assignment-5

By: Alexander
UMSL

Coding Journey Summary:

- ⏱️ 1 hour 19 minutes 19 seconds of active typing
- 📅 1 day 30 minutes 30 seconds total duration
- 🔢 4,966 characters | 150 lines
- 📈 Typing speed: 2 character(s) per second
- 🗄️ Process Data: Showing 240 changes made every 5+ seconds (of 479 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
from number_1 import number_1
from number_2 import number_2
from number_3 import number_3
from number_4 import number_4
from number_5 import number_5
from number_6 import number_6
from number_7 import number_7

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

number_1()
number_2(D, M)
number_3(BDAY)
number_4()
number_5(D, M)
number_6()
number_7()

==+==
FILE: number_1.py
==+==
```

```
import numpy as np

def number_1():
    """
    Broadcasting relaxes numpy's constraint of arrays having the same shape
    Here the scalar b is "stretched" across to multiply each element of array a
    """
    print("Running solution to number 1")
    a = np.array([1.0, 2.0, 3.0])
    b = 2.0
    # result is [2., 4., 6.]
    print(a * b)

    """
    Here is an example of column stretching and row stretching
    the colums are stretched across a (4x1) and rows are stretched across b(3) to create the result
(4x3)
    """
    a = np.array([0.0, 10.0, 20.0, 30.0])
    b = np.array([1.0, 2.0, 3.0])
    """

    array([[ 1.,   2.,   3.],
        [11.,  12.,  13.],
        [21.,  22.,  23.],
        [31.,  32.,  33.]])
    """
    print(a[:, np.newaxis] + b)

==+==
FILE: number_2.py
==+==

import numpy as np

def number_2(D, M):
    print("Running solution to number 2")
    array_1 = np.array([[0, D], [M, 3]])
    array_2 = np.array([[4, 2*D], [2*M, 7]])
    array_3 = np.vstack((array_1, array_2))
    print(f'array_1: {array_1}')
    print(f'array_2: {array_2}')
    print(f'array_3: {array_3}')

    array_4 = np.hstack((array_1, array_2))
    print(f'array_4: {array_4}')
    array_5 = np.vstack((np.copy(array_4), np.copy(array_4)))
    print(f'array_5: {array_5}')

    array_6 = np.hstack((np.copy(array_3), np.copy(array_3)))
    print(f'array_6: {array_6}')

==+==
FILE: number_3.py
==+==

import numpy as np
```

```
def number_3(BDAY):
    print("Solution to number 3")
    if BDAY % 2 == 0:
        checkerboard = np.tile(np.array([['-', '*'], ['*', '-']]), (7,7))
        print(checkerboard)
    print('rand 5x5 array')
    rng = np.random.default_rng()
    random_array = rng.integers(99, size=(5,5,))
    print(random_array)
    # bincount only works on 1d arrays......
    counts = np.bincount(random_array.flatten())
    print(counts)


==+==
FILE: number_4.py
==+==

import numpy as np
def number_4():
    print("Solution to number 4")
    rng = np.random.default_rng()
    random_array = rng.integers(15, size=(10,10))
    # find min value location. This would be best matchup
    print(np.argmin(random_array))

==+==
FILE: number_5.py
==+==

import pandas as pd
import numpy as np

def number_5(D, M):
    some_series = pd.Series([D, M, D+M, M-D])
    print(some_series)
    another_series = pd.Series(data=[100*float(D)]*5)
    print(another_series)
    stats_series = pd.Series(data=np.random.default_rng().integers(11, size=D*10))
    print(stats_series.describe())
    temperature = pd.Series(data=[98.6,98.9,100.2,97.9], index=["Julie", "Charlie", "Sam",
"Andrea"])
    print(temperature)
    temp_dict = {
        "names": ["Julie", "Charlie", "Sam", "Andrea"],
        "values": [98.6,98.9,100.2,97.9]
    }
    temp_series = pd.Series(temp_dict)
    print(temp_series)

==+==
FILE: number_6.py
==+==

import pandas as pd
def number_6():
    print("solution to number 6")
```

```
    temp_dict = {
        "Maxine": [98, 99, 100],
        "James": [94, 98, 103],
        "Amanda": [98, 99, 97.5]
    }
    temp_df = pd.DataFrame(temp_dict)
    print(temp_df)
    index_df = pd.DataFrame(temp_dict, index=["Morning", "Afternoon", "Evening"])
    print(index_df)
    print("\nMaxines temp")
    print(index_df.Maxine)
    print("\nRow of morning temps")
    print(index_df.loc["Morning"])
    print("\nRows for Morning and Evening")
    print(index_df.loc[["Morning", "Evening"]])
    print("\nTemps for Maxine and Amanda")
    print(index_df[["Maxine", "Amanda"]])
    print("\nElements Amanda and Maxine in Morning and Afternoon")
    print(index_df.loc[["Morning", "Afternoon"], ["Maxine", "Amanda"]])
    print("\nDescribe")
    print(index_df.describe())
    print("Sort temps so column names are in alphabetical order")
    print(index_df.sort_index(axis=1))




number_6()

==+==
FILE: number_7.py
==+==

import pandas as pd
def number_7():
    # create two "tables"
    commercials_dict = {
        "Midwest": ["Commercial 1", "Commercial 2", "Commercial 3"],
        "North": ["Commercial 4", "Commercial 5", "Commercial 6"],
        "South": ["Commercial 7", "commercial 8", "Commercial 9"]
    }
    sales_dict = {
        "Midwest": [10, 20, 30],
        "North": [5, 10, 22],
        "South": [11,2,3]
    }
    commercials_df = pd.DataFrame(commercials_dict, index=["Summer", "Fall", "Spring"])
    sales_df = pd.DataFrame(sales_dict, index=["Summer", "Fall", "Spring"])
    print("Commercials")
    print(commercials_df)
    print("\nSales df")
    print(sales_df)
    # map the regions
    for k,v in zip(commercials_df["Midwest"], sales_df["Midwest"]):
        print(k + ' generated ' + str(v) + ' sales.')
    for k,v in zip(commercials_df["North"], sales_df["North"]):
        print(k + ' generated ' + str(v) + ' sales.')
```

```
    for k,v in zip(commercials_df["South"], sales_df["South"]):
        print(k + ' generated ' + str(v) + ' sales.')
```

Final Execution Output

Output of the most recent code execution.

```
solution to number 6
   Maxine  James  Amanda
0      98     94    98.0
1      99     98    99.0
2     100    103    97.5
           Maxine  James  Amanda
Morning        98     94    98.0
Afternoon      99     98    99.0
Evening       100    103    97.5

Maxines temp
Morning       98
Afternoon     99
Evening      100
Name: Maxine, dtype: int64

Row of morning temps
Maxine    98.0
James     94.0
Amanda    98.0
Name: Morning, dtype: float64

Rows for Morning and Evening
         Maxine  James  Amanda
Morning      98     94    98.0
Evening     100    103    97.5

Temps for Maxine and Amanda
           Maxine  Amanda
Morning        98    98.0
Afternoon      99    99.0
Evening       100    97.5

Elements Amanda and Maxine in Morning and Afternoon
           Maxine  Amanda
Morning        98    98.0
Afternoon      99    99.0

Describe
       Maxine       James     Amanda
count     3.0    3.000000   3.000000
mean     99.0   98.333333  98.166667
std       1.0    4.509250   0.763763
min      98.0   94.000000  97.500000
25%      98.5   96.000000  97.750000
50%      99.0   98.000000  98.000000
75%      99.5  100.500000  98.500000
max     100.0  103.000000  99.000000
Sort temps so column names are in alphabetical order
           Amanda  James  Maxine
```

```
Morning      98.0     94      98
Afternoon    99.0     98      99
Evening      97.5    103     100
Running solution to number 1
[2. 4. 6.]
[[ 1.  2.  3.]
 [11. 12. 13.]
 [21. 22. 23.]
 [31. 32. 33.]]
Running solution to number 2
array_1: [[ 0  2]
 [12  3]]
array_2: [[ 4  4]
 [24  7]]
array_3: [[ 0  2]
 [12  3]
 [ 4  4]
 [24  7]]
array_4: [[ 0  2  4  4]
 [12  3 24  7]]
array_5: [[ 0  2  4  4]
 [12  3 24  7]
 [ 0  2  4  4]
 [12  3 24  7]]
array_6: [[ 0  2  0  2]
 [12  3 12  3]
 [ 4  4  4  4]
 [24  7 24  7]]
Solution to number 3
[['-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*']
 ['*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-']
 ['-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*']
 ['*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-']
 ['-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*']
 ['*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-']
 ['-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*']
 ['*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-']
 ['-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*']
 ['*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-']
 ['-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*']
 ['*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-' '*' '-']
 [... [too long STDOUT trimmed]
```

Most Common Error

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

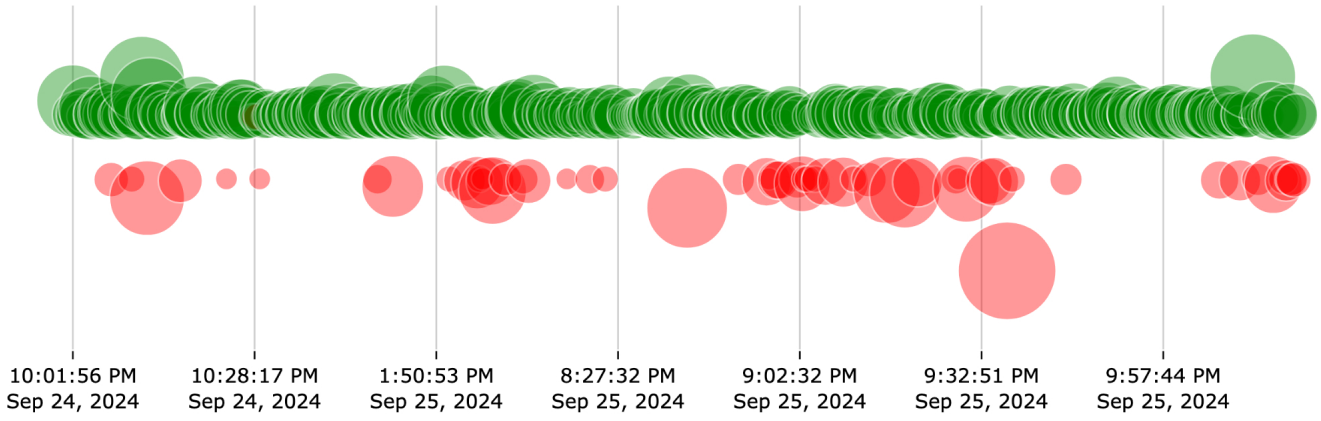
```
STDERR:
Traceback (most recent call last):
  File "number_7.py", line 26, in <module>
    number_7()
  File "number_7.py", line 20, in number_7
    print(k + ' generated ' + v + ' sales.')
TypeError: can only concatenate str (not "int") to str


Commercials
             Midwest         North         South
Summer  Commercial 1  Commercial 4  Commercial 7
Fall    Commercial 2  Commercial 5  commercial 8
Spring  Commercial 3  Commercial 6  Commercial 9

Sales df
        Midwest  North  South
Summer       10      5     11
Fall         20     10      2
Spring       30     22      3
```

Active Sessions

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

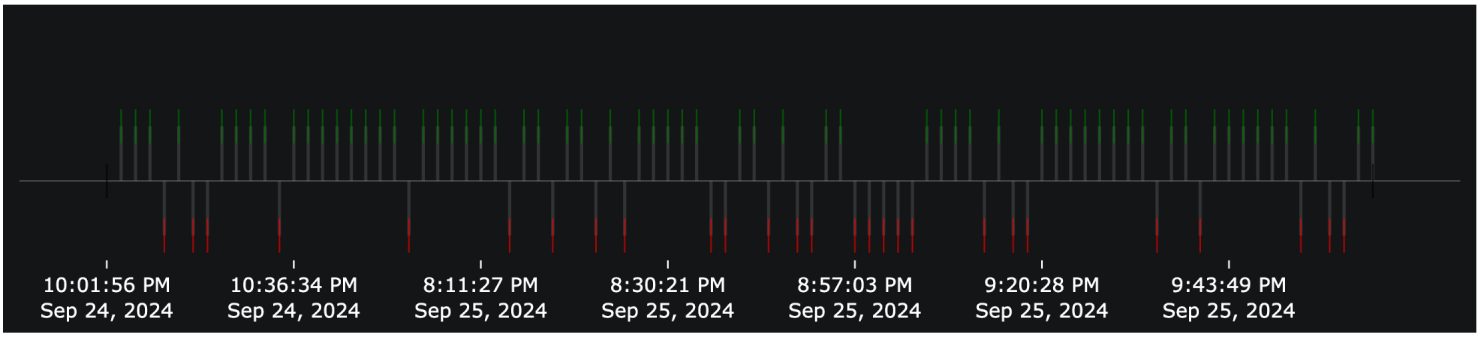
Characters Added and Removed Timeline

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



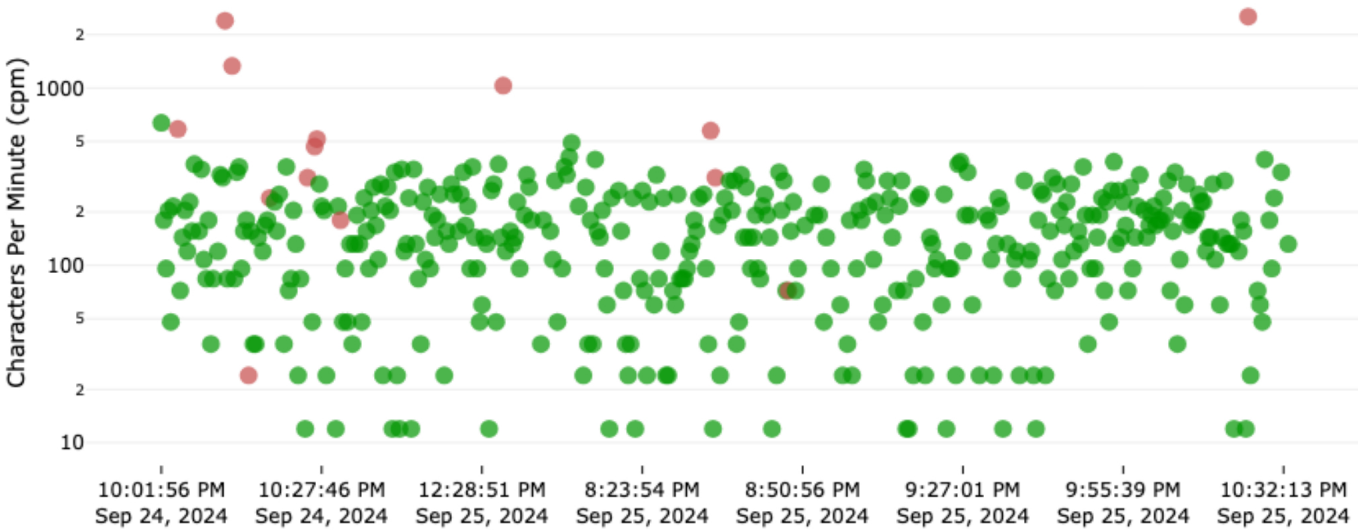
Code Execution Timeline

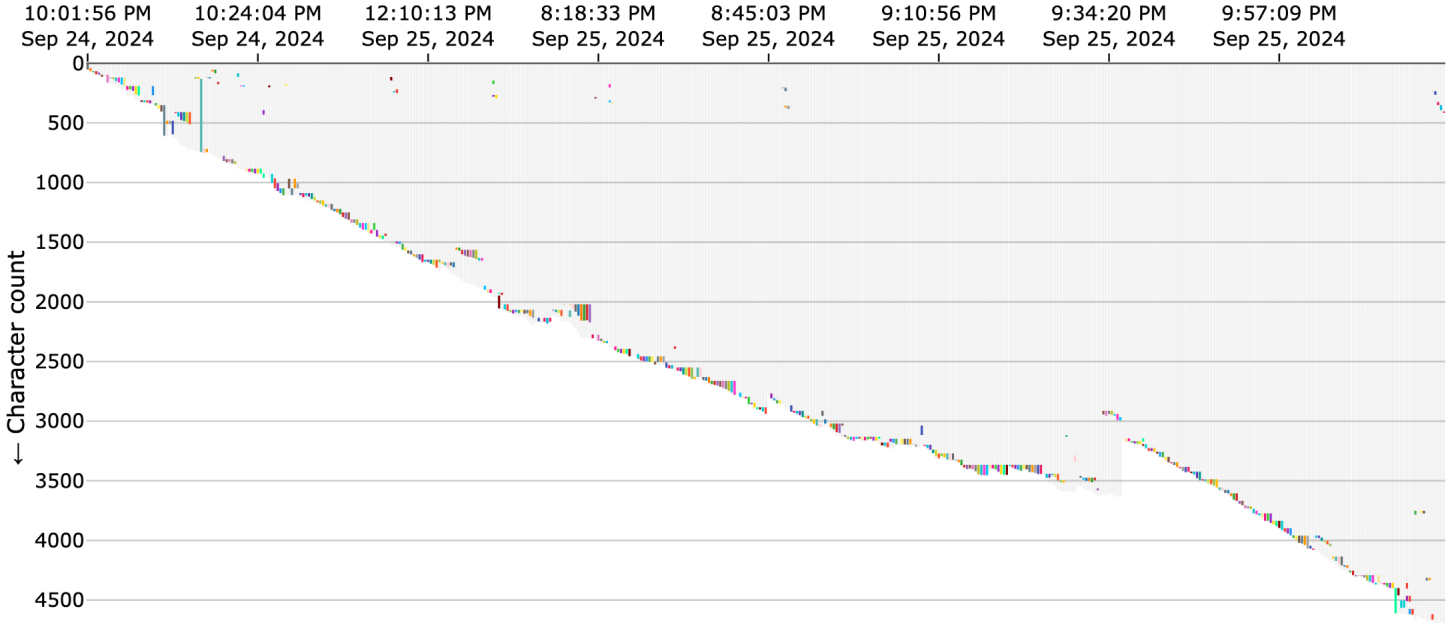
Green and red dots correspond to successful and failed executions.



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.