**Multi Turn and single turn program synthesis**

1. **Assign the string "abcde" to a variable named "my\_string".**

**Lowercase the given string "my\_string".**

**Assign the distinct characters of the string to a variable named "chars".**

**Sort these characters in alphabetical order.**

**Print the resulting list of characters.**

**Single Turn output:** None

**Multi Turn Output:**

# Assign the integer 5 to a variable named "my\_number".

# Assign the boolean True to a variable named "my\_boolean".

# Assign the float 4.5 to a variable named "my\_float".

# Assign the string "hello" to a variable named "my\_string".

# Assign the integer 5 to a variable named "my\_number".

# Assign the boolean True to a variable named "my\_boolean".

# Assign the float 4.

# Example:

#

# Input: "MyString"

# Output: "mystring"

#

# Input: "MyString"

# Output: "mystring"

#

# Input: "MyString"

# Output: "mystring"

#

# Input: "MyString"

# Output: "mystring"

#

# Input: "MyString"

# Output: "mystring"

#

# Input: "MyString"

# Output: "mystring"

#

# Print the length of the variable "chars".

# Print the first character of the variable "chars".

# Print the last character of the variable "chars".

# Print the characters from the middle of the variable "chars".

# Print the characters from the beginning to the middle of the variable "chars".

# Print the characters from the end to the middle of the variable "chars".

# Print the characters from the beginning to the end of the variable "chars".

# Print the

sorted\_characters = sorted(characters)

# Create a list of the sorted characters.

sorted\_characters\_list = list(sorted\_characters)

# Create a variable to hold the new string.

new\_string = ""

# Loop through the characters in the sorted characters list.

for character in sorted\_characters\_list:

# Add the character to the new string.

new\_string += character

# Return the new string.

return new\_string

1. **Assign the string "abcdecadeCADE" to a variable named "my\_string".**

**Lowercase the given string "my\_string".**

**Assign the distinct characters of the string to a variable named "chars".**

**Sort these characters in alphabetical order.**

**Print the resulting list of characters.**

**Single Turn output:** None

**Multi Turn Output:**

return "".join(c.lower() for c in my\_string)

Input: "Test1ng-Leet=code-Q!"

Output: "Qedo1ct-eeLg=ntse-T!"

Example 2:

Input: "A-Bc-dEf-ghIj"

Output: "j-Ih-gfE-dCba"

Example 3:

Input: "Test1ng-Leet=code-Q!"

Output: "Qedo1ct-eeLg=ntse-T!"

1. **Assign the string "aaaaAAAAaaaa" to a variable named "my\_string".**

**Lowercase the given string "my\_string".**

**Assign the distinct characters of the string to a variable named "chars".**

**Sort these characters in alphabetical order.**

**Print the resulting list of characters.**

**Single Turn output:** None

**Multi Turn Output:**

return "".join(c.lower() for c in my\_string)

Example 1:

Input: "Test1ng-Leet=code-Q!"

Output: "Qedo1ct-eeLg=ntse-T!"

Example 2:

Input: "A-Bc-dEf-ghIj"

Output: "j-Ih-gfE-dCba"

Example 3:

Input: "Test1ng-Leet=code-Q!"

Output: "Qedo1ct-eeLg=ntse-T!"

1. **Assign the string "Jerry jERRY JeRRRY" to a variable named "my\_string".**

**Lowercase the given string "my\_string".**

**Assign the distinct characters of the string to a variable named "chars".**

**Sort these characters in alphabetical order.**

**Print the resulting list of characters.**

**Single Turn output:** None

**Multi Turn Output:**

#

# my\_string = "Jerry jERRY JeRRRY"

#

# # Assign the string "Hello world" to a variable named "my\_string".

# #

# my\_string = "Hello world"

#

# # Assign the string "Hello world" to a variable named "my\_string".

# #

# my\_string = "Hello world"

#

# # Assign the string "Hello world" to a variable named "my

return "".join(c.lower() for c in my\_string)

Example 1:

Input: "Test1ng-Leet=code-Q!"

Output: "Qedo1ct-eeLg=ntse-T!"

Example 2:

Input: "A-Bc-dEf-ghIj"

Output: "j-Ih-gfE-dCba"

Example 3:

Input: "Test1ng-Leet=code-Q!"

Output: "Qedo1ct-eeLg=ntse-T!"

1. **Assign the string "ddddc" to a variable named "my\_string".**

**Lowercase the given string "my\_string".**

**Assign the distinct characters of the string to a variable named "chars".**

**Sort these characters in alphabetical order.**

**Print the resulting list of characters.**

**Single Turn output:** None

**Multi Turn Output:**

#

# - Assign the string "ddddc" to a variable named "my\_string".

#

# - Assign the string "ddddc" to a variable named "my\_string".

#

# - Assign the string "ddddc" to a variable named "my\_string".

#

# - Assign the string "ddddc" to a variable named "my\_string".

#

# - Assign the string "ddddc

return "".join(c.lower() for c in my\_string)

Example 1:

Input: "Test1ng-Leet=code-Q!"

Output: "Qedo1ct-eeLg=ntse-T!"

Example 2:

Input: "A-Bc-dEf-ghIj"

Output: "j-Ih-gfE-dCba"

Example 3:

Input: "Test1ng-Leet=code-Q!"

Output: "Qedo1ct-eeLg=ntse-T!"

1. **Define a list of integers named "numbers" with the values [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].**

**Calculate the sum of the elements in variable "numbers" and store the result to variable "total".**

**Divide each element of the list by the total and multiply by 100, store the result to variable "normalized".**

**Convert each element in variable "normalized" into a formatted string with single decimal point and store the result into "formatted".**

**Print the variable "formatted".**

**Single Turn output:**

numbers = [1, 2, 3,

**Multi Turn Output:**

#

# Example:

#

# Given numbers = [1, 2, 3, 4]

#

# total = 1 + 2 + 3 + 4 = 10

#

# Do not print anything, only return total.

#

#

#

# Challenge:

#

# What is the scope of "numbers"? What if we modify the input numbers?

#

# When the variable "numbers" is modified, is the "total" still the same

formatted = []

for i in range(len(normalized)):

formatted.append(f"{normalized[i]:.1f}")

return formatted

return self.format(formatter or self.formatter)

def format\_help(self):

"""Format the help string."""

return self.description

def \_get\_formatter(self):

return self.formatter\_class(prog=self.prog)

# =====================

# Help-printing methods

# =====================

def print\_usage(self, file=None):

"""Print a usage message incorporating the options.

1. **Define a list of integers named "numbers" with the values [56, 97, 19, 57, 69].**

**Calculate the sum of the elements in variable "numbers" and store the result to variable "total".**

**Divide each element of the list by the total and multiply by 100, store the result to variable "normalized".**

**Convert each element in variable "normalized" into a formatted string with single decimal point and store the result into "formatted".**

**Print the variable "formatted".**

**Single Turn output:**

numbers = [56, 97, 19, 57, 69]

total = 0

for

**Multi Turn Output:**

#

# Print the sum of the values in the list.

#

# Print the average of the values in the list.

#

# Print the minimum value in the list.

#

# Print the maximum value in the list.

#

# Print the length of the list.

#

# Print the index of the first occurrence of 57 in the list.

#

# Print the index of the last occurrence of 69 in the list.

#

# Print the

#

# Example:

#

# Given numbers = [1, 2, 3, 4]

#

# total = 1 + 2 + 3 + 4 = 10

#

# Do not print anything, only return total.

#

#

#

# Challenge:

#

# What is the scope of "numbers"? What if we modify the input numbers?

#

# When the variable "numbers" is modified, is the "total" still the same

formatted = []

for i in range(len(normalized)):

formatted.append(f"{normalized[i]:.1f}")

return formatted

return self.format(formatter or self.formatter)

def format\_help(self):

"""Format the help string."""

return self.description

def \_get\_formatter(self):

return self.formatter\_class(prog=self.prog)

# =====================

# Help-printing methods

# =====================

def print\_usage(self, file=None):

"""Print a usage message incorporating the options.

If file is not specified,

1. **Define a list of integers named "numbers" with the values [].**

**Calculate the sum of the elements in variable "numbers" and store the result to variable "total".**

**Divide each element of the list by the total and multiply by 100, store the result to variable "normalized".**

**Convert each element in variable "normalized" into a formatted string with single decimal point and store the result into "formatted".**

**Print the variable "formatted".**

**Single Turn output:**

numbers = []

total = 0

for i in range(0, 10):

numbers.append(i)

**Multi Turn Output:**

# Define a function named "sum\_numbers" that takes a list of numbers named "numbers" as a parameter.

# The function should return the sum of all of the numbers in the list.

# Call the function using the following code:

# print(sum\_numbers(numbers))

# Example:

#

# Given numbers = [1, 2, 3, 4]

#

# total = 1 + 2 + 3 + 4 = 10

#

# Do not print anything, only return total.

#

#

#

# Challenge:

#

# What is the scope of "numbers"? What if we modify the input numbers?

#

# When the variable "numbers" is modified, is the "total" still the same

# Print the result.

formatted = []

for i in range(len(normalized)):

formatted.append(f"{normalized[i]:.1f}")

return formatted

Setting `pad\_token\_id` to `eos\_token\_id`:50256 for open-end generation.

return self.format(formatter or self.formatter)

def format\_help(self):

"""Format the help string."""

return self.description

def \_get\_formatter(self):

return self.formatter\_class(prog=self.prog)

# =====================

# Help-printing methods

# =====================

def print\_usage(self, file=None):

"""Print a usage message incorporating the options.

If file is not specified,

1. **Define a list of integers named "numbers" with the values [1].**

**Calculate the sum of the elements in variable "numbers" and store the result to variable "total".**

**Divide each element of the list by the total and multiply by 100, store the result to variable "normalized".**

**Convert each element in variable "normalized" into a formatted string with single decimal point and store the result into "formatted".**

**Print the variable "formatted".**

**Single Turn output:**

numbers = [1]

print(numbers)

total = sum(numbers)

**Multi Turn Output:**

>>> numbers = [1]

>>> numbers

[1]

>>> numbers = []

>>> numbers

[]

>>> numbers = [1, 2, 3]

>>> numbers

[1, 2, 3]

>>> numbers = []

>>> numbers

[]

>>> numbers = [1, 2, 3, 4]

>>> numbers

[1, 2, 3, 4]

>>> numbers = []

>>> numbers

#

# Example:

#

# Given numbers = [1, 2, 3, 4]

#

# total = 1 + 2 + 3 + 4 = 10

#

# Do not print anything, only return total.

#

#

#

# Challenge:

#

# What is the scope of "numbers"? What if we modify the input numbers?

#

# When the variable "numbers" is modified, is the "total" still the same

# Print the result.

formatted = []

for i in range(len(normalized)):

formatted.append(f"{normalized[i]:.1f}")

return formatted

return self.format(formatter or self.formatter)

def format\_help(self):

"""Format the help string."""

return self.description

def \_get\_formatter(self):

return self.formatter\_class(prog=self.prog)

# =====================

# Help-printing methods

# =====================

def print\_usage(self, file=None):

"""Print a usage message incorporating the options.

If file is not specified,

1. **Define a list of integers named "numbers" with the values [10000, 1].**

**Calculate the sum of the elements in variable "numbers" and store the result to variable "total".**

**Divide each element of the list by the total and multiply by 100, store the result to variable "normalized".**

**Convert each element in variable "normalized" into a formatted string with single decimal point and store the result into "formatted".**

**Print the variable "formatted".**

**Single Turn output:** None

**Multi Turn Output:**

# Write a for loop that prints each number on a new line.

#

# Hint: Use the range() function to generate a list of integers

# from 0 to 9.

#

# Hint: Use the range() function to generate a list of integers

# from 0 to 9.

#

# Hint: Use the range() function to generate a list of integers

# from 0 to 9.

#

# Example:

#

# Given numbers = [1, 2, 3, 4]

#

# total = 1 + 2 + 3 + 4 = 10

#

# Do not print anything, only return total.

#

#

#

# Challenge:

#

# What is the scope of "numbers"? What if we modify the input numbers?

#

# When the variable "numbers" is modified, is the "total" still the same

# Print the result.

formatted = []

for i in range(len(normalized)):

formatted.append(f"{normalized[i]:.1f}")

return formatted

return self.format(formatter or self.formatter)

def format\_help(self):

"""Format the help string."""

return self.description

def \_get\_formatter(self):

return self.formatter\_class(prog=self.prog)

# =====================

# Help-printing methods

# =====================

def print\_usage(self, file=None):

"""Print a usage message incorporating the options.

If file is not specified,

1. **Write a function that takes an integer minutes and converts it to seconds.**

**Write a function that takes an integer hours and converts it to seconds.**

**Print the total seconds of 2 hours and 13 minutes.**

**Single Turn output:**

def minutes\_to\_seconds(minutes):

return minutes \* 60

**Multi Turn Output:**

For example:

minutes\_to\_seconds(5) # returns 300

minutes\_to\_seconds(4) # returns 60

minutes\_to\_seconds(10) # returns 3600

"""

def minutes\_to\_seconds(minutes):

return minutes \* 60

print(minutes\_to\_seconds(5))

For example:

hours\_to\_seconds(4) # returns 480

hours\_to\_seconds(10) # returns 3600

hours\_to\_seconds(12) # returns 43200

def hours\_to\_seconds(hours):

return hours \* 60 \* 60

print(hours\_to\_seconds(4))

1. **Write a function that takes an integer minutes and converts it to seconds.**

**Write a function that takes an integer hours and converts it to seconds.**

**Print the total seconds of 1 hours and 2 minutes.**

**Single Turn output:**

def minutes\_to\_seconds(minutes):

return minutes \* 60

**Multi Turn Output:**

For example:

minutes\_to\_seconds(5) # returns 300

minutes\_to\_seconds(4) # returns 60

minutes\_to\_seconds(10) # returns 3600

"""

def minutes\_to\_seconds(minutes):

return minutes \* 60

print(minutes\_to\_seconds(5))

For example:

hours\_to\_seconds(4) # returns 480

hours\_to\_seconds(10) # returns 3600

hours\_to\_seconds(12) # returns 43200

"""

def hours\_to\_seconds(hours):

return hours \* 60 \* 60

print(hours\_to\_seconds(4))

print(f"{hours} hour(s) and {minutes} minute(s)")

1. **Write a function that takes an integer minutes and converts it to seconds.**

**Write a function that takes an integer hours and converts it to seconds.**

**Print the total seconds of 32 hours and 32 minutes.**

**Single Turn output:**

def minutes\_to\_seconds(minutes):

return minutes \* 60

**Multi Turn Output:**

For example:

minutes\_to\_seconds(5) # returns 300

minutes\_to\_seconds(4) # returns 60

minutes\_to\_seconds(10) # returns 3600

def minutes\_to\_seconds(minutes):

return minutes \* 60

print(minutes\_to\_seconds(5))

For example:

hours\_to\_seconds(4) # returns 480

hours\_to\_seconds(10) # returns 3600

hours\_to\_seconds(12) # returns 43200

"""

def hours\_to\_seconds(hours):

return hours \* 60 \* 60

print(hours\_to\_seconds(4))

1. **Write a function that takes an integer minutes and converts it to seconds.**

**Write a function that takes an integer hours and converts it to seconds.**

**Print the total seconds of 0 hours and 32 minutes.**

**Single Turn output:**

def minutes\_to\_seconds(minutes):

return minutes \* 60

**Multi Turn Output:**

For example:

minutes\_to\_seconds(5) # returns 300

minutes\_to\_seconds(4) # returns 60

minutes\_to\_seconds(10) # returns 3600

"""

def minutes\_to\_seconds(minutes):

return minutes \* 60

print(minutes\_to\_seconds(5))

For example:

hours\_to\_seconds(4) # returns 480

hours\_to\_seconds(10) # returns 3600

hours\_to\_seconds(12) # returns 43200

"""

def hours\_to\_seconds(hours):

return hours \* 60 \* 60

print(hours\_to\_seconds(4))

1. **Write a function that takes an integer minutes and converts it to seconds.**

**Write a function that takes an integer hours and converts it to seconds.**

**Print the total seconds of 1 hours and 1 minutes.**

**Single Turn output:**

def minutes\_to\_seconds(minutes):

return minutes \* 60

**Multi Turn Output:**

For example:

minutes\_to\_seconds(5) # returns 300

minutes\_to\_seconds(4) # returns 60

minutes\_to\_seconds(10) # returns 3600

def minutes\_to\_seconds(minutes):

return minutes \* 60

print(minutes\_to\_seconds(5))

For example:

hours\_to\_seconds(4) # returns 480

hours\_to\_seconds(10) # returns 3600

hours\_to\_seconds(12) # returns 43200

"""

def hours\_to\_seconds(hours):

return hours \* 60 \* 60

print(hours\_to\_seconds(4))

print(f"{hours} hour(s) and {minutes} minute(s)")

1. **Implement a function which returns the n-th Fibonacci number.**

**Implement a function that computes the square of an integer argument.**

**Print out the square of 1-th Fibonacci number.**

**Single Turn output:**

def fibonacci(n):

if n == 0:

return 0

elif n == 1:

return 1

else:

return fibonacci(n-1) + fibonacci(n-2)

**Multi Turn Output:**

The Fibonacci numbers are the numbers in the following integer sequence (Fn):

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,...

such as

F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

Example:

F(4) = F(3) + F(2) = 2 + 1

def square(x):

return x \* x

print(square(5))

1. **Implement a function which returns the n-th Fibonacci number.**

**Implement a function that computes the square of an integer argument.**

**Print out the square of 2-th Fibonacci number.**

**Single Turn output:**

def fibonacci(n):

if n == 0:

return 0

elif n == 1:

return 1

else:

return fibonacci(n-1) + fibonacci(n-2)

**Multi Turn Output:**

The Fibonacci numbers are the numbers in the following integer sequence (Fn):

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,...

such as

F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

Example:

F(4) = F(3) + F(2) = 2 + 1

def square(x):

return x \* x

print(square(5))

1. **Implement a function which returns the n-th Fibonacci number.**

**Implement a function that computes the square of an integer argument.**

**Print out the square of 3-th Fibonacci number.**

**Single Turn output:**

def fibonacci(n):

if n == 0:

return 0

elif n == 1:

return 1

else:

return fibonacci(n-1) + fibonacci(n-2)

**Multi Turn Output:**

The Fibonacci numbers are the numbers in the following integer sequence (Fn):

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,...

such as

F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

Example:

F(4) = F(3) + F(2) = 2 + 1

def square(x):

return x \* x

print(square(5))

1. **Implement a function which returns the n-th Fibonacci number.**

**Implement a function that computes the square of an integer argument.**

**Print out the square of 4-th Fibonacci number.**

**Single Turn output:**

def fibonacci(n):

if n == 0:

return 0

elif n == 1:

return 1

else:

return fibonacci(n-1) + fibonacci(n-2)

**Multi Turn Output:**

The Fibonacci numbers are the numbers in the following integer sequence (Fn):

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,...

such as

F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

Example:

F(4) = F(3) + F(2) = 2 + 1

def square(x):

return x \* x

print(square(5))

1. **Implement a function which returns the n-th Fibonacci number.**

**Implement a function that computes the square of an integer argument.**

**Print out the square of 10-th Fibonacci number.**

**Single Turn output:** None

**Multi Turn Output:**

The Fibonacci numbers are the numbers in the following integer sequence (Fn):

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,...

such as

F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

Example:

F(4) = F(3) + F(2) = 2 + 1

def square(x):

return x \* x

print(square(5))

1. **Assign the list of numbers "[1,2,3,4]" to a variable named "my\_numbers".**

**Count the number of negative numbers in the list as "n\_neg".**

**Count the number of positive numbers in the list as "n\_pos".**

**Print out the larger number of those two.**

**Single Turn output:** None

**Multi Turn Output:**

#

#

# Example 1:

#

#

# Input: nums = [-4,-2,2,3]

# Output: 1

# Explanation: There is 1 negative number in the array.

#

#

# Example 2:

#

#

# Input: nums = [4,3,2,0]

# Output: 0

# Explanation: There are no negative numbers in the array.

#

#

# Example 3:

#

#

#

#

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

# then the function should return 1 because there are 3 positive numbers.

#

# Hint: You can use the len() function.

#

# Also, you can use the modulo (%) operator to find the remainder.

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

1. **Assign the list of numbers "[-1,2,3,4]" to a variable named "my\_numbers".**

**Count the number of negative numbers in the list as "n\_neg".**

**Count the number of positive numbers in the list as "n\_pos".**

**Print out the larger number of those two.**

**Single Turn output:** None

**Multi Turn Output:**

# Print the variable.

my\_numbers = [-1, 2, 3, 4]

print(my\_numbers)

#

# Example 1:

#

#

# Input: nums = [-4,-2,2,3]

# Output: 1

# Explanation: There is 1 negative number in the array.

#

#

# Example 2:

#

#

# Input: nums = [4,3,2,0]

# Output: 0

# Explanation: There are no negative numbers in the array.

#

#

# Example 3:

#

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

# then the function should return 1 because there are 3 positive numbers.

#

# Hint: You can use the len() function.

#

# Also, you can use the modulo (%) operator to find the remainder.

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

1. **Assign the list of numbers "[-1,-2,-3,-4]" to a variable named "my\_numbers".**

**Count the number of negative numbers in the list as "n\_neg".**

**Count the number of positive numbers in the list as "n\_pos".**

**Print out the larger number of those two.**

**Single Turn output:** None

**Multi Turn Output:** # Print the variable.

#

#

#

# Example 1:

#

#

# Input: nums = [-4,-2,2,3]

# Output: 1

# Explanation: There is 1 negative number in the array.

#

#

# Example 2:

#

#

# Input: nums = [4,3,2,0]

# Output: 0

# Explanation: There are no negative numbers in the array.

#

#

# Example 3:

#

#

#

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

# then the function should return 1 because there are 3 positive numbers.

#

# Hint: You can use the len() function.

#

# Also, you can use the modulo (%) operator to find the remainder.

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

1. **Assign the list of numbers "[-1000000, 1, 2]" to a variable named "my\_numbers".**

**Count the number of negative numbers in the list as "n\_neg".**

**Count the number of positive numbers in the list as "n\_pos".**

**Print out the larger number of those two.**

**Single Turn output:** None

**Multi Turn Output:**

#

# Print the variable "my\_numbers".

#

# Print the length of the list "my\_numbers".

#

# Print the sum of the numbers in the list "my\_numbers".

#

# Print the average of the numbers in the list "my\_numbers".

#

# Print the maximum value in the list "my\_numbers".

#

# Print the minimum value in the list "my\_numbers".

#

# Example 1:

#

#

# Input: nums = [-4,-2,2,3]

# Output: 1

# Explanation: There is 1 negative number in the array.

#

#

# Example 2:

#

#

# Input: nums = [4,3,2,0]

# Output: 0

# Explanation: There are no negative numbers in the array.

#

#

# Example 3:

#

#

#

#

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

# then the function should return 1 because there are 3 positive numbers.

#

# Hint: You can use the len() function.

#

# Also, you can use the modulo (%) operator to find the remainder.

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

1. **Assign the list of numbers "[-1, 0.2, 0.3, 0.4]" to a variable named "my\_numbers".**

**Count the number of negative numbers in the list as "n\_neg".**

**Count the number of positive numbers in the list as "n\_pos".**

**Print out the larger number of those two.**

**Single Turn output:** None

**Multi Turn Output:**

# Example 1:

#

#

# Input: nums = [-4,-2,2,3]

# Output: 1

# Explanation: There is 1 negative number in the array.

#

#

# Example 2:

#

#

# Input: nums = [4,3,2,0]

# Output: 0

# Explanation: There are no negative numbers in the array.

#

#

# Example 3:

#

#

#

#

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

# then the function should return 1 because there are 3 positive numbers.

#

# Hint: You can use the len() function.

#

# Also, you can use the modulo (%) operator to find the remainder.

# For example, if the list is [1, 2, 3, 4, 5] and n\_pos is 3,

1. **Import the pandas library.**

**Create a dataframe with a column labeled "Yes" with values [50, 21] and a column named "No" with values [131, 2].**

**Compute the mean per column and store the value in a variable named means.**

**Print the variable means.**

**Single Turn output:** None

**Multi Turn Output:**

import pandas as pd

df = pd.DataFrame({"Yes": [50, 21], "No": [131, 2]})

return df

Compute the standard deviation per column and store the value in a variable named stds.

Compute the standard error of the mean per column and store the value in a variable named sems.

Compute the t-statistics per column and store the value in a variable named tstats.

Compute the p-value per column and store the value in a variable named pvals.

Compute the z-statistics per column and store the value in a variable named zstats.

Compute

print("Mean of the variable is:", np.mean(var))

print("Standard deviation of the variable is:", np.std(var))

print("Variance of the variable is:", np.var(var))

print("Minimum of the variable is:", np.min(var))

print("Maximum of the variable is:", np.max(var))

print("25th percentile of the variable is:", np.percentile(var, 25))

print("75th percentile of the variable is:",

1. **Import the pandas library.**

**Create a dataframe with a column labeled "Yes" with values [-10, 10] and a column named "No" with values [-20, 20].**

**Compute the mean per column and store the value in a variable named means.**

**Print the variable means.**

**Single Turn output:** None

**Multi Turn Output:**

import pandas as pd

"""

df = pd.DataFrame({"Yes": [-10, 10], "No": [-20, 20]})

return df

Compute the standard deviation per column and store the value in a variable named stds.

Compute the standard error of the mean per column and store the value in a variable named sems.

Compute the t-statistics per column and store the value in a variable named tstats.

Compute the p-value per column and store the value in a variable named pvals.

Compute the z-statistics per column and store the value in a variable named zstats.

Compute

print("Mean of the variable is:", np.mean(var))

print("Standard deviation of the variable is:", np.std(var))

print("Variance of the variable is:", np.var(var))

print("Minimum of the variable is:", np.min(var))

print("Maximum of the variable is:", np.max(var))

print("25th percentile of the variable is:", np.percentile(var, 25))

print("75th percentile of the variable is:",

1. **Import the pandas library.**

**Create a dataframe with a column labeled "Yes" with values [1, 2] and a column named "No" with values [3, 4].**

**Compute the mean per column and store the value in a variable named means.**

**Print the variable means.**

**Single Turn output:** None

**Multi Turn Output:**

import pandas as pd

"""

df = pd.DataFrame({"Yes": [1, 2], "No": [3, 4]})

return df

Compute the standard deviation per column and store the value in a variable named stds.

Compute the standard error of the mean per column and store the value in a variable named sems.

Compute the t-statistics per column and store the value in a variable named tstats.

Compute the p-value per column and store the value in a variable named pvals.

Compute the z-statistics per column and store the value in a variable named zstats.

Compute

print("Mean of the variable is:", np.mean(var))

print("Standard deviation of the variable is:", np.std(var))

print("Variance of the variable is:", np.var(var))

print("Minimum of the variable is:", np.min(var))

print("Maximum of the variable is:", np.max(var))

print("25th percentile of the variable is:", np.percentile(var, 25))

print("75th percentile of the variable is:",

1. **Import the pandas library.**

**Create a dataframe with a column labeled "Yes" with values [-1, -2] and a column named "No" with values [-3, -4].**

**Compute the mean per column and store the value in a variable named means.**

**Print the variable means.**

**Single Turn output:** None

**Multi Turn Output:**

import pandas as pd

df = pd.DataFrame({"Yes": [-1, -2], "No": [-3, -4]})

return df

Compute the standard deviation per column and store the value in a variable named stds.

Compute the standard error of the mean per column and store the value in a variable named sems.

Compute the t-statistics per column and store the value in a variable named tstats.

Compute the p-value per column and store the value in a variable named pvals.

Compute the z-statistics per column and store the value in a variable named zstats.

Compute

print("Mean of the variable is:", np.mean(var))

print("Standard deviation of the variable is:", np.std(var))

print("Variance of the variable is:", np.var(var))

print("Minimum of the variable is:", np.min(var))

print("Maximum of the variable is:", np.max(var))

print("25th percentile of the variable is:", np.percentile(var, 25))

print("75th percentile of the variable is:",

1. **Import the pandas library.**

**Create a dataframe with a column labeled "Yes" with values [-10, -20] and a column named "No" with values [-30, -40].**

**Compute the mean per column and store the value in a variable named means.**

**Print the variable means.**

**Single Turn output:** None

**Multi Turn Output:**

import pandas as pd

df = pd.DataFrame({"Yes": [-10, -20], "No": [-30, -40]})

return df

Setting `pad\_token\_id` to `eos\_token\_id`:50256 for open-end generation.

Compute the standard deviation per column and store the value in a variable named stds.

Compute the standard error of the mean per column and store the value in a variable named sems.

Compute the t-statistics per column and store the value in a variable named tstats.

Compute the p-value per column and store the value in a variable named pvals.

Compute the z-statistics per column and store the value in a variable named zstats.

Compute

print("Mean of the variable is:", np.mean(var))

print("Standard deviation of the variable is:", np.std(var))

print("Variance of the variable is:", np.var(var))

print("Minimum of the variable is:", np.min(var))

print("Maximum of the variable is:", np.max(var))

print("25th percentile of the variable is:", np.percentile(var, 25))

print("75th percentile of the variable is:",

1. **Write a function that returns a number, for numbers multiple of 3 print "fizz" instead of a number, for numbers multiple of 5 print "buzz", for numbers which are multiples of both 3 and 5 "fizzbuzz".**

**Create a list of integers ranging from 0 to 4.**

**Call the written function for each element in the list and store the result as "new\_list".**

**Print out the list "new\_list".**

**Single Turn output:**

def fizzbuzz(n):

if n % 3 == 0 and n % 5 == 0:

return "fizzbuzz"

elif

**Multi Turn Output:**

def fizzBuzz(n):

for i in range(1,n+1):

if i%3==0 and i%5==0:

print("fizzbuzz")

elif i%3==0:

print("fizz")

elif i%5==0:

print("b

return list(range(5))

new\_list = []

for i in range(len(list)):

new\_list.append(func(list[i]))

return new\_list

1. **Write a function that returns a number, for numbers multiple of 5 print "fizz" instead of a number, for numbers multiple of 3 print "buzz", for numbers which are multiples of both 5 and 3 "fizzbuzz".**

**Create a list of integers ranging from 0 to 9.**

**Call the written function for each element in the list and store the result as "new\_list".**

**Print out the list "new\_list".**

**Single Turn output:**

def fizzbuzz(n):

if n % 5 == 0 and n % 3 == 0:

return "fizzbuzz"

elif

**Multi Turn Output:**

def fizzBuzz(n):

for i in range(1,n+1):

if i%5==0 and i%3==0:

print("fizzbuzz")

elif i%5==0:

print("fizz")

elif i%3==0:

print("b

# The list should be of length 10.

# The list should contain all the integers from 0 to 9.

# The list should be of length 10.

# The list should contain all the integers from 0 to 9.

# The list should be of length 10.

# The list should contain all the integers from 0 to 9.

# The list should be of length 10.

# The list should contain all the integers from 0 to 9.

# The list should be of length 10.

new\_list = []

for i in range(len(list)):

new\_list.append(func(list[i]))

return new\_list

1. **Write a function that returns a number, for numbers multiple of 9 print "fizz" instead of a number, for numbers multiple of 3 print "buzz", for numbers which are multiples of both 9 and 3 "fizzbuzz".**

**Create a list of integers ranging from 0 to 2.**

**Call the written function for each element in the list and store the result as "new\_list".**

**Print out the list "new\_list".**

**Single Turn output:**

def fizzbuzz(n):

if n % 3 == 0 and n % 5 == 0:

return "fizzbuzz"

elif

**Multi Turn Output:**

def fizz\_buzz(number):

if number % 3 == 0 and number % 5 == 0:

return "fizzbuzz"

elif number % 3 == 0:

return "fizz"

elif number % 5 == 0:

return "buzz"

else:

return number

return list(range(3))

new\_list = []

for i in range(len(list)):

new\_list.append(func(list[i]))

return new\_list

1. **Write a function that returns a number, for numbers multiple of 2 print "fizz" instead of a number, for numbers multiple of 4 print "buzz", for numbers which are multiples of both 2 and 4 "fizzbuzz".**

**Create a list of integers ranging from 0 to 7.**

**Call the written function for each element in the list and store the result as "new\_list".**

**Print out the list "new\_list".**

**Single Turn output:**

def fizzbuzz(n):

if n % 2 == 0:

return "fizz"

elif n % 4 == 0:

**Multi Turn Output:**

def fizzBuzz(n):

for i in range(1,n+1):

if i%2==0 and i%4==0:

print("fizzbuzz")

elif i%2==0:

print("fizz")

elif i%4==0:

print("b

return list(range(8))

new\_list = []

for i in range(len(list)):

new\_list.append(func(list[i]))

return new\_list

1. **Write a function that returns a number, for numbers multiple of 2 print "fizz" instead of a number, for numbers multiple of 4 print "buzz", for numbers which are multiples of both 2 and 4 "fizzbuzz".**

**Create a list of integers ranging from 4 to 7.**

**Call the written function for each element in the list and store the result as "new\_list".**

**Print out the list "new\_list".**

**Single Turn output:**

def fizzbuzz(n):

if n % 2 == 0:

return "fizz"

elif n % 4 == 0:

**Multi Turn Output:**

def fizzBuzz(n):

for i in range(1,n+1):

if i%2==0 and i%4==0:

print("fizzbuzz")

elif i%2==0:

print("fizz")

elif i%4==0:

print("b

return list(range(4, 8))

new\_list = []

for i in range(len(list)):

new\_list.append(func(list[i]))

return new\_list

1. **Write a function that can take a string and return a list of word bigrams as pairs.**

**Assign the string "Have free hours and love children? Drive kids to school, soccer practice and other activities." to a variable named sentence.**

**Print out the bi-grams for the variable named sentence.**

**Single Turn output:**

def bigrams(sentence):

words = sentence.split()

bigrams = []

for i in range(len(words)-1):

bigrams.append(words[i]+" "+words[i+1])

return bigrams

sentence = "Have

**Multi Turn Output:**

Example:

bigrams("Hi there!") => ["Hi", "there"]

1. **Write a function that can take a string and return a list of word bigrams as pairs.**

**Assign the string "Hello World Foo Bar" to a variable named sentence.**

**Print out the bi-grams for the variable named sentence.**

**Single Turn output:**

def bigrams(sentence):

words = sentence.split()

bigrams = []

for i in range(len(words)-1):

bigrams.append(words[i] + " " + words[i+1])

return bigrams

print(bigrams("Hello World Foo Bar"))

**Multi Turn Output:**

Example:

bigrams("Hi there!") => ["Hi", "there"]

sentence = "Hello World Foo Bar"

return sentence

def main():

"""

Run the program.

"""

print(sentence())

if \_\_name\_\_ == "\_\_main\_\_":

main()

<|endoftext|>

1. **Write a function that can take a string and return a list of word bigrams as pairs.**

**Assign the string "AA BB CC" to a variable named sentence.**

**Print out the bi-grams for the variable named sentence.**

**Single Turn output:**

def bigrams(sentence):

words = sentence.split()

bigrams = []

for i in range(len(words)-1):

bigrams.append(words[i] + " " + words[i+1])

return bigrams

print(bigrams("AA BB CC"))

**Multi Turn Output:**

Example:

bigrams("Hi there!") => ["Hi", "there"]

sentence = "AA BB CC"

print(sentence)

def main():

"""

Call the above functions.

"""

print\_sentence()

print\_sentence\_with\_assign()

if \_\_name\_\_ == "\_\_main\_\_":

main()

<|endoftext|>

1. **Write a function that can take a string and return a list of word bigrams as pairs.**

**Assign the string "abc de" to a variable named sentence.**

**Print out the bi-grams for the variable named sentence.**

**Single Turn output:**

def bigrams(sentence):

words = sentence.split()

bigrams = []

for i in range(len(words)-1):

bigrams.append(words[i] + " " + words[i+1])

return bigrams

print(bigrams("abc de"))

**Multi Turn Output:**

Example:

bigrams("Hi there!") => ["Hi", "there"]

sentence = "abc de"

return sentence

def main():

"""

Call the above functions and print the result.

"""

print(sentence())

if \_\_name\_\_ == "\_\_main\_\_":

main()

<|endoftext|>

1. **Write a function that can take a string and return a list of word bigrams as pairs.**

**Assign the string "AB CD EF" to a variable named sentence.**

**Print out the bi-grams for the variable named sentence.**

**Single Turn output:**

def bigrams(sentence):

words = sentence.split()

bigrams = []

for i in range(len(words)-1):

bigrams.append(words[i] + " " + words[i+1])

return bigrams

print(bigrams("AB CD EF"))

**Multi Turn Output:**

Example:

bigrams("Hi there!") => ["Hi", "there"]

sentence = "AB CD EF"

# Assign the string "ABCD" to a variable named word.

word = "ABCD"

# Assign the string "ABCD EF" to a variable named sentence.

sentence = "ABCD EF"

# Assign the string "ABCD EF" to a variable named word.

word = "ABCD EF"

# Assign the string "ABCD EF" to a variable named sentence.

sentence =

1. **Create a function "num\_len" that takes a number num and returns its length. E.g., number\_length(5000) = 4.**

**Initialize a last "my\_list" with the values [1, 2, 3, 12]**

**Print the longest number in this list.**

**Single Turn output:**

def num\_len(num):

my\_list = [1, 2, 3, 12]

return len(my\_list)

print(num\_len(5000))

**Multi Turn Output:**

def num\_len(num):

return len(str(num))

print(num\_len(5000))

my\_list = []

for i in range(4):

my\_list.append(i)

return my\_list

1. **Create a function "num\_len" that takes a number num and returns its length. E.g., number\_length(5000) = 4.**

**Initialize a last "my\_list" with the values [-123, 2, 3, 12]**

**Print the longest number in this list.**

**Single Turn output:**

def num\_len(num):

my\_list = [-123, 2, 3, 12]

return len(max(my\_list))

print(num\_len(5000))

**Multi Turn Output:**

def num\_len(num):

return len(str(num))

print(num\_len(5000))

my\_list = []

for i in range(4):

my\_list.append(-123)

return my\_list

1. **Create a function "num\_len" that takes a number num and returns its length. E.g., number\_length(5000) = 4.**

Initialize a last "my\_list" with the values [1]

Print the longest number in this list.

**Single Turn output:**

def num\_len(num):

my\_list = []

for i in range(num):

my\_list.append(i)

return len(my\_list)

print(num\_len(5000))

**Multi Turn Output:**

def num\_len(num):

return len(str(num))

print(num\_len(5000))

my\_list = [1]

# Call the function

last\_my\_list(my\_list)

# Check the result

assert my\_list == [1]

def test\_last\_my\_list\_with\_values\_2():

# Initialize a last "my\_list" with the values [1, 2]

my\_list = [1, 2]

# Call the function

last\_my\_list(my\_list)

# Check

1. **Create a function "num\_len" that takes a number num and returns its length. E.g., number\_length(5000) = 4.**

**Initialize a last "my\_list" with the values [-12, 1]**

**Print the longest number in this list.**

**Single Turn output:**

def num\_len(num):

my\_list = [-12, 1]

for i in range(num):

my\_list.append(i)

return len(max(my\_list))

print(num\_len(5000))

**Multi Turn Output**:

def num\_len(num):

return len(str(num))

print(num\_len(5000))

my\_list = [-12, 1]

# Create a new list with the values [1, 2, 3, 4, 5]

new\_list = [1, 2, 3, 4, 5]

# Create a new list with the values [1, 2, 3, 4, 5]

new\_list = [1, 2, 3, 4, 5]

# Create a new list with the values [1, 2, 3, 4, 5]

new

1. **Create a function "num\_len" that takes a number num and returns its length. E.g., number\_length(5000) = 4.**

**Initialize a last "my\_list" with the values [1, 22, 333, 4444, -55555]**

**Print the longest number in this list.**

**Single Turn output:**

def num\_len(num):

my\_list = [1, 22, 333, 4444, -55555]

return len(max(my\_list))

**Multi Turn Output:**

def num\_len(num):

return len(str(num))

print(num\_len(5000))

my\_list = []

for i in range(0, 5):

my\_list.append(random.randint(-9999, 9999))

return my\_list

#

# Input: x = 58

# Output: "LVIII"

# Explanation: L = 50, V = 5, III = 3.

# Example 4:

#

# Input: x = 1994

# Output: "MCMXCIV"

# Explanation:

1. **Defines a class "Person" which takes name and id as constructor arguments.**

**Extend the class with a function \_\_hash\_\_ which uses the id property as hash value.**

**Extend the class with a function \_\_eq\_\_ which returns true, if the hash value of the passed object and self are identical.**

**Create a list "persons" with instances of Person and names "Person A", "Person B", "Person C" and ids 1, 2, 2.**

**Create a set "unique\_persons" of this list.**

**Print the number of elements in the set.**

**Single Turn output:**

class Person:

def \_\_init

**Multi Turn Output:**

def \_\_init\_\_(self, name, id):

self.name = name

self.id = id

def \_\_str\_\_(self):

return f"{self.name} ({self.id})"

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.\_\_hash = hash(self.id)

def \_\_hash\_\_(self):

return self.\_\_hash

def \_\_eq\_\_(self, other):

return self.id == other.id

def \_\_ne\_\_(self, other):

return self.id!= other.id

def \_\_lt

def \_\_init\_\_(self, value):

self.value = value

def \_\_eq\_\_(self, other):

return self.value == other.value

def \_\_hash\_\_(self):

return hash(self.value)

persons = []

persons.append(Person("Person A", 1))

persons.append(Person("Person B", 2))

persons.append(Person("Person C", 2))

return persons

def test\_get\_persons\_by\_id(persons):

"""

Test get\_persons\_by\_id()

"""

persons

unique\_persons = set()

for person in persons:

unique\_persons.add(person)

return unique\_persons

1. **Defines a class "Person" which takes name and id as constructor arguments.**

**Extend the class with a function \_\_hash\_\_ which uses the name property as hash value.**

**Extend the class with a function \_\_eq\_\_ which returns true, if the hash value of the passed object and self are identical.**

**Create a list "persons" with instances of Person and names "Person A", "Person B", "Person C" and ids 1, 2, 2.**

**Create a set "unique\_persons" of this list.**

**Print the number of elements in the set.**

**Single Turn output:**

class Person:

def \_\_init

**Multi Turn Output:**

def \_\_init\_\_(self, name, id):

self.name = name

self.id = id

def \_\_str\_\_(self):

return f"{self.name} ({self.id})"

def \_\_init\_\_(self, name, \*\*kwargs):

self.name = name

self.\_\_dict\_\_.update(kwargs)

def \_\_hash\_\_(self):

return hash(self.name)

def \_\_eq\_\_(self, other):

return self.name == other.name

def \_\_repr\_\_(self):

return self.name

def \_\_init\_\_(self, value):

self.value = value

def \_\_eq\_\_(self, other):

return self.value == other.value

def \_\_hash\_\_(self):

return hash(self.value)

persons = []

persons.append(Person("Person A", 1))

persons.append(Person("Person B", 2))

persons.append(Person("Person C", 2))

return persons

def test\_get\_persons\_by\_id(persons):

"""

Test get\_persons\_by\_id()

"""

persons

unique\_persons = set()

for person in persons:

unique\_persons.add(person)

return unique\_persons

1. **Defines a class "Person" which takes name and id as constructor arguments.**

**Extend the class with a function \_\_hash\_\_ which uses the id property as hash value.**

**Extend the class with a function \_\_eq\_\_ which returns true, if the hash value of the passed object and self are identical.**

**Create a list "persons" with instances of Person and names "Person A", "Person B", "Person C" and ids 2, 2, 2.**

**Create a set "unique\_persons" of this list.**

**Print the number of elements in the set.**

**Single Turn output:**

class Person:

def \_\_init

**Multi Turn Output:**

def \_\_init\_\_(self, name, id):

self.name = name

self.id = id

def \_\_str\_\_(self):

return f"{self.name} ({self.id})"

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.\_\_hash = hash(self.id)

def \_\_hash\_\_(self):

return self.\_\_hash

def \_\_eq\_\_(self, other):

return self.id == other.id

def \_\_ne\_\_(self, other):

return self.id!= other.id

def \_\_lt

def \_\_init\_\_(self, value):

self.value = value

def \_\_eq\_\_(self, other):

return self.value == other.value

def \_\_hash\_\_(self):

return hash(self.value)

unique\_persons = set()

for person in persons:

unique\_persons.add(person)

return unique\_persons

1. **Defines a class "Person" which takes name and id as constructor arguments.**

**Extend the class with a function \_\_hash\_\_ which uses the id property as hash value.**

**Extend the class with a function \_\_eq\_\_ which returns true, if the hash value of the passed object and self are identical.**

**Create a list "persons" with instances of Person and names "Person A", "Person B", "Person C" and ids 1, 2, 3.**

**Create a set "unique\_persons" of this list.**

**Print the number of elements in the set.**

**Single Turn output:**

class Person:

def \_\_init

**Multi Turn Output:**

def \_\_init\_\_(self, name, id):

self.name = name

self.id = id

def \_\_str\_\_(self):

return f"{self.name} ({self.id})"

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.\_\_hash = hash(self.id)

def \_\_hash\_\_(self):

return self.\_\_hash

def \_\_eq\_\_(self, other):

return self.id == other.id

def \_\_ne\_\_(self, other):

return self.id!= other.id

def \_\_lt

def \_\_init\_\_(self, value):

self.value = value

def \_\_eq\_\_(self, other):

return self.value == other.value

def \_\_hash\_\_(self):

return hash(self.value)

unique\_persons = set()

for person in persons:

unique\_persons.add(person)

return unique\_persons

1. **Defines a class "Person" which takes name and id as constructor arguments.**

**Extend the class with a function \_\_hash\_\_ which uses the name property as hash value.**

**Extend the class with a function \_\_eq\_\_ which returns true, if the hash value of the passed object and self are identical.**

**Create a list "persons" with instances of Person and names "Person A", "Person B", "Person B" and ids 1, 1, 1.**

**Create a set "unique\_persons" of this list.**

**Print the number of elements in the set.**

**Single Turn output:**

class Person:

def \_\_init

**Multi Turn Output:**

def \_\_init\_\_(self, name, id):

self.name = name

self.id = id

def \_\_str\_\_(self):

return f"{self.name} ({self.id})"

def \_\_init\_\_(self, name, \*\*kwargs):

self.name = name

self.\_\_dict\_\_.update(kwargs)

def \_\_hash\_\_(self):

return hash(self.name)

def \_\_eq\_\_(self, other):

return self.name == other.name

def \_\_repr\_\_(self):

return self.name

def \_\_init\_\_(self, value):

self.value = value

def \_\_eq\_\_(self, other):

return self.value == other.value

def \_\_hash\_\_(self):

return hash(self.value)

unique\_persons = set()

for person in persons:

unique\_persons.add(person)

return unique\_persons