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，検出を行った．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,というか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.

閏年判定プログラム

```
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 配列の中身を上書きで書き込んでいる．

ソートプログラム

```
data = []

file_path = input("ファイル：")
Rule = input("ルール（昇順：0,降順：1）")

with open(file_path, 'r', encoding='utf-8') as f:
    for line in f:
        parts = line.split(':')
        date = parts[0]
        temp = float(parts[1])
        data.append((date, temp))

#ソート処理（昇順: 0, 降順: 1)
# reverse=True で降順、False で昇順
is_reverse = True if Rule == "1" else False

#温度を基準にソート
data.sort(key=lambda x: x[1], reverse=is_reverse)

#出力処理
for date, temp in data:
    print(f"{date}: {temp}")

#ファイル書き換え（上書き）
with open(file_path, 'w', encoding='utf-8') as f:
    for date, temp in data:
        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．

コマンドライン引数を用いたプログラム

```
import sys

output_file = sys.argv[-1]
input_files = sys.argv[1:-1]

results = []

for filename in input_files:
    with open(filename, 'r', encoding='utf-8') as f:
        lines = f.readlines()

        l_count = len(lines)
        w_count = sum(len(line.split()) for line in lines)
        c_count = sum(len(line) for line in lines)

        # データの整形
        results.append(f"file: {filename}")
        results.append(f"lines: {l_count}")
        results.append(f"words: {w_count}")
        results.append(f"chars: {c_count}\n")

# --- 追加：cmd上への表示とファイル書き込み ---
output_text = "\n".join(results)
print(output_text)

# ファイルに保存
with open(output_file, 'w', encoding='utf-8') as f_out:
    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

```
raspberry raspberry
grape
peach
cherry cherry cherry
grape cherry strawberry
peach
apple
```

vegetable.dat

```
cherry mandarin orange apricot
apple banana apple
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 **総合課題** (6.10.6)

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.

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．