Binary Tree Expression Evaluator

You are tasked with developing a simple symbolic expression evaluator using a binary tree. Each node in the binary tree represents either an operand (a number) or an operator (addition `'+'`, subtraction `'-'`, multiplication `'*'`, or exponentiation `'^'`). Your goal is to construct the binary tree, evaluate the expression stored in it, and print the result.

The result should be printed as follows:

- If the result is between 0 and 999 (inclusive), print the value.
- If the result is less than 0, print `"UNDERFLOW"`.
- If the result exceeds 999, print `"OVERFLOW"`.

Function Description

Complete the `'Node'` class by implementing the following methods:

- `'__init__(self, data)'`: Initializes a new node with the given data.
- `'insert(self, data, bracketed)'`: Inserts a new node into the binary tree based on the given data and whether it's bracketed.
- `'evaluate(self)'`: Evaluates the expression represented by the binary tree and returns the result.
- `'get_output(self)'`: Prints the final output based on the evaluation.

Input Format

- The input will consist of a series of method calls to the `'Node'` class, where each method modifies the binary tree or evaluates the expression.

Output Format

- Print the evaluated result, `"OVERFLOW"`, or `"UNDERFLOW"` as per the criteria.

Constraints

- The tree will only contain valid operators (`+`, `-`, `*`, `^`) and operands (integers).
- The operations should be performed following standard operator precedence, and exponents should be calculated as `'a^b'`.

Sample Input & Output

Input	Output

`root = Node(('OPERAND', 10))`
`root = root.insert(('OPERATOR', '^'), False)`
`root = root.insert(('OPERAND', 10), False)`
`root = root.insert(('OPERATOR', '+'), False)`
`root = root.insert(('OPERAND', 900), False)`
`root.get_output()`
	`OVERFLOW`

`root = Node(('OPERAND', 1))`
`root = root.insert(('OPERATOR', '+'), False)`
`root = root.insert(('OPERAND', 2), False)`
`root = root.insert(('OPERATOR', '*'), True)`
`root = root.insert(('OPERAND', 3), False)`
`root = root.insert(('OPERATOR', '+'), False)`
`root = root.insert(('OPERAND', 3), False)`
`root = root.insert(('OPERATOR', '^'), False)`
`root = root.insert(('OPERAND', 2), False)`
`root.get_output()`
	`100`

Explanation

Case 1:

In this example, the binary tree represents the expression $10^{10}+900$. The value of 10^{10} is 10 billion, which exceeds 999, so the output is `"OVERFLOW"`.

Case 2:

In this example, the binary tree represents the expression $1+2\times3+3^2$. The evaluation follows standard precedence rules, leading to the result $1+6+9=16$. This result is within the range of 0 to 999, so the output is the value itself, `‘16’`.