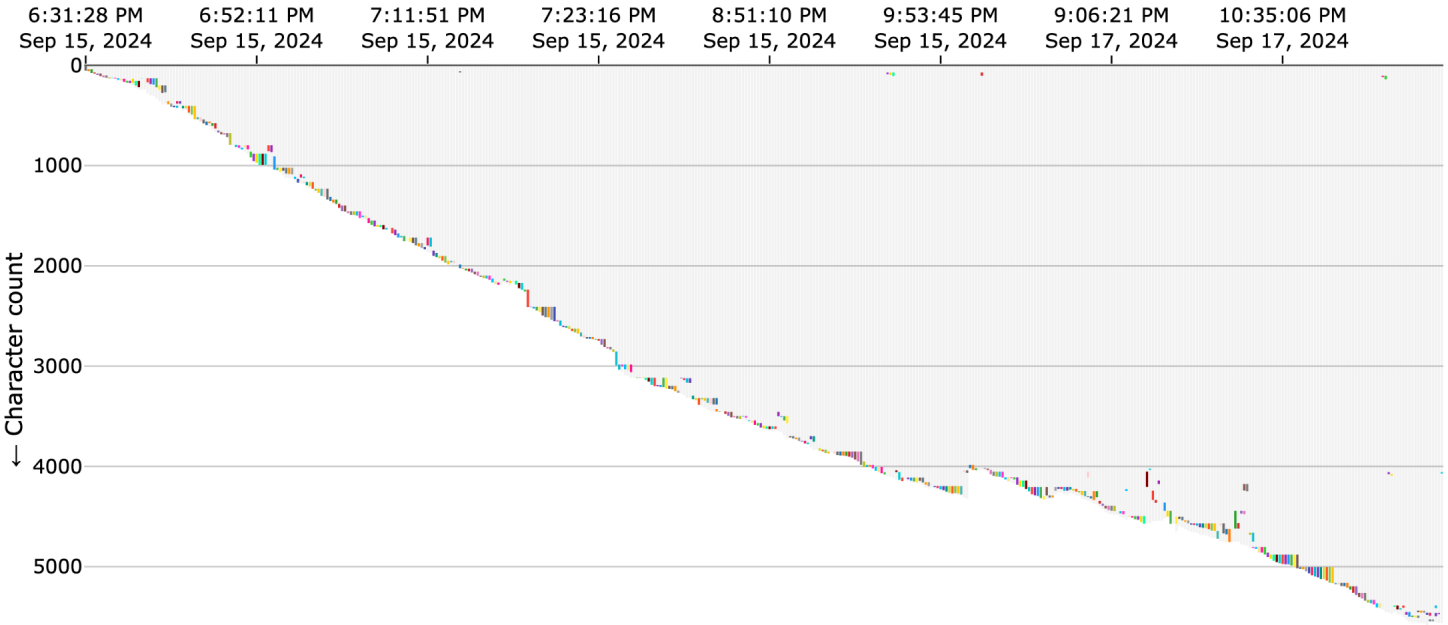
Green and red dots correspond to successful and failed executions.



Typing Speed Timeline

Typing speed can be negative when text is deleted.



Edited Line Locations

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