Calculator

Complete the function `calculate` in the editor. The function takes a mathematical expression string as input and should return the result as an output in string format.

Question 1

Solve simple expressions with a single operator and two operands. All binary operations are permissible (addition, subtraction, multiplication, division and exponentiation).

Acceptable input

- Single operator: + - * / ^
- Positive integers
- White space (should be ignored)

Sample input and expected output

Input	Output
`2 + 5`	7
`8 – 3`	5
`5 * 4`	20
`8 / 2`	4
`4 ^ 2`	16

Question 2

Solve expressions with multiple operators. Expressions may now also contain parentheses, which have a higher precedence level than the other operators.

Acceptable input

- One or more operators: + - * / ^
- Parentheses: ( )
- Positive integers
- White space (should be ignored)

Sample input and expected output

Input	Output
`1 + 2 * 3`	7
`(1 + 2) * 3`	9
`6 + 3 – 2 + 12`	19
`2 * 15 + 23`	53
`10 – 3 ^ 2`	1

Question 3

Expressions may now also contain non-integer numbers, as well as negative numbers and 0.

Acceptable input

- One or more operators: + - * / ^
- Parentheses: ( )
- Integers and decimal numbers
- White space (should be ignored)

Sample input and expected output

Input	Output
`3.5 * 3`	`10.5`
`-53 + -24`	`-77`
`10 / 3`	`3.333`
`(-20 * 1.8) / 2`	`-18`
`-12.315 - 42`	`-54.315`

Question 4

Get creative! Which other features would you want to add to this program? Some ideas:

- Add a way to set the output precision of floating-point numbers
- Add support for hexadecimal numbers
- Functions: e.g. sin(x), cos(x), tan(x), log(x)
- Factorials!

Things to consider

Error handling

How does your code handle invalid input or undefined results?

Order of operations

1. Brackets (parentheses)
2. Order (exponents)
3. Division and Multiplication
4. Addition and Subtraction

Postfix (Reverse Polish Notation)

One potential way to simplify the parsing is to convert expressions to postfix notation. For example, `2 + 3 * 5` would become `2 3 5 * +` in postfix format.

Unit Tests

As well as this document, you will receive a file called `tests.txt`. Using your testing framework of choice, you should read in the contents of this file and use it as data to test your calculator. Each line in the file represents a single equation to test. The equation to test will be separated from the expected answer by a colon character (which serves no other purpose than as a delimiter).

For example:

```
1+2+3 : 6
```

This would be parsed by your test code as the equation “1 + 2 + 3”, with an expected answer of “6”.