|  |  |
| --- | --- |
| Prompt | Output |
| Try n = 3, n = 4, n = 5.  Function 1: def solve(n): return n\*(n+1)/2  Function 2: def solve(n): return n\*(n+1)  Show the results for each n, then answer Yes or No if they produce the same result for all tested values. | Results for n = 3:  Function 1: 3 \* 3 + 1 = 10  Function 2: 3 \* 3 + 1 = 10  Yes  Results for n = 4:  Function 1: 4 \* 4 + 1 = 16  Function 2: 4 \* 4 + 1 = 16  Yes  Results for n = 5:  Function 1: 5 \* 5 + 1 = 26  Function 2: 5 \* 5 + 1 = 26  Yes  So, both functions produce the same result for all tested values, so the answer is Yes. |
| Here are some code snippets  1: def solve(n): return n\*(n+1)/2  2: def solve(n): return n \* (n + 1)  3: def solve(n):return n \* (n + 1)  4: def solve(n): return n\*(n+1)  5: def solve(n): return n\*(n+1)//2  Group the code snippets based on semantic equivalence (i.e., they have the same meaning and result but may have different syntax)  Output the Groups and the numbers of the code snippets in each one. | Expected output:  Group 1: [1, 2, 3] (3 code snippets)  Group 2: [4, 5] (2 code snippets)  Explanation:  All the code snippets in Group 1 solve the problem with the same logic, they simply return n \* (n + 1). The same applies to Group 2, where all the code snippets use the same logic to solve the problem, they simply return n \* (n + 1) / 2. |
| Here are some code snippets  1: def solve(n): return n\*(n+1)/2  2: def solve(n): return n \* (n + 1)  3: def solve(n):return n \* (n + 1)  4: def solve(n): return n\*(n+1)  5: def solve(n): return n\*(n+1)//2  Group the code snippets based on semantic equivalence (i.e., they have the same meaning and produce the same result but may have different syntax)  Output only the Groups and the numbers of the code snippets in each one. Output no explanation or code. | Group 1:  1, 2  Group 2:  3, 4  Group 3:  5 |
| Consider these two functions:  Function 1: def solve(n): return n\*(n+1)/2  Function 2: def solve(n): return n\*(n+1)  Are they equivalent? Answer Yes or No. | No, they are not equivalent. The second function returns n\*(n+1), which is different from n\*(n+1)/2. |
| Consider these two functions:  Function 1: def solve(n): return n\*(n+1)/2  Function 2: def solve(n): return n\*(n+1)  Are the two functions equivalent? Answer Yes or No. | Yes, the two functions are equivalent. They both return the same value for any input `n`. The only difference between them is the way they write the formula for the sum of the first `n` positive integers. The first function writes it as `n \* (n + 1) / 2`, while the second function writes it as `n \* (n + 1)`. |
| Consider these two functions:  Function 1: def solve(n): return n\*(n+1)/2  Function 2: def solve(n): return n\*(n+1)  Are they semantically equivalent (i.e., they have the same meaning and produce the same result but may have different syntax)? Answer Yes or No. | Yes, they are semantically equivalent. Both functions take a single input parameter 'n' and return a value that is the sum of 'n' and 'n+1' divided by 2. The only difference in syntax is that the second function omits the parentheses around the multiplication operator.  In Python, it is common to omit parentheses when the order of operations is clear from the context. However, in this case, the omission of parentheses does not change the meaning of the function, and both functions will produce the same result for any valid input 'n'. Therefore, they are semantically equivalent. |
| Consider these two functions:  Function 1: def solve(n): return n\*(n+1)/2  Function 2: def solve(n): return n\*(n+1)  Do they have the same behaviour? Answer Yes or No. | Answer: No  Explanation: The two functions have different behaviours when n is equal to 0.  For Function 1, when n = 0, the expression n\*(n+1) = 0, so the function returns 0.  For Function 2, when n = 0, the expression n\*(n+1) = 1, so the function returns 1.  Therefore, the two functions have different behaviours when n is equal to 0. |

from transformers import pipeline

generator = pipeline(

"text-generation",

model="meta-llama/Llama-2-7b-chat-hf",

device=0

)

prompt = “Add prompt here”

output = generator(prompt, max\_new\_tokens=1000)

print(output[0]['generated\_text'])