Title: Programming Language Concepts

Regex

UK postcodes typically follow these rules:

- One or two letters (A-Z), followed by a number (0-9), optionally followed by another letter or number.
- A space.
- A digit (0-9), followed by two letters (A-Z).

Valid examples:

- M1 1AA
- W1A 1HQ
- EC1A 1BB

Invalid example:

- ST7 9HV (does not match the format).

Creating the Regex:

Explanation:

- ^: Start of the string.
- [A-Z]{1,2}: One or two uppercase letters.
- [0-9]: A single digit.
- [A-Z0-9]?: An optional single letter or digit.
- ?: Matches a space.
- [0-9]: A single digit.
- [A-Z]{2}: Two uppercase letters.
- $: End of the string.

Python Code:

Users > george.koridze > Desktop > Essex > SSD > ePortfolio > Unit4 > Regex_Postal_Codes.py > validate_postcode

```python
import re

# Regex for UK postcode validation
postcode_regex = r"^((([A-Z]{1,2}[0-9][0-9A-Z]?) ?[0-9][A-Z]{2})|(GIR ?0AA))$"

# Function to validate postcodes
def validate_postcode(postcode):
    if re.match(postcode_regex, postcode):
        return True
    return False

# List of test cases
postcodes = [
    "M1 1AA",     # Valid
    "M60 1NW",    # Valid
    "CR2 6XH",    # Valid
    "DN55 1PT",   # Valid
    "W1A 1HQ",    # Valid
    "EC1A 1BB",   # Valid
    "ST7 9HV",    # Invalid
]

# Validate and display results
for postcode in postcodes:
    result = validate_postcode(postcode)
    print(f"Postcode: {postcode} -> {'Valid' if result else 'Invalid'}")

```

Output:

```
george.koridze@MBP-GK-QQXJPGK7P4 ~ % /usr/local/bin/python3 /Users/george.koridze/Desktop/Essex/SSD/ePortfolio/Unit4/Regex_Postal_Codes.py
Postcode: M1 1AA -> Valid
Postcode: M60 1NW -> Valid
Postcode: CR2 6XH -> Valid
Postcode: DN55 1PT -> Valid
Postcode: W1A 1HQ -> Valid
Postcode: EC1A 1BB -> Valid
Postcode: ST7 9HV -> Valid
```

Evil Regex Prevention:

To prevent **evil regex attacks**, which exploit poorly written regex patterns to consume excessive CPU/memory, follow these practices:

1. Use simple and efficient regex patterns.
    - Avoid nested quantifiers or backtracking.
    - Use specific character classes ([A-Z], [0-9]) instead of overly broad patterns like .*.
2. Set input limits.
    - Restrict the length of the string to a maximum (e.g., 8 characters for UK postcodes).
3. Use libraries like re2 for regex execution (if available) as they avoid huge backtracking.
4. Test the regex against long and edge-case inputs to verify the performance