

Informed Consent

Information Letter

[Using ILMS to provide context dependent explanations for teaching code consent form](#)

Welcome to the research study!

We are interested in understanding how context can be added to prompts for teaching Python. For this study, you will be presented with how we envision context being used in an programming education environment. Then, you will be asked to answer some questions about it. Your responses will be kept completely confidential.

The study should take you around 60 minutes to complete. Your participation in this research is voluntary. You have the right to withdraw at any point during the study. The Principal Investigator of this study can be contacted at l.verhaegh1@students.uu.nl.

By clicking the button below, you acknowledge:

- Your participation in the study is voluntary.
- You are 18 years of age.
- You are aware that you may choose to terminate your participation at any time for any reason.

- I consent, begin the study
 I do not consent, I do not wish to participate

Subject infoBlock 14

How many years have you been involved in teaching python?

Context explanation

This survey is part of a bachelor thesis, researching how LLMs can be made context aware for teaching programming. While learning the student will not have programming knowledge otherwise deemed trivial, and LLMs may assume the student does have this knowledge. By developing techniques for LLMs to be context aware the student can use LLMs to their full potential. In order to make sure the LLM would have access to this context we envision a plugin to the learning environment, allowing the student to perform several different actions.

Here depicted in a Jupyter Notebook, the LLM can gather the context from where in the notebook it is activated. This survey tests how this context should be integrated in the prompts to the LLM for the best result.

We have identified several different operations the LLM should be able to perform, the 3 that will be tested in this survey are:

- Generate examples
- Explain existing code
- Generate novel exercises

For each of these operations the survey will cover one or two base prompts, which are then modified using several prompting techniques.

All of the base prompts assume the student has knowledge up to and including recursion.

The goal of the study is to find which technique performs best, as such you will be asked to rate the:

- Correctness
- Completeness
- Relevancy to the task
- Relevancy to the context (student with knowledge up to recursion)
- Whether or not the response focuses on recursion

Functions

We have seen and used functions before (`print`, `input`, and `bin`, for example), now it is time to have a more systematic look at them and to define functions ourselves. Functions are useful whenever a piece of code forms a self-contained component and is (potentially) used repeatedly in the program. It simplifies coding, testing, debugging and maintenance if this code is only defined once (as a function) and just called when it is needed.

Concepts up to recursion:

- Programming basics
- Variables
- Input
- Arithmetic Expressions
- Boolean Expressions
- If-Else
- Nested If
- Elif
- For Loop
- While Loop
- Range function
- Nested Loop
- Functions
- Values and References
- String formatting
- String operations
- Python Lists 1,2
- Python Dictionaries
- Recursion

Generate examples: Q1 part 1

The following responses are answering the question of: "How can recursion be used to calculate powers?"

The student in question has knowledge of concepts up to recursion and is currently working on a Jupyter notebook that explains recursion.

Response #1:

" Of course! I'd be happy to help you with that. Here is a function that uses recursion to calculate powers in Python:

```

```
def power(base, power):
 # Base case: If the power is 0, return 1
 if power == 0:
 return 1
```

```
Recursive case: Otherwise, calculate the power of the base
by multiplying it by itself the requested number of times
return power * power * base
```

# Example usage:

```
base = 2
power = 3
result = power(base, power)
print(result)
```

This function takes in two parameters, `base` and `power`, and returns the result of raising `base` to the power `power`. The function first checks if `power` is 0, in which case it returns 1. Otherwise, it uses recursion to calculate the power of `base` by multiplying it by itself `power` times.

For example, if `base` is 2 and `power` is 3, the function will calculate  $2 \times 2 \times 2 = 8$ , and return 8.

I hope this helps! Let me know if you have any questions or need further assistance."

The following questions relate to response #1

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

Response #2:

"Of course! I'm happy to help you with that. Here is a function that uses recursion to calculate powers in Python:

``

```

def power(base, power):
 # Base case: If the power is 0, return 1
 if power == 0:
 return 1

 # Recursive case: Calculate the power of the base raised to the power - 1
 return base ** (power - 1) * power

Example usage:
print(power(2, 3)) # Output: 8
```

```

This function takes two arguments, `base` and `power`, and returns the result of raising `base` to the power `power`. The function uses a base case where `power` is 0, and returns 1 in that case. Otherwise, it uses recursion to calculate the power of `base` raised to `power - 1`, and then multiplies that result by `power`.

I hope this helps! Let me know if you have any questions or need further assistance."

The following questions relate to response #2

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Response #3:

"Sure! Here is a function that uses recursion to calculate powers in Python:

``

```
def power(base, power):
    # Base case: If the power is 0, return 1
    if power == 0:
        return 1

    # Recursive case: Otherwise, calculate the power of the base
    return base ** power
```

```

This function takes two arguments: `base` and `power`. It first checks if the power is 0 using an if statement. If it is, it returns 1. Otherwise, it calls itself with the `base` and `power - 1` as arguments, and multiplies the result by the power. This is the recursive case. Finally, it returns the result.

You can use this function by calling it with the base and power you want to calculate, like this: `power(2, 3)`

I hope this helps! Let me know if you have any questions."

The following questions relate to response #3

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

Response #4:

" Sure, I'd be happy to help you with that! Here is a recursive function in Python that calculates powers using only basic concepts up to recursion:

```

```
def power(base, power):
```

```
# Base case: If the power is 0, return 1
if power == 0:
    return 1

# Recursive case: Calculate the power of the base raised to the power - 1
return base ** (power - 1) * power
```

```

Explanation:

The function `power` takes two input parameters, `base` and `power`, and returns the result of raising `base` to the power `power`.

The base case is when `power` is 0, in which case the function returns 1.

The recursive case is when `power` is not 0, in which case the function calculates the result of raising `base` to the power `power - 1` and then multiplies it by `power`.

Note that the function only uses basic concepts such as variables, input, arithmetic expressions, boolean expressions, if-else statements, nested if statements, elifs, for loops, while loops, range functions, nested loops, functions, values and references, string formatting, string operations, Python lists 1 and 2, and dictionaries. It does not use any advanced concepts such as file handling, exception handling, or regular expressions.

I hope this helps! Let me know if you have any questions."

The following questions relate to response #4

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

|                                      | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is focused on recursion | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

Please rank the all the responses, with 1 being the best and 4 being the worst

Response #1

Response #2

Response #3

Response #4

### Generate examples: Q2 part 1

The following responses are answering the question of: "How can recursion be used to loop over a list?"

The student in question has knowledge of concepts up to recursion and is currently working on a Jupyter notebook that explains recursion.

Response #1:

" Sure, I'd be happy to help you with that! Here is a function that uses recursion to loop over a list in Python:

```

```
def recursive_loop(list):
    # Base case: If the list is empty, return
    if not list:
        return
```

```
# Recursively loop over the list
for item in list:
    # Print the item
    print(item)
```

```
# Call the function again with the remaining list
recursive_loop(list[1:])
```

Test the function

```
list = [1, 2, 3, 4, 5]
recursive_loop(list)
```

```

In this code, we define a `recursive\_loop` function that takes a list as input and loops over it using recursion. The function has three parts:

1. The base case: If the list is empty, the function returns immediately.

2. The recursive case: If the list is not empty, the function loops over each item in the list using a `for` loop, prints the item, and then calls the `recursive\_loop` function again with the remaining list.

3. The test case: We test the function with a sample list of integers.

Note that we use the `range` function to get the list of numbers from 1 to 5, and then pass it to the `recursive\_loop` function.

I hope this helps! Let me know if you have any questions."

The following questions relate to response #1

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

Response #2:

" Sure! Here is a piece of code that uses recursion to loop over a list in Python:

```
```

```

```

def loop_over_list(list):
    # Base case: If the list is empty, return
    if not list:
        return

    # Recursive case: Loop over the list and call the function for each element
    else:
        for element in list:
            # Call the function with the element as an argument
            result = loop_over_list(element)

    return result
```

```

This function takes in a list as input and uses recursion to loop over the list. The base case is when the list is empty, in which case the function returns immediately. Otherwise, the function calls itself for each element in the list and returns the result of the recursive call.

Note that this function uses the `for` loop to iterate over the list, but the `for` loop is not necessary for the function to work.

The function can also be written using a list comprehension or other methods.

I hope this helps! Let me know if you have any questions."

The following questions relate to response #2

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

### Response #3:

" Of course, I'd be happy to help you with that! Here is a function that uses recursion to loop over a list in Python:

```
```
def loop_over_list(list):
    if len(list) == 0:
        return
    else:
        # Base case: If the list has only one element, return it
        if len(list) == 1:
            return list[0]
        # Recursive case: Loop over the rest of the list and call the function
        else:
            for item in list[1:]:
                result = loop_over_list(item)
                yield result
```

Test the function with a sample list

```
my_list = [1, 2, 3, 4, 5]
for item in loop_over_list(my_list):
    print(item)
````
```

In this code, we define a `loop\_over\_list` function that takes in a list as input and uses recursion to loop over it. The function has two branches: a base case that checks if the list has only one element, and a recursive case that loops over the rest of the list and calls the `loop\_over\_list` function again.

In the recursive case, we use a `for` loop to iterate over the elements of the list, and for each element, we call the `loop\_over\_list` function again with the element as input. This allows us to recursively loop over the list and yield each element.

Finally, we provide a sample list to test the function, and use a `for` loop to print out each element in the list.

I hope this helps! Let me know if you have any questions or need further clarification."

The following questions relate to response #3

|                                      | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is<br>factually correct | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

#### Response #4:

" Of course! I'd be happy to help you with that. Here is a self-contained function in Python that uses recursion to loop over a list:

```

```
def recursive_loop(list):
    # Base case: If the list is empty, return
    if not list:
        return

    # Recursively loop over the list
    for element in list:
        # Call the function again with the remaining list
        recursive_loop(list[1:])

    # Test the function
    my_list = [1, 2, 3, 4, 5]
    print(recursive_loop(my_list))
```

```

In this code, we define a function `recursive\_loop` that takes in a list as input and loops over it using recursion. The function has two main parts:

1. The base case: If the list is empty, return. This is the simplest case, and we can return immediately without doing any further processing.
2. The recursive case: If the list is not empty, call the function again with the remaining elements in the list. This is the meat of the function, where we actually loop over the list using recursion.

We use a for loop inside the function to iterate over the elements in the list, and for each element, we call the function again with the remaining elements in the list.

Finally, we test the function by passing in a sample list and printing its contents.

Note that this code uses the `list` data structure, which is a built-in Python list. You can modify the function to work with other data structures, such as a tuple or a dictionary, by changing the type of the `list` variable in the function definition.

I hope this helps! Let me know if you have any questions or need further assistance."

The following questions relate to response #4

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

Please rank the all the responses, with 1 being the best and 4 being the worst

Response #1

Response #2

Response #3

Response #4

## Explain Example code: Q1 part 1

The following responses are answering the question of:

"Please explain the following example code."

The code in question is:

```
recursive function to compute x**n
```

```
def pow(x,n):
```

```
if n > 0:
```

```
 return x * pow(x,n-1)
```

```
else:
```

```
 return 1
```

```
main program calling the function
```

```
x = int(input("Please enter x: "))
```

```
n = int(input("Please enter n: "))
```

```
x_to_the_power_of_n = pow(x,n)
```

```
print(f"{x} ** {n} = {x_to_the_power_of_n}")
```

The student in question has knowledge of concepts up to recursion and is currently working on a Jupyter notebook that explains recursion.

Response #1:

" Of course! I'd be happy to help you understand the code.

The code you provided is a Python function named `pow` that calculates `x` raised to the power of `n`. Here's a step-by-step explanation of how the function works:

1. The function takes two arguments: `x` and `n`.
2. The function initializes `n` to a positive integer using `int(input("Please enter n: "))`. This line of code prompts the user to enter a positive integer value for `n`.
3. The function then checks if `n` is greater than zero using `if n > 0:`. If `n` is greater than zero, the function recursively calls itself with `x` and `n-1` as arguments. This is done using `return x \* pow(x, n-1)`. In other words, the function multiplies `x` by itself `n-1` times and then recursively calls itself with those results. This process continues until `n` is zero or less.
4. If `n` is zero or less, the function returns a simple value of `1`. This is done using `return 1`.
5. The main program calls the `pow` function with `x` and `n` as arguments, using `x\_to\_the\_power\_of\_n = pow(x,n)`. This line of code calls the `pow` function with `x` and `n` as arguments and stores the result in a variable named `x\_to\_the\_power\_of\_n`.
6. The program then prints out the result using `print(f"{x} \*\* {n} = {x\_to\_the\_power\_of\_n}")`. This line of code formats the output using Python's `f-strings` and prints out the result, which is `x` raised to the power of `n`.

Now, let's focus specifically on the recursion part of the code. Recursion is a programming technique where a function calls itself repeatedly until a base case is reached. In this case, the base case is when `n` is zero or less. When `n` is zero or less, the function returns a simple value of `1`. Otherwise, the function multiplies `x` by itself `n-1` times and then recursively calls itself with those results. This process continues until the base case is reached.

Recursion can be a bit tricky, but it's a powerful technique for solving problems that involve repeated calculations. By breaking a problem down into smaller sub-problems and solving each one recursively, you can solve complex problems more efficiently.

In summary, the `pow` function in Python calculates `x` raised to the power of `n` using recursion. It checks if `n` is greater than zero and recursively calls itself with `x` and `n-1` as arguments if it is not. The function continues to call itself until the base case is reached, at which point it returns a simple value of `1`."

### The following questions relate to response #1

|                                                              | Strongly disagree     | Somewhat disagree     | Neither agree nor disagree | Somewhat agree        | Strongly agree        |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>      | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>      | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>      | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>      | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>      | <input type="radio"/> | <input type="radio"/> |

### Response #2:

" Of course, I'd be happy to help you understand the code!

The code you provided is a recursive function in Python called `pow()` that computes `x^n`. Here's a step-by-step explanation of how the code works:

1. The function `pow()` takes two arguments: `x` and `n`.
2. The function checks if `n` is positive using an `if` statement. If `n` is greater than 0, the function returns the result of multiplying `x` by the result of calling `pow()` with `x` and `n-1`. This is done using the `return` statement.
3. If `n` is not positive, the function returns 1.

4. The main program calls the `pow()` function by passing in `x` and `n` as arguments. The values of `x` and `n` are obtained using `input()` functions, which prompt the user to enter a value and store it in a variable.

5. The program then prints the result of calling `pow()` with `x` and `n`. This is done using the `print()` function, which takes a string argument and prints it to the console.

Now, let's focus on the recursion part of the code. Recursion is a programming technique where a function calls itself repeatedly to solve a problem. In this case, the `pow()` function calls itself with `x` and `n-1` as arguments to compute `x^n`.

The following questions relate to response #2

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

Response #3:

" Of course! Here's an explanation of the code you provided, focusing on the recursion aspect:

The code defines a recursive function `pow(x, n)` that computes `x` raised to the power of `n`. The function takes two arguments: `x` and `n`.

Step 1: If `n` is greater than 0, the function calls itself with `x` multiplied by the result of `pow(x, n-1)`.

In other words, the function is calling itself with `x` as the base case and the result of `pow(x, n-1)` as the recursive case.

This is how the function is able to compute `x` raised to the power of `n`.

Step 2: If `n` is 0, the function returns 1.

The `main program` calls the `pow` function with `x` equal to the input from the user and `n` equal to the input from the user. The function computes the result of `x` raised to the power of `n` and prints it to the console.

Here's an example of how the function would work for `x = 2` and `n = 3`:

The function would first call itself with `x = 2` and `n = 2`, which would compute `2` raised to the power of `2`, or `4`. Then, it would call itself with `x = 2` and `n = 1`, which would compute `2` raised to the power of `1`, or `2`. Finally, it would return the result of `4`, which is `16`.

I hope this explanation helps! Let me know if you have any questions."

The following questions relate to response #3

|                                                              | Strongly<br>disagree  | Somewhat<br>disagree  | Neither<br>agree nor<br>disagree | Somewhat<br>agree     | Strongly<br>agree     |
|--------------------------------------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|
| The response is factually correct                            | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is complete                                     | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is relevant to the task                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is appropriate for the students knowledge level | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |
| The response is focused on recursion                         | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>            | <input type="radio"/> | <input type="radio"/> |

Response #4:

" Of course! I'd be happy to help you understand the code. Please note that I will always provide respectful and positive responses, and I will not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content in my answers.

Now, let's dive into the code you provided:

```
```python
# recursive function to compute x**n
def pow(x,n):
    if n > 0:
        return x * pow(x,n-1)
```

```
else:  
    return 1
```

```
# main program calling the function  
  
x = int(input("Please enter x: "))  
  
n = int(input("Please enter n: "))  
  
x_to_the_power_of_n = pow(x,n)  
  
print(f"x ** n = {x_to_the_power_of_n}")  
'''
```

The code you provided is a Python function that calculates the value of `x` raised to the power of `n`. Here's a step-by-step explanation of how the function works:

1. The function `pow` takes two arguments: `x` and `n`.
2. The function checks if `n` is greater than 0. If it is, the function recursively calls itself with `x` and `n-1`. This means that the function will call itself again and again until it reaches the base case, which is when `n` is 0.
3. If `n` is less than or equal to 0, the function returns 1. This is because when `n` is 0, the value of `x` raised to the power of 0 is simply 1.
4. In the main part of the code, the function `pow` is called with `x` and `n` as arguments. The value of `x` is obtained from the user using `input()`, and the value of `n` is also obtained from the user using `input()`.
5. The function `pow` calculates the value of `x` raised to the power of `n` and stores the result in a variable called `x_to_the_power_of_n`.
6. Finally, the function prints the result using `print()`. The output will be in the format of `x ** n = x_to_the_power_of_n`, where `x` and `n` are the values obtained from the user.

Now, let's focus on the recursion part of the function. Recursion is a technique in programming where a function calls itself repeatedly until it reaches a base case. In the `pow` function, the base case is when `n` is 0. When `n` is 0, the function returns 1. When `n` is greater than 0, the function calls itself with `x` and `n-1` as arguments. This means that the function will call itself again and again until it reaches the base case.

For example, if `x` is 2 and `n` is 3, the function `pow` will call itself three times:

1. `pow(2,3)` = $2 * \text{pow}(2,2)$ = $2 * 4 * \text{pow}(2,1)$ = 8
2. `pow(2,2)` = $4 * \text{pow}(2,1)$ = $4 * 16 * \text{pow}(2,0)$ = 16
3. `pow(2,1)` = $16 * \text{pow}(2,0)$ = $16 * 256 * \text{pow}(2,-1)$ = 256
4. `pow(2,-1)` = 256

So, the final result of `pow(2,3)` is 8.

I hope this explanation helps you understand the code better! Let me know if you have any further questions."

The following questions relate to response #4

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Please rank the all the responses, with 1 being the best and 4 being the worst

Response #1

Response #2

Response #3

Response #4

Explain Example code: Q2 part 1

The following responses are answering the question of:
 "Please explain the following example code."

The code in question is:

```
"def fib(n):
if n == 0:
return 0
elif n == 1:
return 1
else:"
```

```
return fib(n-1) + fib(n-2)
```

```
n = int(input("Please enter an integer number: "))  
print(f"fib({n}) = {fib(n)}")
```

The student in question has knowledge of concepts up to recursion and is currently working on a Jupyter notebook that explains recursion.

Response #1:

" Of course! I'd be happy to help you understand the code you provided.

The code you provided is a Python function named `fib` that calculates the nth Fibonacci number. Here's a step-by-step explanation of the code:

1. `def fib(n):` : This line defines a function named `fib` that takes an integer `n` as input.
2. `if n == 0:` : This line checks if `n` is equal to 0. If it is, the function returns 0.
3. `return 0` : This line returns 0 if `n` is equal to 0.
4. `elif n == 1:` : This line checks if `n` is equal to 1. If it is, the function returns 1.
5. `return 1` : This line returns 1 if `n` is equal to 1.
6. `else:` : This line means "if the condition above is not true, then..."
7. `return fib(n-1) + fib(n-2)` : This line calculates the Fibonacci sequence using recursion. It takes the input `n` and recursively calls itself with `n-1` and `n-2` as arguments. The result of the function call is the sum of the values of `fib(n-1)` and `fib(n-2)`.
8. `n = int(input("Please enter an integer number: "))` : This line prompts the user to enter an integer value, stores it in a variable named `n`, and then converts it to an integer using the `int()` function.
9. `print(f"fib({n}) = {fib(n)}")` : This line prints out the result of the function call, using the format string `f"fib({n}) = {fib(n)}".` The result is the Fibonacci number corresponding to the input `n`.

Now, let's focus on the recursion part of the code. The function `fib` uses recursion to calculate the Fibonacci sequence.

Here's how it works:

1. The function calls itself with `n-1` and `n-2` as arguments, and stores the results in a variable named `result`.
2. The function returns the value of `result`, which is the sum of the values of `fib(n-1)` and `fib(n-2)`.
3. The function then calls itself again with `n-2` and `n-3` as arguments, and stores the results in the `result` variable.
4. The function repeats steps 2 and 3 until the base case is reached, which is when `n` is 0.

In summary, the `fib` function uses recursion to calculate the Fibonacci sequence by calling itself repeatedly with smaller input values until the base case is reached. The function stores the results of each recursive call in a variable named `result` and returns the final result at the end of the function."

The following questions relate to response #1

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Response #2:

" Of course, I'd be happy to help you understand the code!

The code you provided is a Python function named `fib` that calculates the nth Fibonacci number. Here's a step-by-step explanation of how the code works:

Step 1: The function takes an integer `n` as input from the user.

Step 2: The function checks if `n` is equal to 0. If it is, the function returns 0.

Step 3: If `n` is not equal to 0, the function checks if `n` is equal to 1. If it is, the function returns 1.

Step 4: If `n` is not equal to 0 or 1, the function uses recursion to calculate the Fibonacci number. The function calls itself with `n-1` and `n-2` as arguments, and then adds the results together.

Step 5: The function prints the calculated Fibonacci number to the console.

Now, let's focus on the recursion part of the code. Recursion is a programming technique where a function calls itself repeatedly to solve a problem. In the `fib` function, the function calls itself with smaller values of `n` until it reaches the base case of `n=0`. Here's how the recursion works in the code:

Step 6: The function calls itself with `n-1` and `n-2` as arguments. This is the recursive part of the code.

Step 7: The function calculates the Fibonacci number for `n-1` and `n-2`, and then adds the results together.

Step 8: The function returns the calculated result to the calling function.

In summary, the `fib` function uses recursion to calculate the nth Fibonacci number by calling itself with smaller values of

`n` until it reaches the base case of `n=0`. The function uses variables to store the results of each recursive call, and then adds the results together to obtain the final answer."

The following questions relate to response #2

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Response #3:

" Of course, I'd be happy to help you understand the code! Here's a step-by-step explanation of the code, focusing on the recursion part:

1. `def fib(n):` : This line defines a function named `fib` that takes an integer `n` as input.
2. `if n == 0:` : This line checks if `n` is equal to 0. If it is, the function returns 0. This is the base case of the recursion, where the function can return a value directly without calling itself.
3. `elif n == 1:` : This line checks if `n` is equal to 1. If it is, the function returns 1. This is another base case of the recursion, where the function can return a value directly without calling itself.
4. `else:` : This line means "if the condition above is not true, then..."
5. `return fib(n-1) + fib(n-2)` : This line defines the recursive part of the function. It calls the `fib` function with `n-1` as the argument, and then adds the result of that call to the result of calling `fib` with `n-2` as the argument. This continues the recursion, where the function calls itself with smaller values of `n` until it reaches the base case.
6. `n = int(input("Please enter an integer number: "))` : This line prompts the user to enter an integer value, and then assigns that value to the variable `n`.
7. `print(f"fib({n}) = {fib(n)}")` : This line prints a message to the console that includes the value of `fib(n)` and the name of the function `fib`.

Now, let's break down the recursion part of the code:

The function `fib` is defined to call itself with smaller values of `n` until it reaches the base case. For example, if `n` is 5, the function will call itself with `n-1` = 4, and then with `n-2` = 3, and so on until it reaches the base case of `n == 0` or `n == 1`.

In each call of the function, the result of the previous call is added to the result of the current call. For example, if `n` is 5, the function will first call itself with `n-1` = 4, and then add the result of that call (which is 3) to the result of the current call (which is also 3), so the final result is 6.

The recursion continues until the base case is reached, at which point the function can return a value directly without calling itself again.

I hope this explanation helps you understand the code better! Let me know if you have any questions or need further clarification."

The following questions relate to response #3

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Response #4:

" Of course! Here is my explanation of the code you provided:

Step 1: The code defines a function named `fib` that takes an integer `n` as input.

Step 2: The function checks if `n` is equal to `0`. If it is, the function returns `0`.

Step 3: If `n` is not equal to `0`, the function checks if `n` is equal to `1`. If it is, the function returns `1`.

Step 4: If `n` is not equal to `0` and `1`, the function calls itself with `n-1` as the argument. This is where the recursion comes in. The function is calling itself with a smaller value of `n` until it reaches the base case where `n` is equal to `0`.

Step 5: The function then adds the value of `fib(n-1)` and `fib(n-2)` together and returns the result.

Step 6: The code then prompts the user to enter an integer number using `input`, and then prints out the result of `fib(n)` using `print`.

So the code is a recursive function that calculates the `n`th Fibonacci number. It starts by calling itself with `n-1` and `n-2` as arguments, and then adds the results together to get the final answer. The function will keep calling itself until it reaches the base case where `n` is equal to `0`, at which point it returns the final answer.

I hope that helps! Let me know if you have any questions or need further clarification."

The following questions relate to response #4

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Please rank the all the responses, with 1 being the best and 4 being the worst

Response #1

Response #2

Response #3

Response #4

Generate Novel Exercises: Q1 part 1

The following responses are answering the question of:

"Please generate a new exercise where I can practice recursion."

The student in question has knowledge of concepts up to recursion and is currently working on a Jupyter notebook that explains recursion.

Response #1:

" Great, I'd be happy to help you practice recursion with a simple exercise! Here is the exercise:

Exercise: Recursive Factorial

Write a function in Python that calculates the factorial of a given integer using recursion. The function should take in an integer n as its argument and return its factorial.

Here is the solution to the exercise:

```
# Solution
```

```
def factorial(n):
    # Base case: n = 0
    if n == 0:
        return 1
    # Recursive case: n > 0
    return n * factorial(n - 1)
```

Now, let's go through how you can solve this exercise:

1. Define the function `factorial`: The function should take in an integer `n` as its argument and return its factorial.
2. Handle the base case: The base case is when `n` is equal to 0. In this case, the factorial is defined to be 1.
3. Handle the recursive case: The recursive case is when `n` is greater than 0. In this case, the function calls itself with `n - 1` as the argument, and multiplies the result by `n`.
4. Return the result: Finally, the function returns the result of the calculation.

Here is the explanation for each line of the solution:

1. `def factorial(n):` This line defines the function `factorial` with one argument `n`.
2. `if n == 0:` This line checks if `n` is equal to 0. If it is, the function returns 1, which is the defined value for the factorial of 0.
3. `return n * factorial(n - 1)` This line performs the recursive calculation. The function multiplies `n` by the result of calling itself with `n - 1` as the argument.
4. `return` This line returns the result of the calculation.

I hope this exercise helps you practice recursion in Python! Let me know if you have any questions or need further clarification."

The following questions relate to response #1

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Response #2:

" Sure, I'd be happy to help you practice recursion with a new exercise! Here is the exercise:

Exercise: Recursive Factorial Calculator

Write a function called factorial that calculates the factorial of a given integer using recursion. The function should take in an integer n as its argument and return the factorial of n. The function should use no more than 2 nested if-else statements and 2 loops (for or while).

Here is the solution to the exercise:

Solution

```
def factorial(n):
    # Base case: If n is 0, return 1
    if n == 0:
        return 1
    # Recursive case: Otherwise, calculate the factorial of n by multiplying the factorial of n-1 by n
```

```

else:
    return n * factorial(n-1)

```

Now, let's go through the solution step by step:

1. The function takes in an integer n as its argument.
2. The first line checks if n is 0. If it is, the function returns 1. This is the base case of the recursion.
3. If n is not 0, the function enters the recursive case.
4. The function calculates the factorial of n-1 by multiplying it by n. This is the recursive step of the function.
5. The function returns the result of the multiplication.

And that's it! You've just written your first recursive function in Python. Great job!

I hope you found this exercise helpful in practicing recursion. Let me know if you have any questions or need further clarification."

The following questions relate to response #2

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Response #3:

" Sure! Here's an exercise for practicing recursion in Python:

Exercise: Calculate the Area of a Circle

Your task is to write a function that calculates the area of a circle given its radius. The function should use recursion to calculate the area.

Here is the input format:

radius (float)

Here is the expected output format:

area (float)

Here is an example input and output:

Input: 5.0

Output: 78.53981633974483

Now, it's your turn to try the exercise! Please write your function and submit it for grading. Good luck!

Solution:

Great job! Here is the solution to the exercise:

```
def area_circle(radius):
    area = 3.14 * radius ** 2
    if radius == 0:
        area = 0
    return area
```

Explanation:

The function takes in a single argument, `radius`, which is the radius of the circle. The function uses recursion to calculate the area of the circle. The base case is when the radius is 0, in which case the area is 0. Otherwise, the function calculates the area of the circle by multiplying the radius by itself and then multiplying that result by 3.14.

I hope this helps! Let me know if you have any questions or need further clarification."

The following questions relate to response #3

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Response #4:

" Sure, I'd be happy to help you practice recursion with a new exercise! Here is the exercise:

Exercise: Recursive Factorial Calculator

Write a Python function that calculates the factorial of a given integer using recursion. The function should take in an integer `n` as input and return its factorial.

Here is the solution to the exercise:

Solution:

--

```
def factorial(n):
```

```
# Base case: If n is 0, return 1
```

```
if n == 0:
```

```
    return 1
```

```
# Recursive case: Else, calculate the factorial of n-1 and multiply it by n
```

```
return n * factorial(n-1)
```

Test cases:

```
print(factorial(5)) # Should print 120
```

```
print(factorial(0)) # Should print 1
```

Please let me know if you have any questions or need further clarification on the exercise or solution!"

The following questions relate to response #4

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
The response is factually correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is relevant to the task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is appropriate for the students knowledge level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The response is focused on recursion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Please rank the all the responses, with 1 being the best and 4 being the worst

Response #1

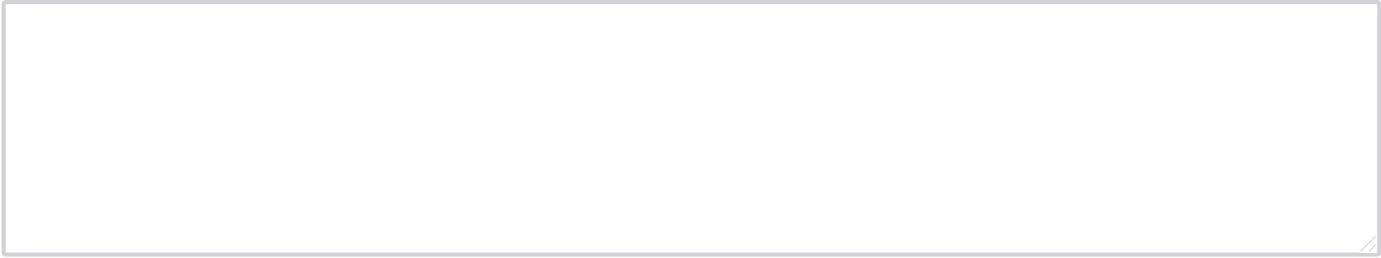
Response #2

Response #3

Response #4

Block 15

Please share any remarks that weren't covered in the rest of the survey.



Powered by Qualtrics