**Algebra Operations**

As there are many functions under this folder and writing report and test cases for them will be too extensive and will exceed the page limit, we will restrict testing to some common and important algebraic operations, specifically the gcd.py and is\_prime.py function.

Test cases for is\_prime.py will have *IP* in the beginning of test case ID while gcd.py will have GCD at the start of test case ID

***White box testing*:**

Test cases table for each method:

* Dirty testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Input** | **Expected** | **Actual** | **Description** |
| IP-1 | "hello" | TypeError | TypeError | Invalid input type: string. |
| IP-2 | None | TypeError | TypeError | Invalid input type: None. |
| IP-3 | 3.5 | TypeError | False | Invalid input type: float. |
| IP-4 | -17 | FALSE | FALSE | Negative input: no negative number is prime. |
| IP-5 | 0 | FALSE | FALSE | Edge case: 0 is not prime. |
| IP-6 | 1 | FALSE | FALSE | Edge case: 1 is not prime. |
| IP-7 | 9 | FALSE | FALSE | Non-prime odd number. |
| IP-8 | 10\*\*18 + 1 | FALSE | FALSE | Very large composite number, tests performance and correctness. |
| GCD-1 | (10, "5") | TypeError | TypeError | Invalid input type: string in place of an integer. |
| GCD-2 | (10, None) | TypeError | 10 | Invalid input type: None. |
| GCD-3 | (10.5, 5) | TypeError | 0.5 | Invalid input type: float. |
| GCD-4 | (-10, 5) | 5 | 5 | Negative input: should return absolute GCD. |
| GCD-5 | (0, 5) | 5 | 5 | GCD involving zero and non-zero numbers. |
| GCD-6 | (0, 0) | 0 | 0 | Edge case: GCD of two zeros. |
| GCD-7 | (10\*\*18, 10) | 10 | 10 | Large inputs to test performance and correctness. |

* Boundary value analysis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Input** | **Expected** | **Actual** | **Description** |
| IP-BVA-1 | 1 | FALSE | FALSE | Boundary value below smallest prime. |
| IP-BVA-2 | 2 | TRUE | TRUE | Smallest prime number. |
| IP-BVA-3 | 3 | TRUE | TRUE | Small prime number, valid boundary. |
| IP-BVA-4 | 4 | FALSE | FALSE | First composite after 2 (boundary test). |
| IP-BVA-5 | 1.00E+19 | FALSE | TypeError | Large boundary to test performance. |
| GCD-BVA-1 | (0, 0) | 0 | 0 | Both inputs are zero. |
| GCD-BVA-2 | (0, 5) | 5 | 5 | One input is zero, positive boundary. |
| GCD-BVA-3 | (0, -5) | 5 | -5 | One input is zero, negative boundary. |
| GCD-BVA-4 | (-10, 5) | 5 | 5 | Negative input with valid positive. |
| GCD-BVA-5 | (10\*\*18, 5) | 5 | 5 | Large positive number as boundary test. |

* Extreme point combination:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Input (n)** | **Expected** | **Actual** | **Description** |
| IP-EPC-1 | 1 | FALSE | FALSE | Below smallest valid input. |
| IP-EPC-2 | 2 | TRUE | TRUE | Smallest valid prime number. |
| IP-EPC-3 | 10\*\*9 + 7 | TRUE | TRUE | Large known prime number. |
| IP-EPC-4 | 10\*\*9 + 6 | FALSE | FALSE | Large known composite number. |
| GCD-EPC-1 | (0, 0) | 0 | 0 | Both inputs are zero. |
| GCD-EPC-2 | (0, 1) | 1 | 1 | One input is zero, small positive input. |
| GCD-EPC-3 | (1, 0) | 1 | 1 | Small positive input, other is zero. |
| GCD-EPC-4 | (-1, 1) | 1 | 1 | Small negative input and small positive. |
| GCD-EPC-5 | (10\*\*9, 1) | 1 | 1 | Large positive and small positive input. |
| GCD-EPC-6 | (-10\*\*9, 10\*\*9) | 10\*\*9 | 1000000000 | Large negative and large positive input. |

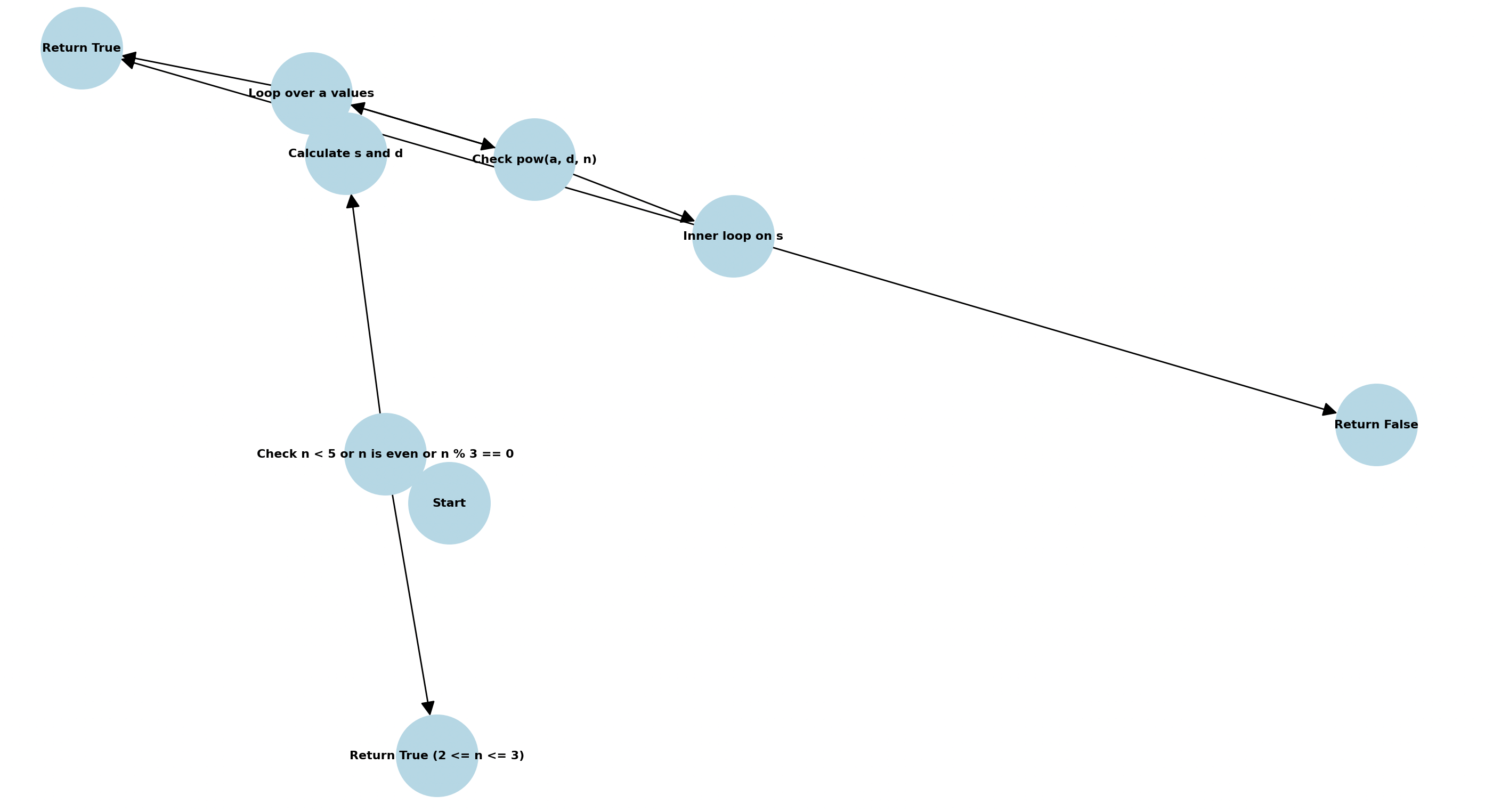
* Weak nx1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Input (n)** | **Expected** | **Actual** | **Description** |
| IP-WN1-1 | 2 | TRUE | TRUE | Smallest valid prime number. |
| IP-WN1-2 | 1 | FALSE | FALSE | Smallest non-prime number. |
| IP-WN1-3 | 10\*\*9 + 7 | TRUE | TRUE | Large prime number. |
| IP-WN1-4 | 10\*\*9 + 6 | FALSE | FALSE | Large composite number. |
| GCD-WN1-1 | (0, 5) | 5 | 5 | One input is zero, nominal value for y. |
| GCD-WN1-2 | (5, 0) | 5 | 5 | Nominal value for x, other input is zero. |
| GCD-WN1-3 | (-10\*\*9, 5) | 5 | 5 | Large negative input for x. |
| GCD-WN1-4 | (10\*\*9, 5) | 5 | 5 | Large positive input for x. |
| GCD-WN1-5 | (5, -10\*\*9) | 5 | -5 | Large negative input for y. |
| GCD-WN1-6 | (5, 10\*\*9) | 5 | 5 | Large positive input for y. |

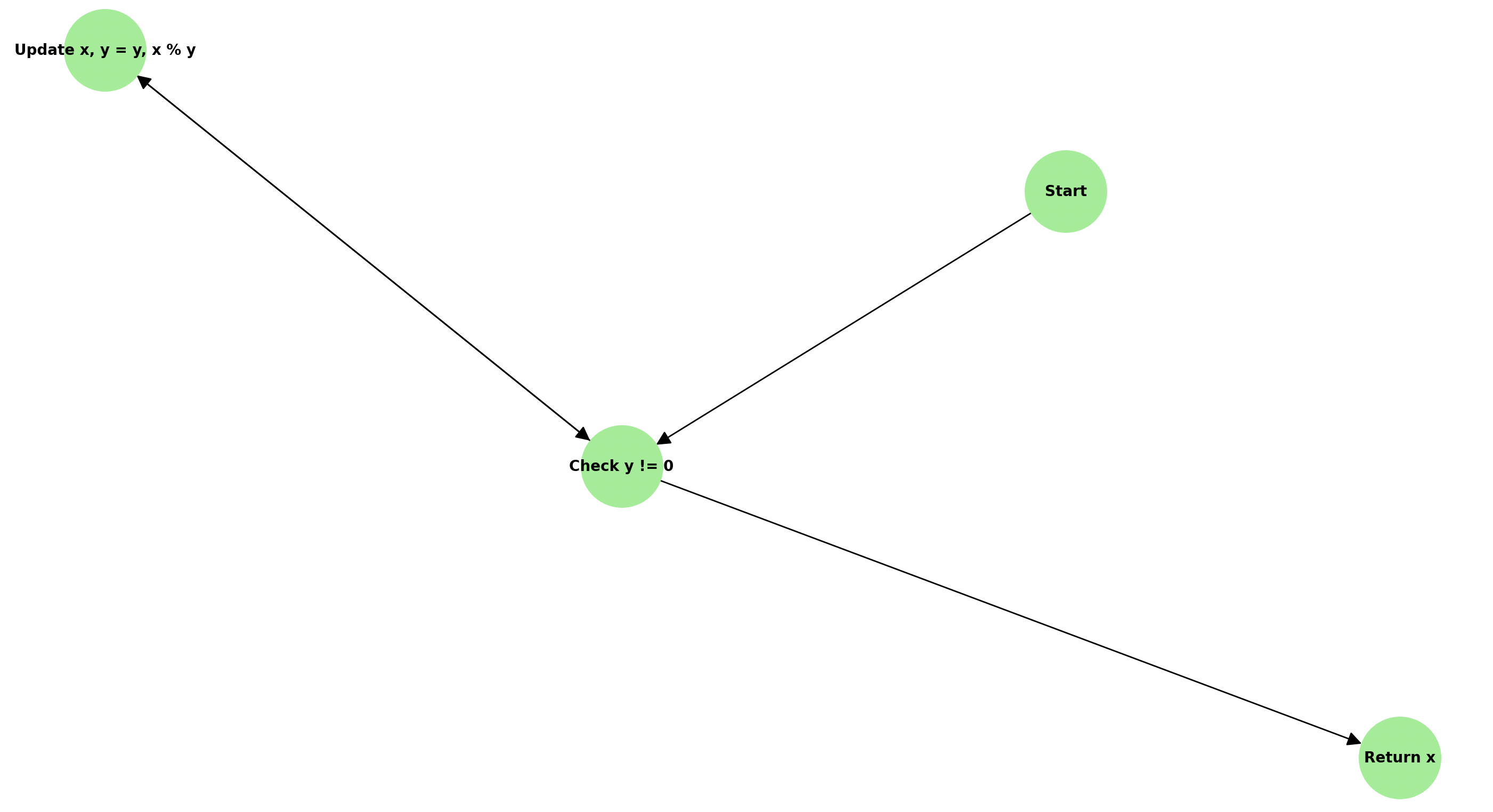
* Pairwise testing:
  + For is\_prime.py, pairwise testing is not directly applicable for the function as there is only 1 input for the function
  + For gcd.py with 2 inputs variables, pair wise testing is also not that effective as the method considers all pairs of input variables and in this case, pairwise testing will not help optimize and reduce the number of cases.

***Black box testing***:

* Integration testing is not applicable in this case as the functions provide independent computations and only require some external libraries from Python but no other modules in the library. Therefore, methods like Big Bang testing, bottom-up or top-down integration testing is not a viable option in this case. However, we can still perform black box testing on these functions but unit testing, making sure to strive for full coverage.
* At first, we let the agent to generate the control flow of the functions before moving forward
* We can see that the agent did a sufficient job generating the control flow diagrams. However, the diagrams still do not go into details into line-by-line logic of the functions, some lack a fair chunk of code through compacting multiple lines of code into 1 node.
* Here are the diagrams for:
  + is\_prime.py



* + gcd.py



* The unit test cases for the 2 functions:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Input (n)** | **Expected** | **Actual** | **Description** |
| IP-TC-1 | 0 | FALSE | False | Testing input less than 2 (non-prime). |
| IP-TC-2 | 1 | FALSE | False | Testing input less than 2 (non-prime). |
| IP-TC-3 | 2 | TRUE | True | Smallest prime number. |
| IP-TC-4 | 3 | TRUE | True | Second smallest prime number. |
| IP-TC-5 | 4 | FALSE | False | Smallest composite number > 2. |
| IP-TC-6 | 6 | FALSE | False | Even composite number > 2. |
| IP-TC-7 | 29 | TRUE | True | Random small prime number. |
| IP-TC-8 | 28 | FALSE | False | Random composite number. |
| IP-TC-9 | 104729 | TRUE | True | Large known prime number. |
| IP-TC-10 | 104730 | FALSE | False | Large known composite number. |
| GCD-TC-1 | (0, 0) | 0 | 0 | Both inputs are zero. |
| GCD-TC-2 | (0, 5) | 5 | 5 | One input is zero, the other positive. |
| GCD-TC-3 | (7, 0) | 7 | 7 | One input is zero, the other positive. |
| GCD-TC-4 | (-10, 5) | 5 | 5 | Negative and positive inputs. |
| GCD-TC-5 | (-10, -5) | 5 | -5 | Both inputs are negative. |
| GCD-TC-6 | (13, 7) | 1 | 1 | Prime numbers with no common divisor. |
| GCD-TC-7 | (100000000, 50000000) | 50000000 | 50000000 | Large inputs with a common divisor. |
| GCD-TC-8 | (24, 36) | 12 | 12 | Positive inputs with a common divisor. |
| GCD-TC-9 | (-24, 36) | 12 | 12 | Negative and positive inputs with a divisor. |

***Conclusion***:

* From various testing methods, we observed that the is\_prime.py and gcd.py functions do not validate whether the input values are integers before performing operations. This oversight leads to errors in test cases involving invalid input types, as the functions deviate from the mathematical definitions that only integers and whole numbers can undergo these operations.
* We also observed that is\_prime.py returns Type Error for only a certain of float numbers specifically floating point numbers under scientific exponential notation like 10e13 or 5e2, but fails to do so for other kinds of floating numbers.

***ChatGPT use***:

* Overall, ChatGPT did an OK job at generating test cases for most testing methods, although in some cases, the agent fails to recognize the limitations of the program inputs that make them impossible to be tested with some methods.
* For example, ChatGPT had some trouble recognizing that both functions cannot take full advantage of all pairs testing or integration testing due to the reasons described above.
* In some testing methods, ChatGPT completely failed to come up with test cases that meet the requirements or satisfy the testing methods’ definitions. This is partly due to the lack of documentations and materials of the testing methods online available for the model to train on. This is the case with extreme point combination, boundary value analysis or weak nx1 testing where the agent either failed to come up with appropriate cases or came up with test cases that were not consistent with methods’ definitions
* One way to mitigate this limitation is to provide ChatGPT the right documentations and definitions of the testing methods before prompting it to generate test cases, this can either be a series of explanation prompts or uploading a document with testing methods explanations and examples inside. Then the test cases the agent generated for the explained methods become much more relevant and consistent with the definitions.
* Using ChatGPT to generate test cases can he helpful and effective. However, prior precautions and careful supervision must be applied to assess the quality of responses from the agent and all relevant materials plus explanations for testing methods specifically should be provided to the agent to ensure the most relevant responses. Alternatively, examples can also be provided so that the agent can model after the given example for the generated test cases
* To ensure maximum repeatability and consistency in test cases generation and their relevance to the right testing methods, prior documentations and examples should be provided and explained to the agent before prompting it to generate actual test cases as the materials the model is trained on may vary from time to time. Giving it examples and documentations beforehand to model after allows the LLM a robust source of information and instructions to follow, improving the consistency and relevance of the generated results.