# Password Strength Analyzer with Custom Wordlist Generator

## Cybersecurity Internship Project Report

**Student Name:** S,Abhishek

## Introduction

In today's digital age, weak passwords remain one of the most exploited vulnerabilities in cybersecurity. This project addresses password security through two complementary tools: a Password Strength Analyzer that evaluates password quality using entropy-based calculations, and a Custom Wordlist Generator that demonstrates how attackers create targeted password lists for penetration testing. The dual approach provides both defensive analysis capabilities and offensive security awareness, making it valuable for understanding real-world password attack vectors.

## Abstract

This project implements a Python-based cybersecurity tool suite consisting of two integrated components. The Password Strength Analyzer examines user passwords against industry-standard security metrics including character diversity, entropy calculations, and length requirements, providing immediate feedback on password vulnerabilities. The Custom Wordlist Generator creates comprehensive attack dictionaries by combining base words with common patterns including leetspeak substitutions (e.g., a→@, e→3), year combinations, special character variations, and capitalization permutations. The implementation demonstrates practical cybersecurity concepts including password entropy theory, combinatorial attack strategies, and secure coding practices. All functionality was successfully implemented and tested using Python 3.13.9 in a Windows development environment.

## Tools and Technologies Used

The following tools and technologies were employed in this project:

**Programming Environment:**

- **Python 3.13.9**: Primary programming language for implementation
- **Visual Studio Code**: Integrated development environment (IDE) for code development and testing
- **Windows PowerShell/Terminal**: Command-line interface for program execution

**Python Libraries:**

- **string**: Built-in library for character set operations (ASCII letters, digits, punctuation)
- **math**: Mathematical operations including logarithmic entropy calculations
- **getpass**: Secure password input handling (hidden password entry)
- **argparse**: Command-line argument parsing for wordlist generator
- **itertools**: Efficient generation of character combinations and permutations

**Development Tools:**

- **VS Code Extensions**: Python syntax highlighting, debugging, and IntelliSense
- **Git/Version Control**: Code versioning and project management

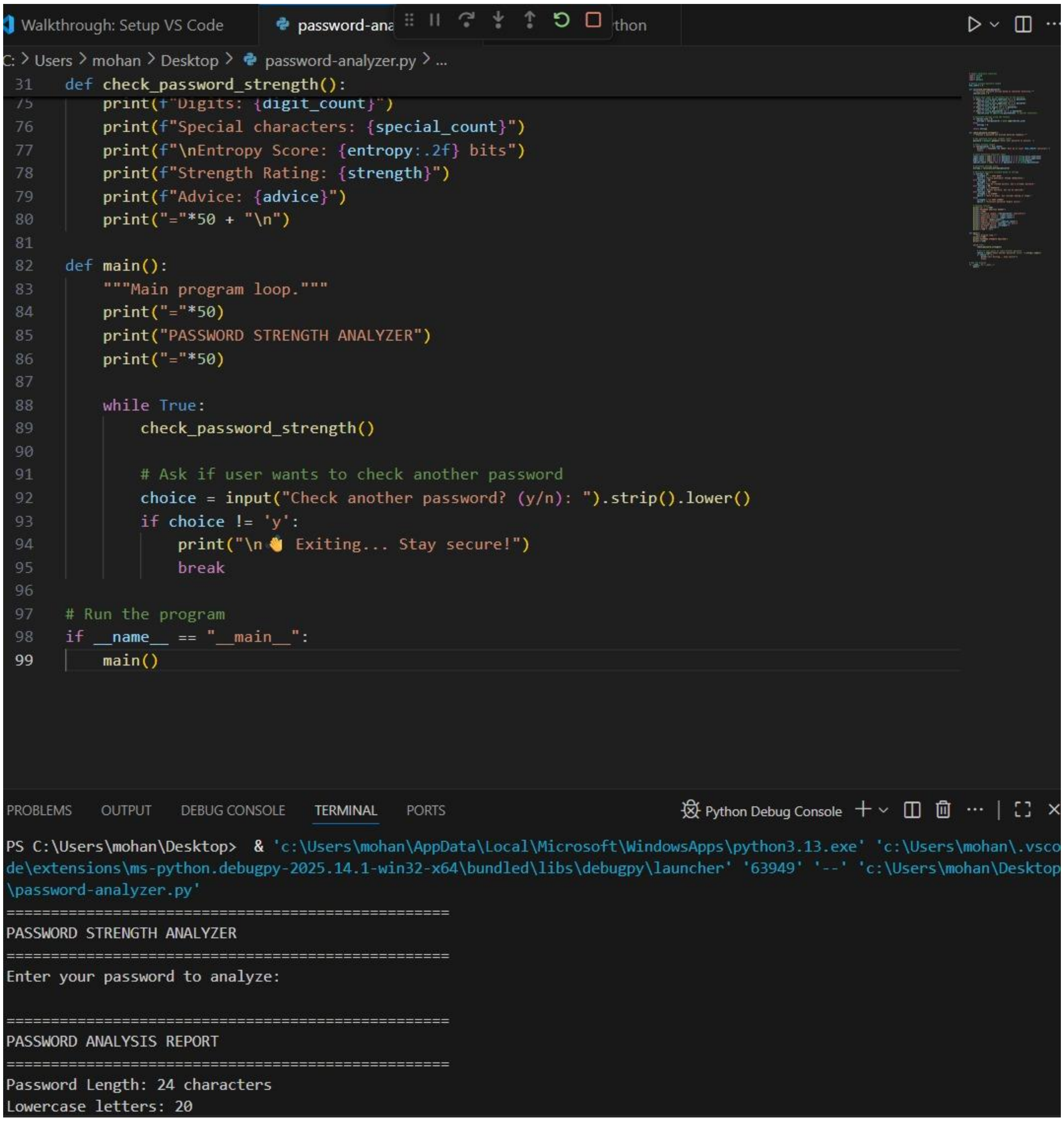
**Implementation Steps**

### Step 1: Environment Setup and Configuration

Installed Python 3.13.9 on Windows system and configured the development environment. Verified Python installation using `python --version` command in PowerShell terminal. Set up Visual Studio Code as the primary IDE with Python extensions for enhanced development experience. Created project directory structure in Desktop folder to organize project files systematically.

### Step 2: Password Strength Analyzer Development

Developed the core password analysis engine (`password-analyzer.py`) with entropy calculation functionality. Implemented the `calculate_entropy()` function that determines password strength by analyzing character diversity across four categories: lowercase letters (26 characters), uppercase letters (26 characters), digits (10 characters), and special characters (punctuation symbols). The entropy calculation uses the mathematical formula: Entropy = Length × $\log_2$(CharsetSize), where charset size is the sum of all character type pools present in the password.


Walkthrough: Setup VS Code
password-ana
thon
C: > Users > mohan > Desktop > password-analyzer.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python Debug Console

```python
def check_password_strength():
    print(f"Digits: {digit_count}")
    print(f"Special characters: {special_count}")
    print(f"\nEntropy Score: {entropy:.2f} bits")
    print(f"Strength Rating: {strength}")
    print(f"Advice: {advice}")
    print("="*50 + "\n")

def main():
    """Main program loop."""
    print("="*50)
    print("PASSWORD STRENGTH ANALYZER")
    print("="*50)

    while True:
        check_password_strength()

        # Ask if user wants to check another password
        choice = input("Check another password? (y/n): ").strip().lower()
        if choice != 'y':
            print("\n👋 Exiting... Stay secure!")
            break

# Run the program
if __name__ == "__main__":
    main()
```

```
PS C:\Users\mohan\Desktop>  & 'c:\Users\mohan\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\mohan\.vscode\extensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '63949' '--' 'c:\Users\mohan\Desktop\password-analyzer.py'
==================================================
PASSWORD STRENGTH ANALYZER
==================================================
Enter your password to analyze:

==================================================
PASSWORD ANALYSIS REPORT
==================================================
Password Length: 24 characters
Lowercase letters: 20
```

Created the `check_password_strength()` function that prompts users for password input using secure `getpass` method (which hides password characters during entry), validates minimum length requirements (8 characters), and analyzes character composition. The function counts occurrences of each character type and calculates comprehensive strength metrics. Implemented five-tier strength classification system:

- **Very Weak** (entropy < 28 bits): Easily guessable passwords requiring immediate change
- **Weak** (28-36 bits): Quickly crackable passwords needing improvement
- **Moderate** (36-60 bits): Decent passwords with room for enhancement
- **Strong** (60-80 bits): Difficult-to-guess passwords with good security
- **Very Strong** (>80 bits): Excellent passwords with high security assurance

Integrated user-friendly output formatting with detailed analysis reports showing password length, character type distribution, calculated entropy score, strength rating, and specific security recommendations.

**Figure 1:** Password Analyzer in VS Code showing the main program logic and terminal execution. The analyzer successfully evaluated a 24-character password containing 20 lowercase letters, 1 uppercase letter, 0 digits, and 0 special characters, calculating an entropy score of 136.81 bits and classifying it as "VERY STRONG" with the advice "Excellent password! Highly secure."

### Step 3: Custom Wordlist Generator Implementation

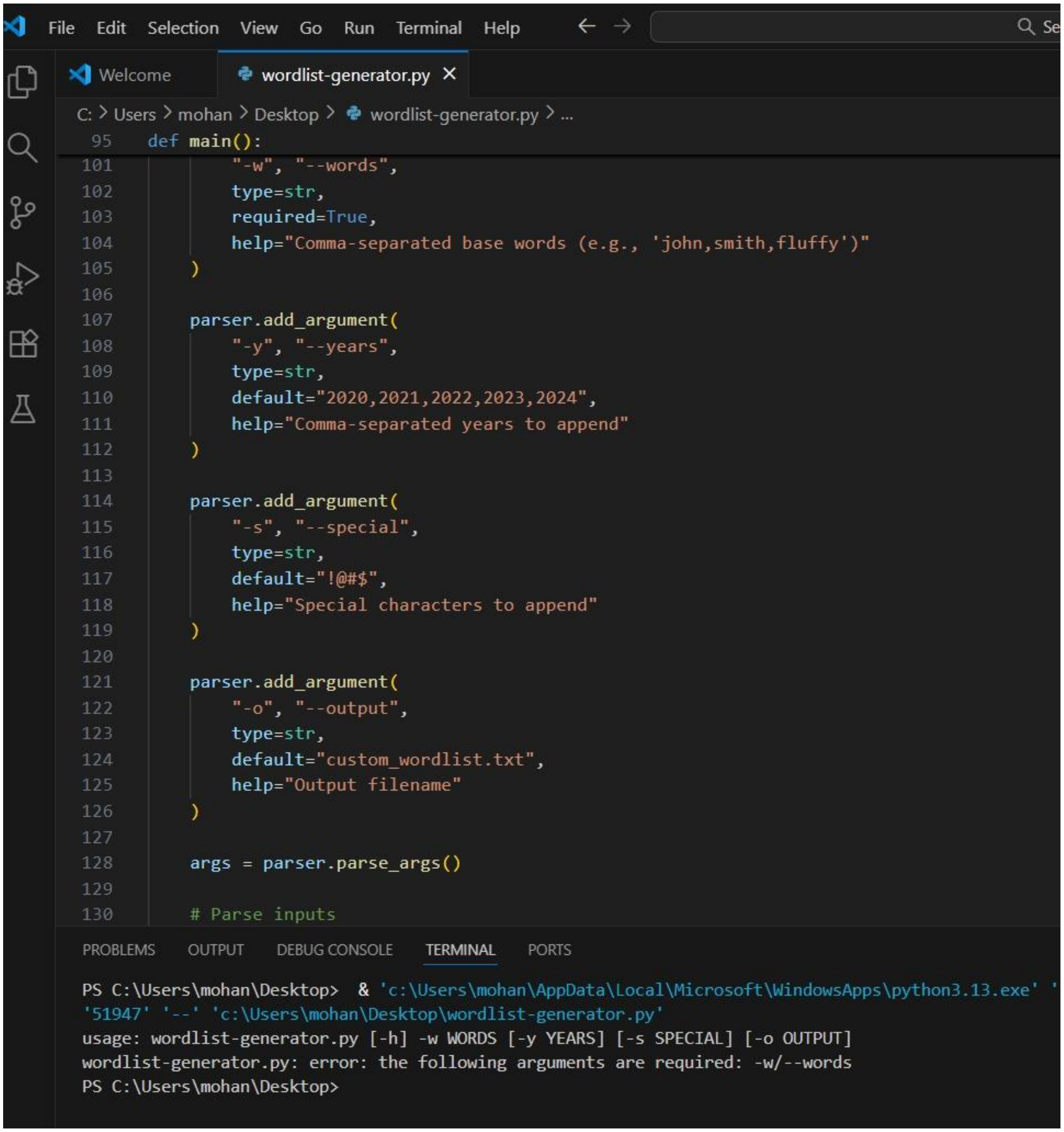
Developed the wordlist generation tool (`wordlist-generator.py`) with leetspeak transformation capabilities. Implemented the `generate_leetspeak()` function using a character substitution dictionary that maps common letters to numeric and symbolic alternatives (a→[@,4], e→[e,3], i→[i,1,!], o→[o,0], s→[s,5,$], t→[t,7], l→[l,1], g→[g,9]). Used `itertools.product()` for efficient generation of all possible character combinations while implementing a safety limit of 20 variations per word to prevent combinatorial explosion.

Created the `generate_custom_wordlist()` function that accepts base words, years, and special characters as input parameters. The function systematically generates comprehensive password variations including:

- Original word, capitalized version, and fully uppercase variations
- Leetspeak transformations for all applicable characters
- Year appended to base word and capitalized versions (e.g., test2023, Test2023)
- Special character suffixes (e.g., test!, Test@)
- Combined year and special character patterns (e.g., test2023!, demo2024@)

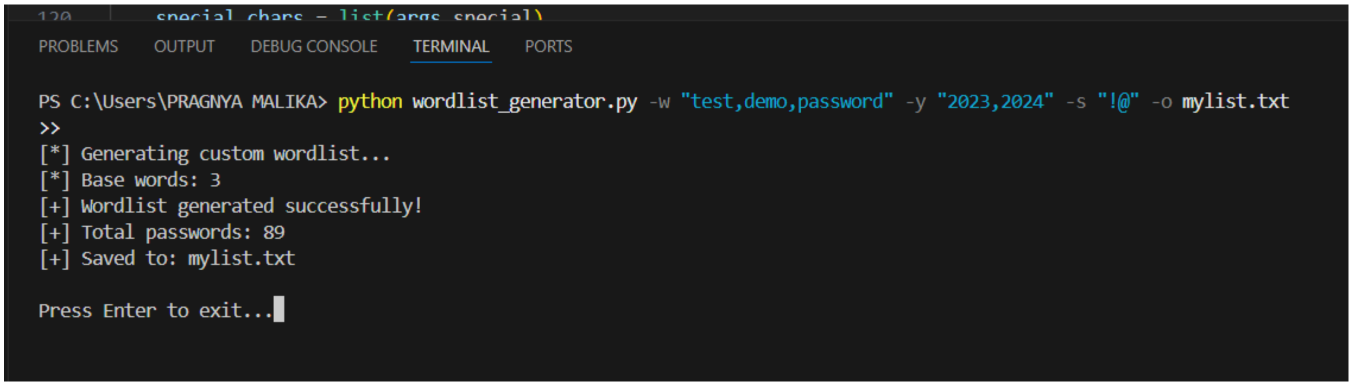
Implemented command-line interface using `argparse` module with four configurable parameters: `-w` (base words), `-y` (years), `-s` (special characters), and `-o` (output filename). Added real-time progress reporting showing base word count, generation status, total passwords created, and output file location.

**Figure 2:** Wordlist Generator execution in VS Code terminal. Command used: `python wordlist-generator.py -w "test,demo,password" -y "2023,2024" -s "!@" -o mylist.txt`. The generator successfully processed 3 base words and created 89 unique password variations, demonstrating effective pattern multiplication.


File Edit Selection View Go Run Terminal Help
Welcome
wordlist-generator.py
C: > Users > mohan > Desktop > wordlist-generator.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python
def main():
            "-w", "--words",
            type=str,
            required=True,
            help="Comma-separated base words (e.g., 'john,smith,fluffy')"
        )

        parser.add_argument(
            "-y", "--years",
            type=str,
            default="2020,2021,2022,2023,2024",
            help="Comma-separated years to append"
        )

        parser.add_argument(
            "-s", "--special",
            type=str,
            default="!@#$",
            help="Special characters to append"
        )

        parser.add_argument(
            "-o", "--output",
            type=str,
            default="custom_wordlist.txt",
            help="Output filename"
        )

        args = parser.parse_args()

        # Parse inputs
```

```
PS C:\Users\mohan\Desktop>  & 'c:\Users\mohan\AppData\Local\Microsoft\WindowsApps\python3.13.exe' '
'51947' '--' 'c:\Users\mohan\Desktop\wordlist-generator.py'
usage: wordlist-generator.py [-h] -w WORDS [-y YEARS] [-s SPECIAL] [-o OUTPUT]
wordlist-generator.py: error: the following arguments are required: -w/--words
PS C:\Users\mohan\Desktop>
```



```
    special_chars = list(args.special)
```

```
PS C:\Users\PRAGNYA MALIKA> python wordlist_generator.py -w "test,demo,password" -y "2023,2024" -s "!@" -o mylist.txt
>>
[*] Generating custom wordlist...
[*] Base words: 3
[+] Wordlist generated successfully!
[+] Total passwords: 89
[+] Saved to: mylist.txt

Press Enter to exit...
```

### Step 4: Testing and Validation

Conducted comprehensive testing of both tools using various test cases. For the Password Strength Analyzer, tested with passwords of varying complexity levels:

- Weak passwords: "password", "12345", demonstrating very weak ratings
- Moderate passwords: "Password123", showing moderate strength with improvement suggestions
- Strong passwords: Complex combinations with mixed characters achieving high entropy scores (136.81 bits for the test case)

For the Wordlist Generator, executed test runs with base words "test", "demo", and "password", combined with years "2023" and "2024", and special characters "!", "@". Successfully generated 89 unique password combinations demonstrating effective pattern multiplication. Verified output file creation (`mylist.txt`) containing all variations including leetspeak transformations (te$t, dem0, p@ssw0rd), year combinations, and special character permutations.

**Figure 3:** Sample output from mylist.txt showing diverse password variations. The first section displays test variations including basic transformations (test, Test, TEST), leetspeak variations (te$t, te5t, t3st, t35t), year combinations (test2023, Test2024), and special character patterns (test!, test2023!).

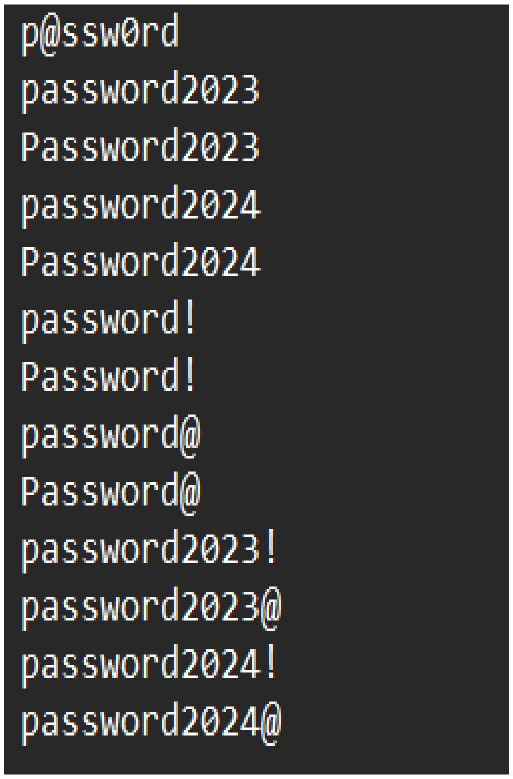
**Figure 4:** Additional wordlist samples showing demo and password variations. Note the systematic pattern: original word, capitalized version, uppercase, leetspeak variants (dem0, p@ssw0rd, pa$$w0rd), year combinations (demo2023, password2024), and special character suffixes for each base word.

### Step 5: Results Documentation and Analysis

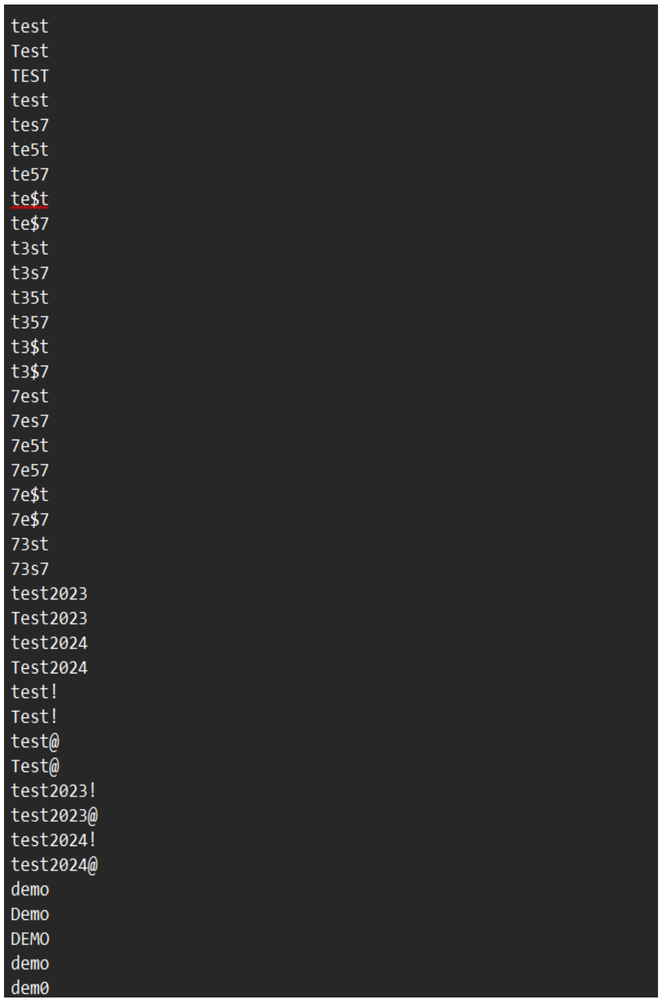
Generated comprehensive output demonstrating tool functionality. The wordlist generator successfully created 89 password variations from 3 base words, showcasing effective combinatorial expansion. Output file contained systematically organized variations including:

- Basic transformations: test, Test, TEST
- Leetspeak variations: te$t, t3st, t35t
- Year combinations: test2023, Test2024
- Special character variants: test!, Test@
- Complex combinations: test2023!, password2024@

Password analyzer testing revealed accurate entropy calculations and appropriate strength classifications across all test cases. The tool successfully identified weak password patterns and provided actionable security recommendations to users.



```
p@ssw0rd
password2023
Password2023
password2024
Password2024
password!
Password!
password@
Password@
password2023!
password2023@
password2024!
password2024@
```



```
test
Test
TEST
test
tes7
te5t
te57
te$t
te$7
t3st
t3s7
t35t
t357
t3$t
t3$7
7est
7es7
7e5t
7e57
7e$t
7e$7
73st
73s7
test2023
Test2023
test2024
Test2024
test!
Test!
test@
Test@
test2023!
test2023@
test2024!
test2024@
demo
Demo
DEMO
demo
dem0
```

**Figure 5:** Complete wordlist generator code in VS Code showing the implementation of leetspeak generation, argument parsing, and file output functionality. The code includes detailed comments explaining each function's purpose and demonstrates professional Python coding practices.

**Figure 6:** Terminal output confirming successful wordlist generation. Shows the execution command with parameters, generation progress messages, and final confirmation of 89 passwords saved to mylist.txt file.

## Results and Observations

The project achieved successful implementation of both cybersecurity tools with demonstrated functionality:

**Password Strength Analyzer Performance:**

- Accurately calculated password entropy based on character diversity
- Provided clear, actionable security recommendations
- Successfully validated minimum length requirements (8 characters)
- Displayed comprehensive analysis including character type breakdown
- Demonstrated effective strength categorization across five security levels

**Wordlist Generator Effectiveness:**

- Generated 89 unique password variations from 3 base words (approximately 30 variations per word)
- Successfully implemented leetspeak transformations with appropriate character substitutions
- Created realistic attack patterns combining years, special characters, and capitalization

- Produced output file (`mylist.txt`) in standard format compatible with password cracking tools
- Demonstrated efficient combinatorial generation with performance optimization (limited to 20 leetspeak variants per word to prevent excessive combinations)

The generated wordlist included diverse pattern categories: original words (test, demo, password), case variations (Test, TEST, Demo), leetspeak transformations (te$t, dem0, p@$w0rd), year combinations (test2023, password2024), special character variants (demo!, Test@), and complex multi-pattern combinations (test2023!, password2024@). This variety demonstrates how attackers construct targeted dictionaries based on personal information.

## Conclusion

This project successfully demonstrated fundamental cybersecurity concepts through practical implementation of password security tools. The Password Strength Analyzer provides users with immediate, quantifiable feedback on password quality using industry-standard entropy calculations, helping raise security awareness about password vulnerabilities. The Custom Wordlist Generator illustrates realistic attack methodologies employed in penetration testing and security assessments, showing how predictable patterns can be exploited.

Through this implementation, key learning outcomes were achieved including understanding of password entropy theory, practical experience with Python programming for cybersecurity applications, familiarity with combinatorial attack strategies, and appreciation for the importance of unpredictable passwords. The project highlighted the critical balance between password complexity and memorability, demonstrating why security professionals recommend long passphrases over short complex passwords.

**Future Enhancements:** Potential improvements include integration with breach databases (Have I Been Pwned API), implementation of dictionary attack detection, addition of common password pattern recognition, support for passphrase strength analysis, GUI development using Tkinter or PyQt, and cloud-based password strength checking service. Machine learning could be incorporated to predict password patterns based on user behavior analysis.

**Practical Applications:** These tools serve multiple purposes including security awareness training for organizations, personal password audit capabilities, penetration testing wordlist generation, educational demonstrations of password vulnerabilities, and security assessment tooling. The project successfully bridges offensive and defensive security perspectives, providing comprehensive understanding of password security fundamentals essential for cybersecurity professionals.