```
 import random
def generate_number():
    return str(random.randint(1000, 9999))


def count_cows_and_bulls(secret_number, guess):
    cows = 0
    bulls = 0
    for i in range(4):
        if guess[i] == secret_number[i]:
            cows += 1
        elif guess[i] in secret_number:
            bulls += 1
    return cows, bulls


def main():
    secret_number = generate_number()
    guesses = 0
    print("Welcome to the Cows and Bulls Game!")


    while True:
        user_guess = input("Enter a 4-digit number: ")


        if len(user_guess) != 4 or not user_guess.isdigit():
            print("Please enter a valid 4-digit number.")
            continue


        guesses += 1
        cows, bulls = count_cows_and_bulls(secret_number, user_guess…
Creating Variables with Different Naming Conventions:
snake_case_variable = 42
camelCaseVariable = "Hello, world!"
```

```
PascalCaseVariable = [1, 2, 3]

CONSTANT_VARIABLE = "This won't change"
```

Reversing a String using Slicing:

```
original_string = "Hello, world!"

reversed_string = original_string[::-1]

print(reversed_string)  # Outputs: "!dlrow ,olleH"
```

Exploring dir() and Using help():

The dir() function returns a list of names in the current local scope or a specified object's attributes. The help() function provides documentation about a particular Python object.

Let's explore the attributes of the str (string) class:

```
# Explore dir() and help()

string_attributes = dir(str)

# Print out the first 10 attributes

print(string_attributes[:10])

# Let's pick an attribu…
```

String Operations:

```
# Creating a string

name = "some name"

# Convert to upper case

upper_case_name = name.upper()

# Convert to lower case

lower_case_name = name.lower()

# Capitalize the string
```

```
capitalized_name = name.capitalize()

# Replace 'e' with 'E'
replaced_name = name.replace('e', 'E')

print("Original Name:", name)
print("Upper Case:", upper_case_name)
print("Lower Case:", lower_case_name)
print("Capitalized:", capitalized_name)
print("Replaced:", replaced_name)
```

List Operations:

```
# Creating a list
L = [1, 2, 3]

# Extend the list
L.extend([5, 6, 7])

# Remove the 5th value (index 4)
del L[4]

print("Extended List:", L)
```

Dictionary Operations:

```
# Creating a dictionary
d = {'mango': 10, 'banana': 0, 'apple': 15, 'orange': 0, 'pineapple': 20}

# Remove …
```

Concatenation Examples:

String Concatenation:

```
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print("Full Name:", full_name)
```

List Concatenation:

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
combined_list = list1 + list2
print("Combined List:", combined_list)
```

Tuple Concatenation:

```
tuple1 = (10, 20)
tuple2 = (30, 40)
combined_tuple = tuple1 + tuple2
print("Combined Tuple:", combined_tuple)
```

Formatting Techniques Examples:

Using f-strings:

```
name = "Alice"
age = 30
formatted_string = f"My name is {name} and I am {age} years old."
print(formatted_string)
```

Using .format():

```
product = "Widget"
price = 19.99
formatted_string = "The {} costs ${:.2f}.".format(product, price)
print(formatted_string)
```

Using % Formatting (Old Style):

name = "Bob"

s…

Power of a Number:

You can use the * operator to calculate the power of a number. For example, a * b means "a raised to the power of b".

base = 2

exponent = 3

result = base ** exponent

print(f"{base} raised to the power of {exponent} is {result}")

Left and Right Shift:

Left shift (<<) and right shift (>>) operators are used to shift the bits of a binary number to the left or right by a specified number of positions.

number = 8 # Binary: 1000

left_shifted = number << 2 # Shift left by 2 positions: 100000

right_shifted = number >> 2 # Shift right by 2 positions: 10

print("Left Shifted:", left_shifted)

print("Right Shifted:", right_shifted)

Bitwise AND Operator (&):

The & operator performs a bitwise AND operation between the individual bits of two number…

Refer .capitalize() and Replicate .upper() and .lower() Functions:

text = "hello world"

capitalized_text = text.capitalize()

upper_text = text.upper()

lower_text = text.lower()

print("Original Text:", text)

```
print("Capitalized:", capitalized_text)
print("Upper Case:", upper_text)
print("Lower Case:", lower_text)
```

Create an Odd Sequence:

```
sequence = [1, 2, 34, 65, 1, 2, 65, 66, 44, 33, 22, 87, 123412, 9, 78, 76]

odd_sequence = [num for num in sequence if num % 2 != 0]
print("Odd Sequence:", odd_sequence)
```

Filter Fruits with More Than 20:

```
fruits = {'apple': 10, 'mango': 20, 'pineapple': 25, 'orange': 30, 'strawberry': 50, 'jackfruit': 10}

fruits_more_than_20 = {fruit: quantity for fruit, quantity in fruits.items() if quantity > 20}
print("Fruits with More Than 20:", fruits_more_than_20)
```

Replicate sum() Function:

```
def custom_sum(numbers):
    total = 0
    for num in numbers:
        total += num
    return total

numbers = [1, 2, 3, 4, 5]
result = custom_sum(numbers)
print("Custom Sum:", result)
```

Replicate String Attributes: ljust() and rjust():

```
def custom_ljust(text, width, fillchar=' '):
    if len(text) >= width:
        return text
    else:
```

```
        return text + fillchar * (width - len(text))

def custom_rjust(text, width, fillchar=' '):
    if len(text) >= width:
        return text
    else:
        return fillchar * (width - len(text)) + text

text = "Hello"
width = 10
left_aligned = custom_ljust(text, width, '*')
right_aligned = custom_rjust(text, width, '-')

print("Left Justified:", left_aligned)
print("Right Justified:", right_al…
```

Refer .capitalize() and Replicate .upper() and .lower() Functions:

```
# Refer capitalize function and replicate .upper() and .lower() functions
text = "hello, world!"

capitalized_text = text.capitalize()
upper_text = text.upper()
lower_text = text.lower()

print("Original Text:", text)
print("Capitalized Text:", capitalized_text)
print("Upper Case Text:", upper_text)
print("Lower Case Text:", lower_text)
```

Create an Odd Sequence:

```
# Create an odd sequence from the given list
sequence = [1, 2, 34, 65, 1, 2, 65, 66, 44, 33, 22, 87, 123412, 9, 78, 76]
```

```
odd_sequence = [num for num in sequence if num % 2 != 0]
print("Odd Sequence:", odd_sequence)
```

Filter Fruits with More Than 20:

```
# Filter fruits with more than 20 quantity using a dictionary comprehension
fruits = {'a…
```

Replicate sum() Function:

```
def custom_sum(numbers):
    total = 0
    for num in numbers:
        total += num
    return total

numbers = [1, 2, 3, 4, 5]
result = custom_sum(numbers)
print("Custom Sum:", result)
```

Replicate String Attributes ljust() and rjust():

```
def custom_ljust(text, width, fillchar=' '):
    if len(text) >= width:
        return text
    else:
        return text + fillchar * (width - len(text))

def custom_rjust(text, width, fillchar=' '):
    if len(text) >= width:
        return text
    else:
        return fillchar * (width - len(text)) + text
```

```
text = "Hello"
width = 10
left_aligned = custom_ljust(text, width, '*')
right_aligned = custom_rjust(text, width, '-')

print("Left Justified:", left_aligned)
print("Right Justified:", right_ali…
```

Generator Function to Replicate range():

```
def custom_range(start, stop, step=1):
    current = start
    while current < stop:
        yield current
        current += step

for num in custom_range(2, 10, 2):
    print(num)
```

Recursive Function to Replicate range():

```
def custom_recursive_range(start, stop, step=1):
    if start >= stop:
        return []
    else:
        return [start] + custom_recursive_range(start + step, stop, step)

for num in custom_recursive_range(2, 10, 2):
    print(num)
```

Recursive and Lambda Function for Greatest Common Divisor (GCD):

```
gcd_recursive = lambda a, b: a if not b else gcd_recursive(b, a % b)
```

```
# Example usage
print(gcd_recursive(48, 18))
```

Creating a module using an Editor / IDE:

To create a module, you'll need to follow the…

Create a module named mymathmodule.py with palindrome, Fibonacci, and factorial functions:

```
# mymathmodule.py

def is_palindrome(s):
    s = s.lower().replace(" ", "")  # Convert to lowercase and remove spaces
    return s == s[::-1]  # Check if the string is equal to its reverse

def fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        fib_sequence = [0, 1]
        while len(fib_sequence) < n:
            fib_sequence.append(fib_sequence[-1] + fib_sequence[-2])
        return fib_sequence

def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

Create another module to invoke the functions from mymathmodule:

mainmodule....

Single Inheritance:

```
class Parent:
    def parent_method(self):
        print("This is the parent class method.")

class Child(Parent):
    def child_method(self):
        print("This is the child class method.")

# Creating an object of the Child class
child_obj = Child()
child_obj.parent_method()
child_obj.child_method()
```

Multiple Inheritance:

```
class Parent1:
    def method1(self):
        print("This is method 1 from Parent1.")

class Parent2:
    def method2(self):
        print("This is method 2 from Parent2.")

class Child(Parent1, Parent2):
    def child_method(self):
        print("This is the child class method.")

# Creating an object of the Child class
```

```
child_obj = Child()
child_obj.method1()
child_obj.method2()
child_obj.child_method()
```

Multilevel I…

```
def main():
    try:
        # NameError example
        print(undefined_variable)
    except NameError:
        print("NameError: The variable is not defined.")

    try:
        # TypeError example
        x = "five"
        y = 2
        result = x + y
    except TypeError:
        print("TypeError: Unsupported operation between different data types.")

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

In this program, we have two separate try-except blocks. The first block attempts to print an undefined variable undefined_variable, which raises a NameError. The exception is caught, and a custom message is printed indicating that the variable is not defined.

The second block attempts to add a string and an integer (x + y), which would result in a TypeError. The excep…