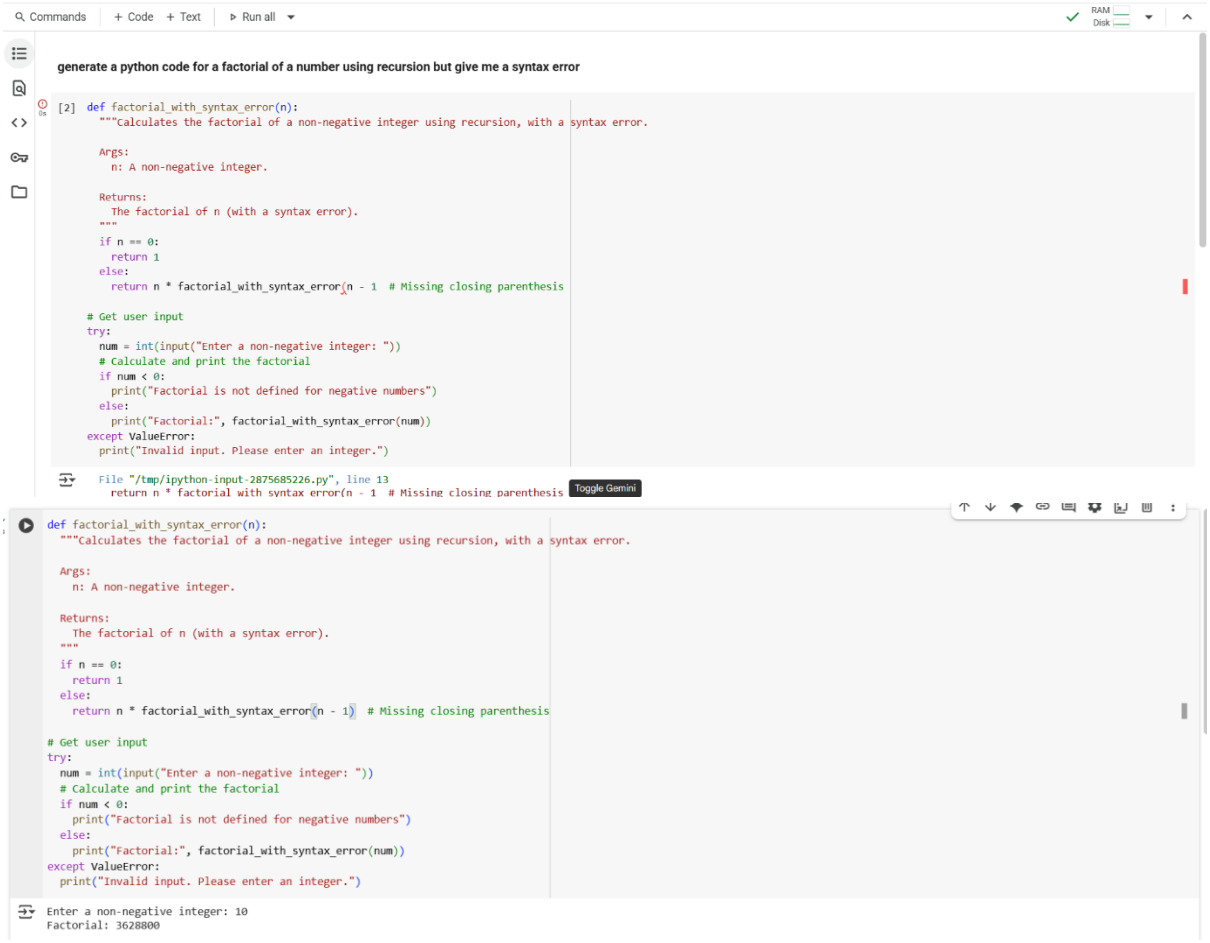
TASK 1:

PROMPT:

generate a python code for a factorial of a number using recursion but give me a syntax error.



generate a python code for a factorial of a number using recursion but give me a syntax error

```
[2] def factorial_with_syntax_error(n):
      """Calculates the factorial of a non-negative integer using recursion, with a syntax error.

      Args:
        n: A non-negative integer.

      Returns:
        The factorial of n (with a syntax error).
      """
      if n == 0:
        return 1
      else:
        return n * factorial_with_syntax_error(n - 1  # Missing closing parenthesis

    # Get user input
    try:
      num = int(input("Enter a non-negative integer: "))
      # Calculate and print the factorial
      if num < 0:
        print("Factorial is not defined for negative numbers")
      else:
        print("Factorial:", factorial_with_syntax_error(num))
    except ValueError:
      print("Invalid input. Please enter an integer.")
```

```
  File "/tmp/ipython-input-2875685226.py", line 13
    return n * factorial_with_syntax_error(n - 1  # Missing closing parenthesis
```

Toggle Gemini

```
def factorial_with_syntax_error(n):
  """Calculates the factorial of a non-negative integer using recursion, with a syntax error.

  Args:
    n: A non-negative integer.

  Returns:
    The factorial of n (with a syntax error).
  """
  if n == 0:
    return 1
  else:
    return n * factorial_with_syntax_error(n - 1)  # Missing closing parenthesis

# Get user input
try:
  num = int(input("Enter a non-negative integer: "))
  # Calculate and print the factorial
  if num < 0:
    print("Factorial is not defined for negative numbers")
  else:
    print("Factorial:", factorial_with_syntax_error(num))
except ValueError:
  print("Invalid input. Please enter an integer.")
```

```
Enter a non-negative integer: 10
Factorial: 3628800
```

TASK 2 :

PROMPT:

generate a Python code to sort the list [3, "10", 5, "2", 8]. The current version fails with a TypeError because of mixed integers and strings. Detect the issue and fix it so the list sorts consistently.

```
data = [3, "10", 5, "2", 8]

# Syntax error: missing colon in function definition
def sort_list(lst)
    return sorted(lst, key=int)

print(sort_list(data))
```

```
  File "/tmp/ipython-input-1674977981.py", line 4
    def sort_list(lst)
                      ^
SyntaxError: expected ':'
```

```
data = [3, "10", 5, "2", 8]

# Syntax error: missing colon in function definition
def sort_list(lst):
    return sorted(lst, key=int)

print(sort_list(data))
```

```
['2', 3, 5, 8, '10']
```

TASK 3:

PROMPT:

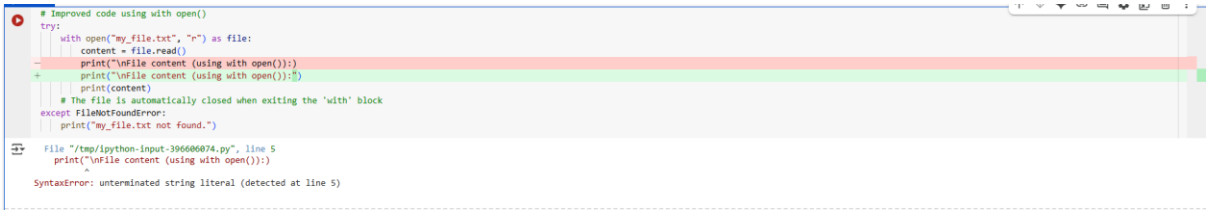
generate a Python snippet that opens a file for reading but forgets to close it. Then, improve the code using best practices for file handling (like using a with open() block).

```
# Create a dummy file for demonstration
with open("my_file.txt", "w") as f:
    f.write("This is a sample file.\n")
    f.write("It has a few lines.\n")

# Example of opening a file without closing
try:
-   file = open("my_file.txt", "r"
+   file = open("my_file.txt", "r")
    content = file.read()
    print("File content (without closing):")
    print(content)
    # The file is not explicitly closed here, which is a potential issue
except FileNotFoundError:
    print("my_file.txt not found.")
```

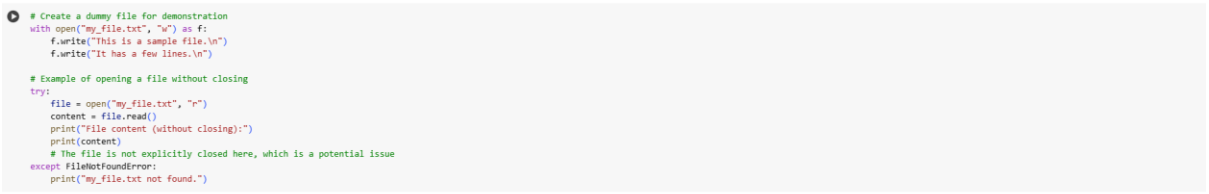
```
  File "/tmp/ipython-input-585883796.py", line 8
    file = open("my_file.txt", "r"
                                  ^
SyntaxError: '(' was never closed
```

Next steps: Explain error



```
# Improved code using with open()
try:
    with open("my_file.txt", "r") as file:
        content = file.read()
        print("\nFile content (using with open()):")
        print(content)
    # The file is automatically closed when exiting the 'with' block
except FileNotFoundError:
    print("my_file.txt not found.")
```

```
  File "/tmp/ipython-input-396606074.py", line 5
    print("\nFile content (using with open()):")
          ^
SyntaxError: unterminated string literal (detected at line 5)
```



```
# Create a dummy file for demonstration
with open("my_file.txt", "w") as f:
    f.write("This is a sample file.\n")
    f.write("It has a few lines.\n")

# Example of opening a file without closing
try:
    file = open("my_file.txt", "r")
    content = file.read()
    print("File content (without closing):")
    print(content)
    # The file is not explicitly closed here, which is a potential issue
except FileNotFoundError:
    print("my_file.txt not found.")
```

```
File content (without closing):
This is a sample file.
It has a few lines.
```

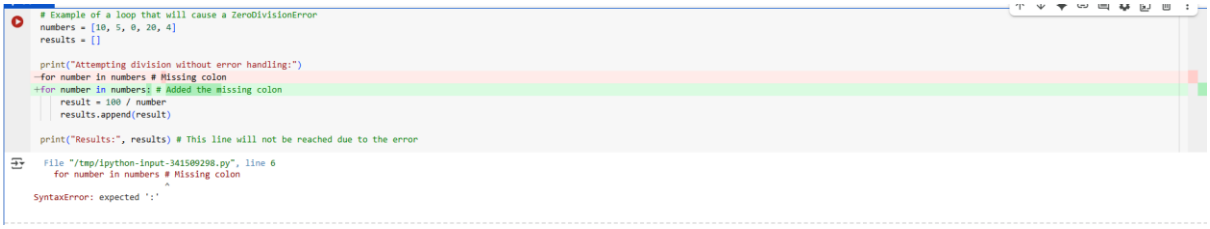
```
# Improved code using with open()
try:
    with open("my_file.txt", "r") as file:
        content = file.read()
        print("\nFile content (using with open()):")
        print(content)
    # The file is automatically closed when exiting the 'with' block
except FileNotFoundError:
    print("my_file.txt not found.")
```

```
File content (using with open()):
This is a sample file.
It has a few lines.
```

TASK 4:

PROMPT:

generate a Python snippet with a loop that causes a ZeroDivisionError (e.g., dividing numbers in a list). Then, improve it by adding try-except so the loop continues execution safely without crashing

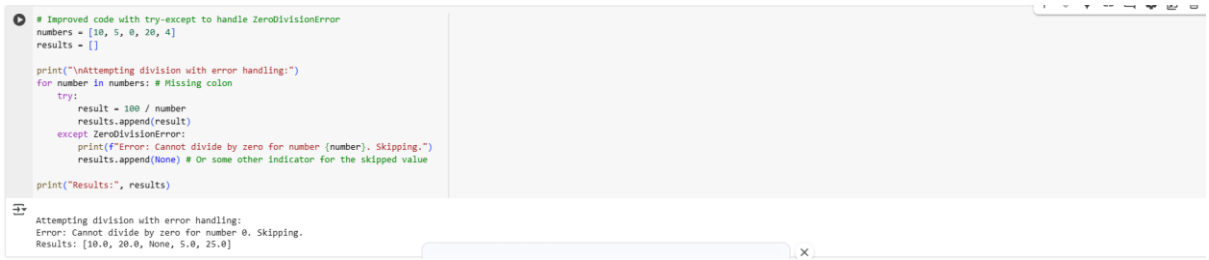


```
# Example of a loop that will cause a ZeroDivisionError
numbers = [10, 5, 0, 20, 4]
results = []

print("Attempting division without error handling:")
for number in numbers # Missing colon
for number in numbers: # Added the missing colon
    result = 100 / number
    results.append(result)

print("Results:", results) # This line will not be reached due to the error
```

```
  File "/tmp/ipython-input-341509298.py", line 6
    for number in numbers # Missing colon
                         ^
SyntaxError: expected ':'
```



```
# Improved code with try-except to handle ZeroDivisionError
numbers = [10, 5, 0, 20, 4]
results = []

print("\nAttempting division with error handling:")
for number in numbers: # Missing colon
    try:
        result = 100 / number
        results.append(result)
    except ZeroDivisionError:
        print(f"Error: Cannot divide by zero for number {number}. Skipping.")
        results.append(None) # Or some other indicator for the skipped value

print("Results:", results)
```

```
Attempting division with error handling:
Error: Cannot divide by zero for number 0. Skipping.
Results: [10.0, 20.0, None, 5.0, 25.0]
```

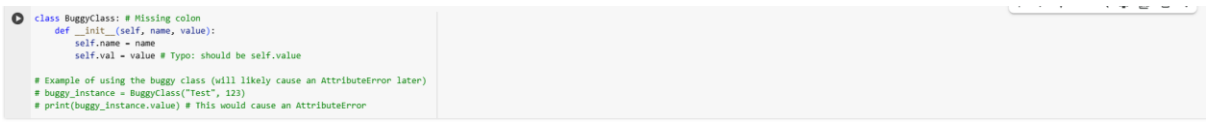
TASK 5:

PROMPT:

generate a Python class definition that contains a bug in its __init__ method, such as mismatched parameters or incorrect attribute references. Then, analyze the issue and rewrite the class with a corrected constructor and proper attribute usage.

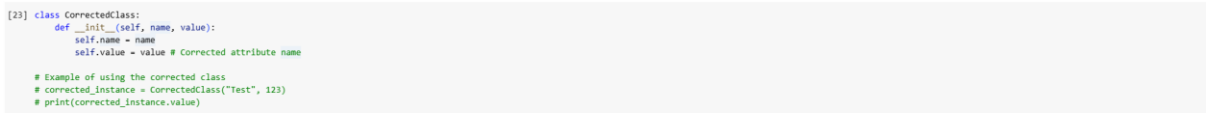
```
[22] -class BuggyClass:
     +class BuggyClass # Missing colon
          def __init__(self, name, value):
              self.name = name
              self.val = value # Typo: should be self.value

      # Example of using the buggy class (will likely cause an AttributeError later)
      # buggy_instance = BuggyClass("Test", 123)
      # print(buggy_instance.value) # This would cause an AttributeError
```



```
class BuggyClass: # Missing colon
    def __init__(self, name, value):
        self.name = name
        self.val = value # Typo: should be self.value

# Example of using the buggy class (will likely cause an AttributeError later)
# buggy_instance = BuggyClass("Test", 123)
# print(buggy_instance.value) # This would cause an AttributeError
```



```
[23] class CorrectedClass:
         def __init__(self, name, value):
             self.name = name
             self.value = value # Corrected attribute name

     # Example of using the corrected class
     # corrected_instance = CorrectedClass("Test", 123)
     # print(corrected_instance.value)
```