```python
def sum_of_multiples(limit):
    total_sum = 0
    for number in range(limit):
        if number % 3 == 0 or number % 5 == 0:
            total_sum += number
    return total_sum

limit = 1000

result = sum_of_multiples(limit)

print(result)
```

```
233168
```

```python
def largest_prime_factor(n):

    largest_factor = None

    while n % 2 == 0:
        largest_factor = 2
        n //= 2

    factor = 3
    while factor * factor <= n:
        if n % factor == 0:
            largest_factor = factor
            while n % factor == 0:
                n //= factor
        factor += 2

    if n > 2:
        largest_factor = n
    return largest_factor

number = 600851475143
result = largest_prime_factor(number)
print(result)
```

```
6857
```

```python
import math
from functools import reduce

def lcm(a, b):
    return abs(a * b) // math.gcd(a, b)

def lcm_multiple(numbers):
    return reduce(lcm, numbers)

numbers = range(1, 21)
```

```
result = lcm_multiple(numbers)
print(result)
```

```
232792560
```

```
def is_prime(num):
    if num <= 1:
        return False
    if num <= 3:
        return True
    if num % 2 == 0 or num % 3 == 0:
        return False
    i = 5
    while i * i <= num:
        if num % i == 0 or num % (i + 2) == 0:
            return False
        i += 6
    return True
def find_nth_prime(n):
    count = 0
    num = 2
    while count < n:
        if is_prime(num):
            count += 1
        num += 1
    return num - 1
position = 10001
result = find_nth_prime(position)
print(result)
```

```
104743
```

```
def sieve_of_eratosthenes(limit):
    is_prime = [True] * limit
    p = 2
    while (p * p <= limit):
        if (is_prime[p] == True):
            for i in range(p * p, limit, p):
                is_prime[i] = False
        p += 1
    return [p for p in range(2, limit) if is_prime[p]]
def sum_of_primes_below(limit):
    primes = sieve_of_eratosthenes(limit)
    return sum(primes)
limit = 2000000
result = sum_of_primes_below(limit)
print(result)
```

```
142913828922
```

```
import math
import re

class Calculator:
    def __init__(self):
        self.operators = {
            '+': self.add,
            '-': self.subtract,
            '*': self.multiply,
            '/': self.divide,
            '^': self.exponent
        }

    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b

    def multiply(self, a, b):
        return a * b

    def divide(self, a, b):
        if b == 0:
            raise ZeroDivisionError("Cannot divide by zero")
        return a / b

    def exponent(self, a, b):
        return a ** b

    def parse_expression(self, expression):
        try:

            expression = expression.replace(' ', '')
            expression = expression.replace('^', '**')
            result = eval(expression, {"__builtins__": None},
self.operators)
            return result
        except ZeroDivisionError as e:
            return str(e)
        except Exception as e:
            return f"Error: {e}"

    def calculate(self, expression):
        # Replace "^" with "**" to make use of Python's exponentiation
        expression = re.sub(r'(\d+)\^(\d+)', r'\1**\2', expression)
        # Parse and evaluate the expression
        return self.parse_expression(expression)

def main():
```

```
    calc = Calculator()
    print("Welcome to the command-line calculator!")
    print("Supported operations: +, -, *, /, ^ (exponentiation)")
    print("Example usage: (2 + 3) * 4 ^ 2")

    while True:
        try:
            expression = input("Enter expression (or type 'exit' to
quit): ")
            if expression.lower() == 'exit':
                break
            result = calc.calculate(expression)
            print(f"Result: {result}")
        except Exception as e:
            print(f"An error occurred: {e}")

if __name__ == "__main__":
    main()
```

```
Welcome to the command-line calculator!
Supported operations: +, -, *, /, ^ (exponentiation)
Example usage: (2 + 3) * 4 ^ 2

Enter expression (or type 'exit' to quit):  exit
```

```
import re

def validate_password(password):
    blacklisted_passwords = ["A1b#cD3e", "Xy4$Zz7!", "P@ssw0rd", "M!
n3r4L^", "T7r$eN8f"]
    if password in blacklisted_passwords:
        return "Error: Password is blacklisted."

    # Check length
    if len(password) != 8:
        return "Error: Password must be exactly 8 characters long."
    if re.match(r'^[0-9\W]', password):
        return "Error: Password should not start with a number or
special character."

    if not re.search(r'[A-Z]', password):
        return "Error: Password must contain at least one uppercase
letter."
    if not re.search(r'[a-z]', password):
        return "Error: Password must contain at least one lowercase
letter."
    if not re.search(r'[!@#$%^&*()_+{}\[\]:;"\'<>,.?/\\|`~]',
password):
        return "Error: Password must contain at least one special
character."
```

```python
    # If all checks pass
    return "Password is valid."

# Example usage
test_passwords = [
    "A1b#cD3e",  # Blacklisted
    "Xy4$Zz7!",  # Blacklisted
    "P@ssw0rd",  # Blacklisted
    "M!n3r4L^",  # Blacklisted
    "T7r$eN8f",  # Blacklisted
    "aB1#cD2e",  # Valid
    "1aA@bcDE",  # Valid
    "Abc123!!",  # Invalid - length
    "!@#$%^&*",  # Invalid - starts with special char
    "Abcdefgh",  # Invalid - missing special char
    "Ab1@efgh",  # Valid
]

for pwd in test_passwords:
    print(f"Testing password '{pwd}': {validate_password(pwd)}")
```

```
Testing password 'A1b#cD3e': Error: Password is blacklisted.
Testing password 'Xy4$Zz7!': Error: Password is blacklisted.
Testing password 'P@ssw0rd': Error: Password is blacklisted.
Testing password 'M!n3r4L^': Error: Password is blacklisted.
Testing password 'T7r$eN8f': Error: Password is blacklisted.
Testing password 'aB1#cD2e': Password is valid.
Testing password '1aA@bcDE': Error: Password should not start with a
number or special character.
Testing password 'Abc123!!': Password is valid.
Testing password '!@#$%^&*': Error: Password should not start with a
number or special character.
Testing password 'Abcdefgh': Error: Password must contain at least one
special character.
Testing password 'Ab1@efgh': Password is valid.
```

```python
import random

class TicTacToe:
    def __init__(self):
        self.board = [' ' for _ in range(9)]  # Board is a list of 9
spaces
        self.current_player = 'X'  # Player 'X' starts
        self.winner = None

    def print_board(self):
        print("\n")
        print(f"{self.board[0]} | {self.board[1]} | {self.board[2]}")
        print("--+---+--")
```

```
        print(f"{self.board[3]} | {self.board[4]} | {self.board[5]}")
        print("--+---+--")
        print(f"{self.board[6]} | {self.board[7]} | {self.board[8]}")
        print("\n")

    def is_winner(self, player):
        win_conditions = [
            [0, 1, 2], [3, 4, 5], [6, 7, 8],  # Rows
            [0, 3, 6], [1, 4, 7], [2, 5, 8],  # Columns
            [0, 4, 8], [2, 4, 6]              # Diagonals
        ]
        for condition in win_conditions:
            if all(self.board[i] == player for i in condition):
                return True
        return False

    def is_full(self):
        return ' ' not in self.board

    def make_move(self, position, player):
        if self.board[position] == ' ':
            self.board[position] = player
            if self.is_winner(player):
                self.winner = player
            elif self.is_full():
                self.winner = 'D'  # Draw
            return True
        else:
            return False

    def player_move(self):
        while True:
            try:
                position = int(input("Enter your move (1-9): ")) - 1
                if 0 <= position <= 8:
                    if self.make_move(position, 'X'):
                        break
                    else:
                        print("Invalid move. The position is already
occupied.")
                else:
                    print("Invalid position. Choose a number between 1
and 9.")
            except ValueError:
                print("Invalid input. Please enter a number between 1
and 9.")

    def computer_move(self):
        available_moves = [i for i, spot in enumerate(self.board) if
spot == ' ']
```

```
        position = random.choice(available_moves)
        self.make_move(position, 'O')

    def play_game(self):
        print("Welcome to Tic-Tac-Toe!")
        self.print_board()

        while not self.winner:
            if self.current_player == 'X':
                self.player_move()
                self.current_player = 'O'
            else:
                self.computer_move()
                self.current_player = 'X'
            self.print_board()

        if self.winner == 'D':
            print("It's a draw!")
        else:
            print(f"Player {self.winner} wins!")

if __name__ == "__main__":
    game = TicTacToe()
    game.play_game()
```

```
Welcome to Tic-Tac-Toe!


  |   |
--+---+--
  |   |
--+---+--
  |   |


Enter your move (1-9):  4


  |   |
--+---+--
X |   |
--+---+--
  |   |



  |   |
--+---+--
```

```
X |   |
--+---+--
  |   | O


Enter your move (1-9):  3


  |   | X
--+---+--
X |   |
--+---+--
  |   | O



O |   | X
--+---+--
X |   |
--+---+--
  |   | O


Enter your move (1-9):  6


O |   | X
--+---+--
X |   | X
--+---+--
  |   | O



O |   | X
--+---+--
X | O | X
--+---+--
  |   | O


Player O wins!
```