

情報工学実験 II 第5回レポート

サイバーセキュリティ基礎実験2

3年 情報工学科 19番 瀧口大地

提出期限: 2025年12月26日09:00

提出日: 2025年12月22日23:00

共同実験者:3班

4番 井村周慈

9番 カリラ

14番 後藤輝一

30番 三原瑚桜

34番 山口紗音

アブストラクト

本実験では、Python(パイソン)と呼ばれるスクリプト言語の文法や基本的な構造について学習し、Pythonを用いて様々なデータ整理や加工、テキストファイル処理及びツールの基本的な作成方法について習得すること、他のプログラミング言語と比較してPythonの特徴や利点についても学ぶことを目的とする。具体的にはPythonの基本文法を学ぶため、演算プログラム、九九表の出力プログラムを作成する。また、Pythonで用いられるモジュールを用いて日数計算プログラムを作成する。さらに、テキストファイルを読み込み、ソートするプログラム、重複データを削除するプログラムを作成する。それらの総括として、アクセスログから正規表現を用いてSQLコマンドインジェクションを検出するプログラム、各IPアドレスのアクセス回数を棒グラフとして出力するプログラムを作成する。また、発展的な課題としてmatplotlibを用いて、各IPアドレスのアクセス数を棒グラフとして出力するプログラムを作成する。具体的な方法として、重複データを削除するプログラムでは重複しているかどうかの判定をsetを用いて行った。また、アクセスログから正規表現を用いてSQLコマンドインジェクションを検出するプログラムでは、SQLコマンドインジェクションで見られる特徴的な文字列を正規表現で検出することで、検出を試みた。また、各IPアドレスのアクセス回数を棒グラフとして出力するプログラムでは、辞書を用いて各IPアドレスのアクセス回数をカウントしopenpyxlを用いてエクセルファイルに棒グラフを出力した。同様にmatplotlibを用いて各IPアドレスごとのアクセス数を棒グラフとして出力した。作成したこれらのプログラムは実験時間内に作成し、正常に動作することが確認できた。

第1章 実験目的

本実験では，Python(パイソン)と呼ばれるスクリプト言語の文法や基本的な構造について学習し，Python を用いて様々なデータ整理や加工，テキストファイル処理及びツールの基本的な作成方法について習得することを目的とする．また，他のプログラミング言語と比較してPython の特徴や利点についても学ぶ．

第2章 実験結果

指導書に示されたプログラミング課題をそれぞれアイデア、プログラム、結果に分けて述べていく。

2.1 スクリプトプログラミングの学習 (6.10.1)

2.1.1 アイデア

本実験課題ではPythonを用いて簡単な演算を行う。演算子と役割は以下の表 2.1 に示すとおりである。

表 2.1: python での四則演算子

+	加算
-	減算
*	乗算
/	除算
%	剰余
//	除算 (切り捨て)

これらの演算子を用いて指導書に示されたサンプルプログラムを実行する。

2.1.2 プログラム

以下に指導書のサンプルプログラムを示す。

サンプルプログラム

```
1 print("10 + 3 =", 10+3)
2 print("10 - 3 =", 10-3)
3 print("10 * 3 =", 10*3)
4 print("10 / 3 =", 10/3)
5 print("10 % 3 =", 10%3)
6 print("10 // 3 =", 10//3)
```

サンプルプログラムでは1行目から順に10と3の加算、減算、乗算、除算、剰余、切り捨て除算を行っている。

2.1.3 結果

サンプルプログラムを実行した結果を図 2.1 に示す.

```
>python four_arithmetic.py
10 + 3 = 13
10 - 3 = 7
10 * 3 = 30
10 / 3 = 3.3333333333333335
10 % 3 = 1
10 // 3 = 3
```

図 2.1 サンプルプログラムの実行結果

図 2.1 より, 表 2.1 で示した通りに動作していることがわかる. よって指導書に示されたサンプルプログラムは正しく動作したと言える.

2.2 九九表の出力 (6.10.2)

本実験課題では図 2.2 のような九九表を出力するプログラムを作成する.

```

      | 1 2 3 4 5 6 7 8 9
-----+-----
1| 1 2 3 4 5 6 7 8 9
2| 2 4 6 8 10 12 14 16 18
3| 3 6 9 12 15 18 21 24 27
4| 4 8 12 16 20 24 28 32 36
5| 5 10 15 20 25 30 35 40 45
6| 6 12 18 24 30 36 42 48 54
7| 7 14 21 28 35 42 49 56 63
8| 8 16 24 32 40 48 56 64 72
9| 9 18 27 36 45 54 63 72 81

```

図 2.2 出力したい九九の表

2.2.1 アイデア

図 2.2 の最初の 2 行はそれぞれ掛ける数と区切りの横線を出力しているだけである。3 行目以降は掛ける数と同じような形で九九表が出力されているため、指導書の指定の通りにフォーマットを使うとコンパクトなコーディングができる。また、各数値は 1 桁のものでも 2 桁のものでも 2 桁相当の空間を取って出力するため、これもフォーマットを活用できる。

2.2.2 プログラム

アイデアに則って組んだ九九表を出力するプログラムを以下に示す。end="" は改行無しを示す。

九九表を出力するプログラム

```
1  # フォーマット
2  f = " {0} |"
3  n = " {0:2}"
4
5  # 1行目
6  print(f.format(" "),end='')
7  for i in range(1,10):
8      st = " " + (str)(i) # C++キャストも動く, といふかnの意味よ
9      print(n.format(st),end='')
10
11 # 2行目
12 print("\n"+"----+-----")
13
14 # 3行目以降
15 for i in range(1,10):
16     print(f.format(i),end='')
17     for j in range(1,10):
18         print(n.format(i*j),end='')
19     print()
```

コメントにある通り, 2~3行目がフォーマット, 6~9行目が出力の1行目, 12行目
が出力の2行目, 15~19行目が出力の3行目以降を処理している.

2.2.3 結果

図 2.3 に実行結果を示す. 図 2.3 の九九表が図 2.2 と一致していることから, プログラ
ムは正しく動作したと言える.

```

>python k001.py
  | 1  2  3  4  5  6  7  8  9
-----+-----
1| 1  2  3  4  5  6  7  8  9
2| 2  4  6  8 10 12 14 16 18
3| 3  6  9 12 15 18 21 24 27
4| 4  8 12 16 20 24 28 32 36
5| 5 10 15 20 25 30 35 40 45
6| 6 12 18 24 30 36 42 48 54
7| 7 14 21 28 35 42 49 56 63
8| 8 16 24 32 40 48 56 64 72
9| 9 18 27 36 45 54 63 72 81

```

図 2.3 k001.py の実行結果

2.3 閏年判定 (6.10.3)

本実験課題では入力された年が閏年かどうかを判定するプログラムを作成する。

2.3.1 アイデア [1]

Python には calendar モジュールというものがあり，これを用いることで 1 行のコードで与えられた数値型の年が閏年かどうかを判定できる．このモジュールを用いて本課題を解決する．

2.3.2 コード

前述の calendar モジュールを用いて作成したコードを以下に示す．

```

1  import calendar
2
3  year = input("何年 (YYYY) : ")
4  year_i = int(year)
5
6  if calendar.isleap(year_i):
7      print(year_i , "年は閏年です。")
8  else:
9      print(year_i , "年は平年です。")

```

コードの1行目ではcalendarモジュールをインポートしており、3行目で年数の入力処理、4行目で入力の数値型変換、6行目以降でcalendarモジュールを用いた閏年判定と出力を行っている。

2.3.3 結果

以下の図2.4にプログラムで示したコードを実行した結果を示す。平年と閏年をそれぞれ入力し、正しい結果が出力されるか試した。

```

> python k011.py
何年 (YYYY) : 2025
2025 年は平年です。
> python k011.py
何年 (YYYY) : 2020
2020 年は閏年です。

```

図 2.4 k011.py の実行結果

図2.4の出力結果から、閏年を閏年、平年を平年と出力できたことがわかる。このことから本実験のプログラムは正しく動作したと言える。

2.4 ファイル操作 (6.10.4)

本実験では temperature.dat ファイルを読み込み、昇順か降順か指定された順にソートするプログラムを作成する。

2.4.1 アイデア

本実験課題では入力に読み込むファイル (temperature.dat) と昇順か降順かを表す値 (0,1) を入れる。次にその dat ファイルを読み込み, " :" で区切られている日付と数値変換した温度とをセット (配列) として配列に格納する。その配列の中身を指定された通りにソートし、ファイルの更新を行う。また、指導書のコマンドプロンプト出力例に変更されたファイルの中身があったため、念のため print による出力も行う。本実験でのファイル操作は指導書の指定通りに with 文を用いて行うこととする。

2.4.2 プログラム

以下にアイデアに準じたプログラムを示す。

他実験と同様に、本ソートプログラムの説明を各行で行う。1 行目の data 配列は前述したセット (配列) を格納する配列である。3,4 行目はファイルの名前とソート順の入力である。6~11 行目は入力されたファイルを with 文で閲覧 (' r') し、for 文を用いて行ごとに処理を行っている。処理の中身としては " :" の前後で情報を区切り、それらを date,temp として data 配列の中に格納している。ここで、temp はソートの際に数値として扱うため float 型に変換している。15 行目はコメントにある通り、ソート順を表す入力の Rule が 1 なら True, その他なら false とする変数を定義している。18 行目はそれらを用いてソートを行っている。key には扱うデータを参照させるのだが、データの位置を関数の出力しか指定できない。そのため簡易関数 lambda を使用して data の 1 個目、つまり温度 (0 個目が日付) を参照している。21,22 行目はコマンドプロンプト用にソートされた data 配列の中身を " :" を挟み print で出力している。25~27 行目は with 文の書き込み (' w' で) dat ファイルを開き data 配列の中を上書きで書き込んでいる。

```
1  data = []
2
3  file_path = input("ファイル：")
4  Rule = input("ルール（昇順：0,降順：1）")
5
6  with open(file_path, 'r', encoding='utf-8') as f:
7      for line in f:
8          parts = line.split(':')
9          date = parts[0]
10         temp = float(parts[1])
11         data.append((date, temp))
12
13     #ソート処理（昇順：0, 降順：1）
14     # reverse=True で降順、False で昇順
15     is_reverse = True if Rule == "1" else False
16
17     #温度を基準にソート
18     data.sort(key=lambda x: x[1], reverse=is_reverse)
19
20     #出力処理
21     for date, temp in data:
22         print(f"{date}: {temp}")
23
24     #ファイル書き換え（上書き）
25     with open(file_path, 'w', encoding='utf-8') as f:
26         for date, temp in data:
27             f.write(f"{date}: {temp}\n")
```

2.4.3 結果

まず指導書の出力例に則ってコマンドプロンプト上の出力結果を示す。最初に temperature.dat と 1 を入力し降順ソートを行った結果を図 2.5 に示す。

```
> python k023.py
ファイル：temperature.dat
ルール（昇順：0, 降順：1） 1
2015/Aug/08: 99.27
2015/Aug/15: 39.65
2015/Aug/02: 39.56
2015/Aug/23: 39.25
2015/Aug/20: 39.06
2015/Aug/06: 37.93
2015/Aug/26: 37.47
2015/Aug/24: 37.12
2015/Aug/19: 36.77
2015/Aug/27: 36.48
2015/Aug/05: 36.23
2015/Aug/10: 35.71
2015/Aug/29: 35.48
2015/Aug/30: 35.39
2015/Aug/17: 35.22
2015/Aug/12: 34.75
2015/Aug/09: 34.63
2015/Aug/11: 34.52
2015/Aug/25: 33.77
2015/Aug/16: 33.62
2015/Aug/07: 33.24
2015/Aug/13: 32.59
2015/Aug/03: 32.38
2015/Aug/04: 32.36
2015/Aug/18: 31.81
2015/Aug/21: 31.65
2015/Aug/01: 31.34
2015/Aug/28: 31.13
2015/Aug/14: 31.07
2015/Aug/31: 30.12
2015/Aug/22: 12.58
```

図 2.5 k023.py の実行結果 (cmd)

図 2.5 を見るとコマンドプロンプトへの出力は正しく降順ソートされたと言える。次

にこの状態のままもう一度 k023 を実行し temperature.dat と 0 を入力し昇順ソートを行ったコマンドプロンプトの結果を図 2.6 に示す。

```
> python k023.py
ファイル: temperature.dat
ルール (昇順: 0, 降順: 1) 0
2015/Aug/22: 12.58
2015/Aug/31: 30.12
2015/Aug/14: 31.07
2015/Aug/28: 31.13
2015/Aug/01: 31.34
2015/Aug/21: 31.65
2015/Aug/18: 31.81
2015/Aug/04: 32.36
2015/Aug/03: 32.38
2015/Aug/13: 32.59
2015/Aug/07: 33.24
2015/Aug/16: 33.62
2015/Aug/25: 33.77
2015/Aug/11: 34.52
2015/Aug/09: 34.63
2015/Aug/12: 34.75
2015/Aug/17: 35.22
2015/Aug/30: 35.39
2015/Aug/29: 35.48
2015/Aug/10: 35.71
2015/Aug/05: 36.23
2015/Aug/27: 36.48
2015/Aug/19: 36.77
2015/Aug/24: 37.12
2015/Aug/26: 37.47
2015/Aug/06: 37.93
2015/Aug/20: 39.06
2015/Aug/23: 39.25
2015/Aug/02: 39.56
2015/Aug/15: 39.65
2015/Aug/08: 99.27
```

図 2.6 k023.py の実行結果 (cmd)

図 2.5 から data 配列の更新とコマンドプロンプトへの出力が正しく行えたと言える。また、ファイルの更新が行えているかを確認するために、図 2.5 時点の temperature.dat ファイルと図 2.6 時点の temperature.dat ファイルを記録していた。これらを図 2.7 と図 2.8 に示す。

```
2015/Aug/08: 99.27
2015/Aug/15: 39.65
2015/Aug/02: 39.56
2015/Aug/23: 39.25
2015/Aug/20: 39.06
2015/Aug/06: 37.93
2015/Aug/26: 37.47
2015/Aug/24: 37.12
2015/Aug/19: 36.77
2015/Aug/27: 36.48
2015/Aug/05: 36.23
2015/Aug/10: 35.71
2015/Aug/29: 35.48
2015/Aug/30: 35.39
2015/Aug/17: 35.22
2015/Aug/12: 34.75
2015/Aug/09: 34.63
2015/Aug/11: 34.52
2015/Aug/25: 33.77
2015/Aug/16: 33.62
2015/Aug/07: 33.24
2015/Aug/13: 32.59
2015/Aug/03: 32.38
2015/Aug/04: 32.36
2015/Aug/18: 31.81
2015/Aug/21: 31.65
2015/Aug/01: 31.34
2015/Aug/28: 31.13
2015/Aug/14: 31.07
2015/Aug/31: 30.12
2015/Aug/22: 12.58
```

図 2.7 k023.py の実行結果 (ファイル)

```
2015/Aug/22: 12.58
2015/Aug/31: 30.12
2015/Aug/14: 31.07
2015/Aug/28: 31.13
2015/Aug/01: 31.34
2015/Aug/21: 31.65
2015/Aug/18: 31.81
2015/Aug/04: 32.36
2015/Aug/03: 32.38
2015/Aug/13: 32.59
2015/Aug/07: 33.24
2015/Aug/16: 33.62
2015/Aug/25: 33.77
2015/Aug/11: 34.52
2015/Aug/09: 34.63
2015/Aug/12: 34.75
2015/Aug/17: 35.22
2015/Aug/30: 35.39
2015/Aug/29: 35.48
2015/Aug/10: 35.71
2015/Aug/05: 36.23
2015/Aug/27: 36.48
2015/Aug/19: 36.77
2015/Aug/24: 37.12
2015/Aug/26: 37.47
2015/Aug/06: 37.93
2015/Aug/20: 39.06
2015/Aug/23: 39.25
2015/Aug/02: 39.56
2015/Aug/15: 39.65
2015/Aug/08: 99.27
```

図 2.8 k023.py の実行結果 (ファイル)

図 2.7, 図 2.8 からファイルの更新が適切に行えていることが分かった。
以上のことから本プログラムは題意に沿って動作できたと言える。

2.5 コマンドライン引数 (6.10.5)

本実験では複数のファイルを指定し、それぞれのファイルの文字数、単語数、行数をカウントしそれぞれの結果を出力する。ただしプログラム引数でファイルを指定する必要があり、(python k032.py fruit.dat vegetable.dat wc.txt) といったような形にする。この例では最後の 1 つを除く全てのファイルである fruit.dat と vegetable.dat の 2 つがカウントされるファイルであり、最後の wc.txt がカウントした結果が格納されるファイルである。

2.5.1 アイデア

入力を指定通りに行うために sys モジュールの sys.argv を用いる (参考: 指導書)。カウントされるファイルをこれまでと同様に with 文で読み込み、readlines 関数でテキスト全体を読み取る。そこから行数、空白や改行の split による単語数、単純な文字数を出力する。また、リストの配列を用いるため最終出力はリストを並べて出力できる join も用いる。最後にこれまでと同様に with 文の書き込みモードでカウントしたデータを一番後ろのファイルに格納する。

2.5.2 プログラム

以下にアイデアに準じたプログラムを示す。

アイデアで述べた通り、1 行目で sys モジュールをインポートし、3 行目の sys.argv[1:-1] (ファイル指定の最初から最後の 1 個手前まで) を読み込むファイル、4 行目 sys.argv[-1] (ファイル指定の最後) を書き込むファイルとしている。6 行目にリストを入れる配列 results を定義している。8~20 行目でアイデアにて述べた通りに行数、単語数、文字数をカウントし results にリストとして入れている。これを拡張 for 文で読み込むファイルの回数分行う。最後に join を用いて results コマンドプロンプトでの出力、ファイルへの書き込みを行い処理が終了する。


```

1  import sys
2
3  output_file = sys.argv[-1]
4  input_files = sys.argv[1:-1]
5
6  results = []
7
8  for filename in input_files:
9      with open(filename, 'r', encoding='utf-8') as f:
10         lines = f.readlines()
11
12         l_count = len(lines)
13         w_count = sum(len(line.split()) for line in lines)
14         c_count = sum(len(line) for line in lines)
15
16         # データの整形
17         results.append(f"file: {filename}")
18         results.append(f"lines: {l_count}")
19         results.append(f"words: {w_count}")
20         results.append(f"chars: {c_count}\n")
21
22     # --- 追加：cmd上への表示とファイル書き込み ---
23     output_text = "\n".join(results)
24     print(output_text)
25
26     # ファイルに保存
27     with open(output_file, 'w', encoding='utf-8') as f_out:
28         f_out.write(output_text)

```

2.5.3 結果

プログラムを実行した結果を示す。今回は fruit.dat と vegetable.dat に資料にあった overlap.dat から一部抜き出した内容を書き込みカウントされるファイルとして扱う。そのため以下に各 dat ファイルの中身を示す。

fruit.dat

```
1  raspberry raspberry
2  grape
3  peach
4  cherry cherry cherry
5  grape cherry strawberry
6  peach
7  apple
```

vegetable.dat

```
1  cherry mandarin orange apricot
2  apple banana apple
3  strawberry strawberry grape
```

これらを用いてプログラム k032 を実行した際のコマンドプロンプト出力を図 2.9 に示す。この時、vegetable.dat の文字数が 79 となっているが、改行を除いた文字数が $30+18+27=75$ 文字であり改行は 1 文字で合計 77 文字となる。計算が合わないが良くファイルを確認すると最後の行以外の最後に半角空白があった。これで合計 77 となり計算が成り立つ。fruit.dat でも同様に処理が行われているため、このような結果になったと分かる。思わぬハプニングだったが理論的に納得できた。以上のことから、内部の処理とコマンドプロンプト出力は正しく動作したと言える。

```
> python k032.py .\fruit.dat .\vegetable.dat wc.txt
file: .\fruit.dat
lines: 7
words: 12
chars: 95

file: .\vegetable.dat
lines: 3
words: 10
chars: 79
```

図 2.9 k032.py の実行結果 (cmd)

また、出力ファイル wc.txt の中身も図 2.10 に示し確認する.

```
file: .\fruit.dat
lines: 7
words: 12
chars: 95

file: .\vegetable.dat
lines: 3
words: 10
chars: 79
```

図 2.10 k032.py の実行結果 (ファイル)

これでファイル出力も問題なく行えていることが分かった.

以上のことから本実験課題のプログラムは想定・指定通りに動作したと言える.

2.6 アクセスログからの情報抽出 (6.10.6.1)

本実験課題では第 5 章実験で行った, Web サーバのアクセスログからアクセス数とアクセス元の IP アドレス数を調べる課題を Python で行う.

2.6.1 アイデア [2]

IP アドレスの抽出や SQL インジェクションの検出は指導書の指定の通り re モジュールを使って解決する. 指導書の URL が壊れていたためプログラム作成時に都度調べて書く. 本課題で今までと違うのは re モジュールともう 1 つ, アクセス元の IP アドレス数という単純な数ではなく種類数を調べるところである. 指導書にて軽く触れられているが, Python には set という重複を排除しながらデータを格納できる関数がある. これを用いることで効率的にアクセス元の IP アドレスを出力できる. また, リスト配列とは違って set 関数はハッシュ値を用いてデータを探するため, データの量が増えても検索速度が高速のままであるという長所もある. 本実験課題の都合上その仕様が活きることはないが有用である. その他の機能はこれまでの実験課題で用いた機能を使いまわすだけである. with を用いたファイルの閲覧や join を用いた複数データの効率的な出力, sys によるコマンドライン引数の活用等を re モジュールや set 関数と組み合わせ本実験課題の題意を満たす.

2.6.2 プログラム [3][4]

アイデアに準じたプログラムを以下に示す。

アクセスログからの情報抽出を行うプログラム

```
1  import sys
2  import re
3
4  # IPアドレス抽出用（行頭の数字列）
5  ip_regex = re.compile(r'^(\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})')
6
7  # ログ内の %27 ('), -- (コメント), %3D (=), OR などを検知
8  sqli_regex = re.compile(r"(%27|'|--|'%3D|or\s+)")
9
10 input_file = sys.argv[-1]
11
12 total_access = 0
13 unique_ips = set()
14 attacker_ips = set()
15
16 with open(input_file, 'r', encoding='utf-8') as f:
17     for line in f:
18         total_access += 1
19
20         # IPアドレスの抽出
21         match = ip_regex.match(line)
22
23         if match:
24             ip = match.group(1)
25             unique_ips.add(ip)
26
27         # SQLインジェクションのパターンが含まれるかチェック
28         if sqli_regex.search(line):
29             attacker_ips.add(ip)
30
31 # 結果の出力
32 print("アクセス数：", total_access)
33 print("アクセス元のIPアドレス数：", len(unique_ips))
34 print("攻撃元のIPアドレス：", ', '.join(attacker_ips))
```

1,2 行目はモジュールのインポートを行っている。5,8 行目では re モジュールを用いて正規表現で IP アドレスと SQL インジェクションを検出している。具体的には、IP アドレスは行の先頭にあるため”^” で先頭の要素に限定し、\d(数値) の {1,3} (1~3 文字のセット) と”.” の連結で成り立つと定義した。SQL コマンドインジェクションの検出の方はコード内で説明されている通り特有の記号などを含まかどうかで定義している。10 行目はコマンドライン引数の復習で access.log を入力する。12 行目から 14 行目は合計アクセス数、アクセスした IP アドレス、攻撃者の IP アドレスを扱う変数の定義である。2 種の IP アドレスは set 関数を用いて重複が起らないようにデータを格納できるようにしている。16 行目からの with 文も以前の課題と同様に行ごとの for 文で制御する。18 行目では行ごとにアクセスが行われるため for 文が回るたびにアクセス数に 1 を加算している。21 行目のように ip_regex.match(line) とすることで行ごとの IP アドレスが先に定義した ip_regex に適する値があるかどうかを bool 型で得られる。23~29 行目ではこれを用いて 2 種 IP アドレスに定義した正規表現に当てはまるものがあれば変数に追加するという処理を作っている。32,33 行目はアクセス数とアクセス元の IP アドレスの出力を行っている。34 行目では SQL コマンドインジェクションを行った IP アドレスを出力している。また、以下に今回使用した access.log(S0414) の中身のある 1 行を示す。これで最初の要素のみを IP アドレスとして使う理由や IP アドレスの正規表現のロジックを上のようにした理由等が裏付けられる。

Listing 1 Apache のログファイルの例

```
1 10.4.0.19 - - [14/Nov/2025:09:30:01 +0900] "GET
    /login.php?id=osaka&pass=1qaz%21QAZ HTTP/1.1" 200 403
    "http://10.4.4.0/" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
    Gecko/20100101 Firefox/128.0"
```

2.6.3 結果

プログラム k061 として上記のプログラムを実行した結果は以下図 2.11 のようになった。access.log の行数は 78 行であるため正しい結果が得られている。アクセス元の IP アドレスは班員の 7 人分あるかと思われたが、確認してみたところ班員の 1 人 (38 番) が逆の方のみで作業を行っていた。そのためアクセス元 IP アドレス数は 6 人で間違いない。SQL インジェクションを行っていた IP アドレスも 10.4.0.19, 10.4.4.0, 10.4.0.15 以外では見つからなかった。そのため本実験課題のプログラムは正しく動いたと言える。実行結果が得られるまでは作成した正規表現が適しているかわからなかったが正しく結果が得られているためこれは今後も活用できると考えられる。

```
> python k061.py access.log
アクセス数： 78
アクセス元の IP アドレス数： 6
攻撃元の IP アドレス： 10.4.0.19, 10.4.4.0, 10.4.0.15
```

図 2.11 k061.py の実行結果

2.7 Python,Excel 間の連携

ここでは、Web サーバのアクセスログから「IP アドレスごとのアクセス数」を調べ、第 1 列を IP アドレス、第 2 列をアクセス回数とする CSV を出力し、それを元に縦軸をアクセス回数、横軸を IP アドレスとしたグラフを作成するプログラムについて述べる。

2.7.1 アイデア [5][6][7]

本実験課題は途中までが前課題と同じ内容である。しかし私が実験時間中に AI を用いてコードを出力させてみたところ、画期的な関数を発見したためこちらを活用し改めて処理を作り直す。用いるのは Counter 関数であり、これは何が何個あるか、を数えられるものである。これを用いることでリスト内の重複数を数えながら IP アドレスとその回数のリストでデータを格納しておける。さらに `.most_common()` を用いることで値（今回の場合回数）が多い順に並べ替えることができる。これを用いて IP アドレスとその登場回数を効率的に調べ活用する。Excel(csv) についての指導書に指定されている通り `openpyxl` を用いる。Excel ファイルを作る `Workbook` 関数や初期シートを指定できる `wb.active` コマンドなどを用い Excel にデータを入力していく。最後に `openpyxl` に含まれる `BarChart` 関数で棒グラフを作成し操作する。Reference コマンドで使用するデータや横軸の単位に使う値の範囲指定が行える。 `add_data` でデータの追加、 `set_categories` で軸の単位追加、 `add_chart` コマンドで場所を決めグラフを表示することなどができる。これらを駆使しプログラムを作成する。

2.7.2 プログラム

アイデアに基づき作成したプログラムを以下に示す。授業中に AI に作成させたコードをそのまま使ってもよかったが、あまりに冗長（驚異の 75 行）だったため自分で作り直すことにした。

```
1 import sys, re, openpyxl
2 from collections import Counter
3 from openpyxl.chart import BarChart, Reference
4
5 log_file, out_xlsx = sys.argv[1], sys.argv[2]
6 ip_regex = re.compile(r'^(\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})')
7
8 # 1. IP抽出と集計 (Counterで一撃)
9 with open(log_file, 'r', encoding='utf-8') as f:
10     ips = [ip_regex.match(line).group(1) for line in f if ip_regex.match(line)]
11 counts = Counter(ips).most_common() # 多い順のリストが返る
12
13 # 2. Excel書き込み
14 wb = openpyxl.Workbook()
15 ws = wb.active
16 ws.append(["IPアドレス", "アクセス回数"])
17 for row in counts:
18     ws.append(row)
19
20 # 3. グラフ作成
21 chart = BarChart()
22 chart.title = "IPアクセス件数"
23 data = Reference(ws, min_col=2, min_row=1, max_row=len(counts)+1)
24 cats = Reference(ws, min_col=1, min_row=2, max_row=len(counts)+1)
25 chart.add_data(data, titles_from_data=True)
26 chart.set_categories(cats)
27 ws.add_chart(chart, "D2")
```

新しい処理ばかりであったためほとんどアイデアで触れてしまったが行ごとに説明していく。1~3行目は恒例のモジュールインポートである。2,3行目で少し違う書き方なのは必要な機能のみをfromに続くモジュールから抜き取っているためである。5行目は何度も行ったコマンドライン引数定義を2つ同時に行っている。この書き方もAIのコードから得られた知識である。6行目は正規表現でここは前実験課題のプログラムと全く同じである。9~10行目は以前に用いているwith文とreモジュールをfor文で回す処理である。これも前と同じ内容だがコンパクトになっている。group(1)部分も今までと同じでmatchしているか確認する際に正規表現で定義した()があるか確認しそこでIPアドレスを認識する、というものである。正規表現の方の()を消しgroup(0)とすることで同じ

ように動くが、文字コードによって前後の値が数値化され IP アドレスに接触するなどを危惧し () でくくっている。これも前実験と同様である、11 行目は新発見関数の Counter を用い、登場回数を ips から集計し IP アドレスと一緒に data の集合として格納している。さらに most_common() コマンドを用いることによってソートも同時に行っている。14 行目は前述した通りに Excel ファイルを開き、15 行目も前述した通りに初期位置のシート (セル) を指定している。16 行目は指定したところに” IP アドレス” と” アクセス回数” というデータの見出しを入力している。この入力では” アクセス回数” がセルの初期位置の 1 つ右のセルに入力され、指導書の指定通りにデータを格納する準備が整う。17,18 行目は Counter で格納されたデータの counts を for 文で行ごとに入力している。これで指導書の指定通りにデータが格納されるはずである。21 行目も前述した通り BarChart 関数で何も参照していない棒グラフを出力する。22 行目で棒グラフのタイトルを決定し、23,24 行目も前述した通り Reference を用いて参照するデータと横軸に使う値の参照範囲を決定している、ここで min_row (行) の最初に参照する値の場所が違うのは、数値データの 1 段目はタイトルとして扱われるという仕様だからである。そのため数値データであるアクセス回数 (登場回数) は 1 行先からデータを選択するのである。25,26 行目は指定した範囲のデータと軸に使用する値をグラフに入力している。27 行目で仕様が完全になった棒グラフを D2 に配置して処理が終了する。

2.7.3 結果

本実験課題の結果として、出力された ip.csv のスクリーンショットを図 2.12 に示す。

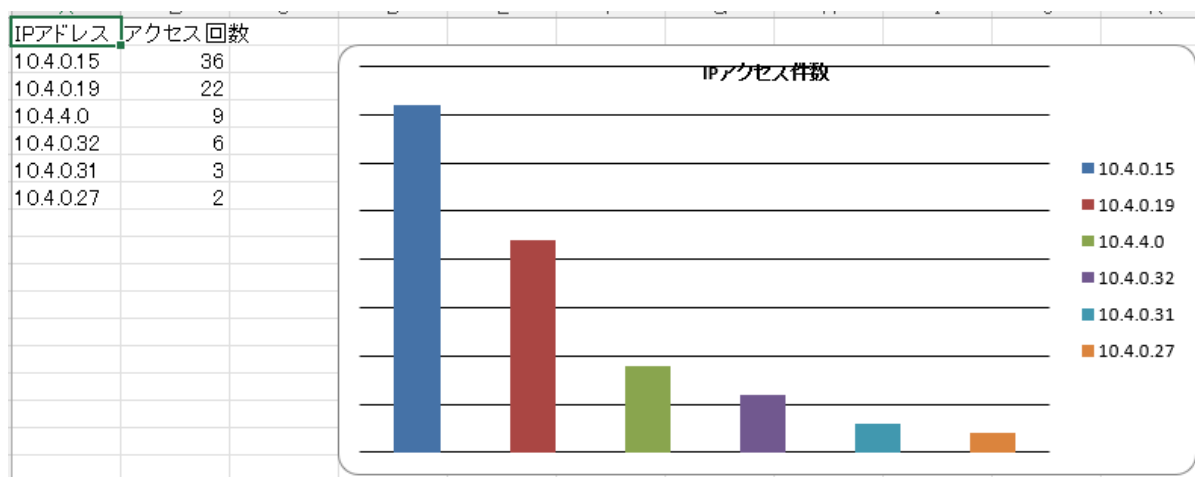


図 2.12 ip.csv

この結果は前課題で確認されたものと等しく、データとグラフの整合性もとれている。以上のことから本実験課題の題意を満たしたと言える。

2.8 発展課題

本実験課題では前課題で出力した棒グラフを matplotlib を用いて出力する。

2.8.1 アイデア

アイデア自体は前の実験とおおよそ変わらない。matplotlib を用いてグラフを出力するところのみが前課題と違うが、正直なところリストとして用いていた値を別々にして直でグラフを出す、というところくらいしか違いがない。後はほとんど同じようなコードを参考資料を見ながら書いていく。

2.8.2 プログラム

以下にアイデアに即したプログラムを示す。

matplotlib を用い棒グラフを出力するプログラム

```
1  import sys, re, matplotlib.pyplot as plt
2  from collections import Counter
3
4  ip_regex = re.compile(r'^(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})')
5  ips = []
6
7  with open(sys.argv[1], 'r', encoding='utf-8') as f:
8      for line in f:
9          match = ip_regex.match(line)
10         if match:
11             ips.append(match.group(1))
12
13     # 集計
14     counts = Counter(ips).most_common()
15     labels, values = zip(*counts)
16
17     # グラフ作成
18     plt.bar(labels, values)
19     plt.xticks(rotation=45)
20     plt.tight_layout()
21     plt.show()
```

14行目までは前課題と被るため説明は割愛する。15行目はCounterでまとめたcountsをIPアドレスだけの塊と回数だけの塊に分けている。これは前述した通りmatplotlibにそのまま値として参照させるためである。18行目は棒グラフを描画している。引数は順に横軸のデータ、縦軸のデータとなっている。19行目は横軸の見出しであるIPアドレスの文字を45度回転させている。そのままだと文字同士がかぶって大変なことになる（なった）ためである。20行目はグラフのタイトルやラベルがウィンドウ外にはみ出さないようにしている。21行目でグラフを1つのウィンドウとして出力する。

2.8.3 結果

プログラムを実行し出力されたウィンドウを図2.13に示す。

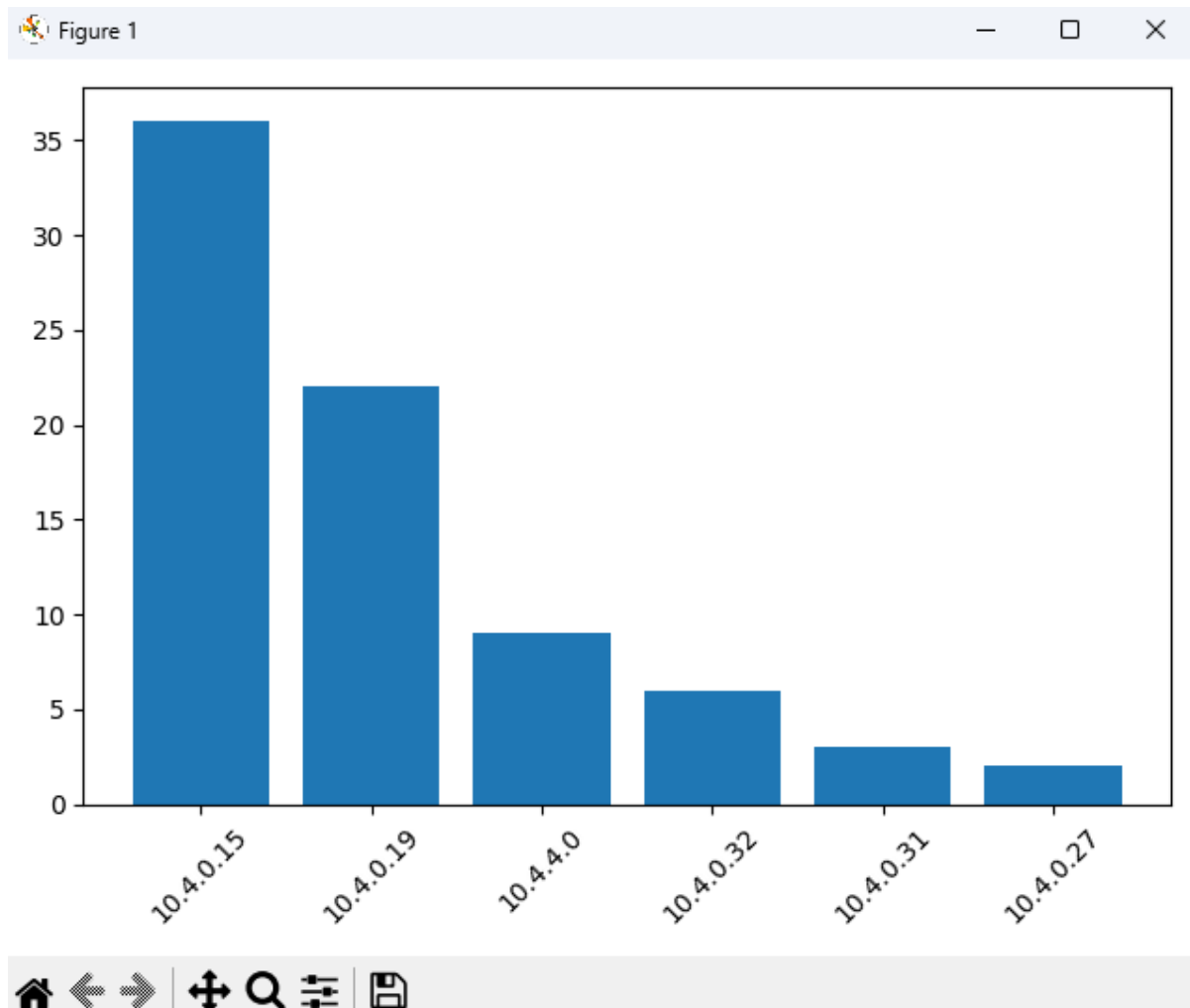


図 2.13 matplotlib で出力した棒グラフ

図2.13のグラフと図2.12のグラフの値が等しいためプログラムは正しく動作し課題の題意を満たせたと言える。

参考文献

- [1]Pythonでうるう年を判定・カウント・列挙, <https://note.nkmk.me/python-calendar-leap-year/>, 2026 年 01/11 参照.
- [2]Python, set 型で集合演算 (和集合、積集合や部分集合の判定など), <https://note.nkmk.me/python-set/>, 2026 年 01/11 参照.
- [3]【Python】正規表現を扱う re モジュール とは? 主要メソッドや活用シーンをわかりやすく解説!, <https://pythonaclarifydoubts.com/python-import-re/>, 2026 年 01/11 参照.
- [4]re — 正規表現操作, <https://docs.python.org/ja/3/library/re.html>, 2026 年 01/11 参照.
- [5]Python の Counter でリストの各要素の出現個数をカウント, <https://note.nkmk.me/python-collections-counter/>, 2026 年 01/11 参照.
- [6]Python で Excel 操作を自動化しよう! openpyxl & pandas (入門編), https://qiita.com/Yamashita_Sosuke/items/1e4d8a68d4058d1585a8, 2026 年 01/11 参照.
- [7]Python と openpyxl で Excel 作業を効率化! BarChart/BarChart3D を使ったグラフ作成入門, <https://tech-compass.jp/python/python%E3%81%A8openpyxl%E3%81%A7excel%E4%BD%9C%E6%A5%AD%E3%82%92%E5%8A%B9%E7%8E%87%E5%8C%96%EF%BC%81barchart-barchart3d%E3%82%92%E4%BD%BF%E3%81%A3%E3%81%9F%E3%82%B0%E3%83%A9%E3%83%95%E4%BD%9C%E6%88%90/>, 2026 年 01/11 参照.
- [8]【Python 初心者】Matplotlib の基本的な使い方まとめ, <https://qiita.com/gaku-devlog/items/5ccac4c2948dc17b5c4f>, 2026 年 01/11 参照.

感想

Python はインデントによる強制的な可読性の向上と記述スタイルの統一感により、他者のコードを解読する際のストレスが少ないと感じた。豊富なモジュール群によって最小限の記述で実装できる点は、開発効率の面で非常に強力である。特に、実行結果を即座に確認できるインタプリタとしての特性はデータの整理やスクレイピングといった日常的な問題にも適しており、将来的にデータ分析や機械学習を扱う際にも、その優位性は極めて高いと考えた。