Click to verify

Have you ever written a complex mathematical expression in Python and gotten the wrong answer? It might be due to a misunderstanding of the order of operations in Python. In this article, we'll go over the basics of Python order of operations and how to use parentheses to control the order of evaluation. It is a defined set of rules that dictate the order in which mathematical operations are performed. In mathematics, we often represent the order of operations by the acronym PEMDAS. The order of operations is similar in Python, but with a few differences: Parentheses Exponents Multiplication, Division, and Modulo (from left to right) Addition and Subtraction (from left to right) Let's take a look at some examples to see how this works in practice. The first step in evaluating an expression in Python is to evaluate any expressions inside parentheses. For example: In this expression, the addition of 2 + 3 is evaluated first, resulting in 5. The expression inside the parentheses is then treated as a single value, so 5 * 4 is evaluated next, giving us the final result of 20. Exponents in Python are evaluated next. The operator for exponents is **. For example: In this expression, 2 is the base and 3 is the exponent, resulting in 8. Next, Python evaluates any expressions involving multiplication, division, or modulo. We perform multiplication and division from left to right. For example: In this expression, 4 * 3 is evaluated first, resulting in 12. Then, 12 / 2 is evaluated, giving us the final result of 6.0. To get the remainder of a division operation, we use the modulo % operator. For example: In this expression, 7 divided by 3 has a remainder of 1, so 1 is the result. Finally, the language evaluates any remaining expressions involving addition and subtraction from left to right. For example: In this expression, 2 + 3 is evaluated first, resulting in 5. Then, 5 - 1 is evaluated, giving us the final result of 4. When writing complex expressions in Python, it can be helpful to use parentheses to control the order of evaluation. For example: In this expression, the expression inside the parentheses is evaluated first, resulting in 7. Then, 2 * 7 is evaluated, giving us the final result of 14. Another helpful tip is to use whitespace to make your code easier to read. For example: >>> 2 * (3 + 4) / 2 - 3 5.0 In this expression, the parentheses are evaluated first, resulting in 7. Then, 2 * 7 / 2 is evaluated, giving us 7.0. Finally, 7.0 - 3 is evaluated, giving us the final result of 5.0. We use parentheses and whitespace to make complex expressions easier to read and understand. This prevent mistakes and makes our code more maintainable. and operator has a higher precedence than the or operator. This also means that when you have both and and or operators in an expression , the and operator evaluates first following the or operator. Here is an example: x = 2 y = 3 z = 4 x == 2 or y == 4 and z == 4 #Output True In Python, boolean expressions are calculated using the order of operations defined by the language. This means that certain operations are performed before others based on their precedence in the hierarchy of operators. The order of precedence for boolean operators in Python is as follows: not has the highest precedence and has higher precedence than or or has the lowest precedence This means that in a boolean expression with multiple operators, the not operator is evaluated first, then the and operator, and finally the or operator. For example, consider the following boolean expression: not True or False and True According to the order of precedence, the not operator is evaluated first, so the expression is equivalent to: Next, the and operator is evaluated, so the expression is equivalent to: Finally, the or operator is evaluated, so the entire expression evaluates to False. In Python, most operations are evaluated from left to right. This means that when you have an expression with multiple operators of the same precedence, the operators are evaluated in the order they appear in the expression, from left to right. For example, consider the following expression: Since the - and + operators have the same precedence, we evaluate them from left to right. This means that the expression is equivalent to: So the result is 8. However, there are some operators in Python that do not follow the left-to-right evaluation order. Like, the exponentiation operator ** has a higher precedence than the arithmetic operators, which is evaluated from right to left. This means that an expression like 2 ** 3 ** 2 is evaluated as 2 ** (3 ** 2), which is 2 ** 9 or 512. How does the order of operations in Python compare to other programming languages? The order of operations in Python is similar to most other programming languages, including C, Java, and JavaScript. The main difference is the order of precedence of operations, which can vary from language to language. Can I use whitespace to make my code more readable? Yes, you can use whitespace to make your code more readable by adding spaces between operators and operands, and using newlines to break up long expressions into multiple lines. Understanding the order of operations in Python is important for writing correct and efficient code. You can write complex mathematical expressions by using parentheses and order of operations to control the order of evaluation with confidence. Remember, the order of operations in Python is: Parentheses Exponents Multiplication, Division, and Modulo (from left to right) Addition and Subtraction (from left to right) By following these rules and using parentheses and whitespace, you can write complex expressions that are easy to read and understand. Beginner Video Tutorial Videos can also be accessed from our Full Stack Playlist 3 on YouTube. Python math operators and PEMDAS order of operations | Python for Beginners (4:50) Code Examples and Video Script Welcome. Today's question: What are the rules for Math in Python? I'm Paul, and many of us learn the rules of Math, first on paper, then in a calculator, followed by a spreadsheet, and all of these were much easier than in a programming language. At least, that's what I found. So here I hope to make your journey less difficult than mine, by highlighting Python's seven, out of the box, basic operations. At high school in California, we use PEMDAS to memorize the order of operations for math, and we'll see if Python conforms. We will talk about two numerical data types: integers and floats, and see how we keep them straight with practice in Python. (Commands in Linux) (Lessons in Python) # ( ) ** * / // + - exit() Next we will cover another subject in Math: relational operators. Step 1 - Basic Math Using Python Operators In Project 3 (Python for Beginners) so far we installed python3 and now we'll dive into, what to me, is the most exciting part, hands on learning in Python. paul@fullstack:~$ less notes/video0023.txt Heading to the Terminal, let's review PEMDAS, which stands for parentheses, exponent, multiplication, division, addition and subtraction. Rules for math in Python 3 (Video 23) Order of operation - PEMDAS 1. Parentheses ( ) 2. Exponent ** 3. Multiplication * 4. Division / // % 5. Addition + 6. Subtraction - Notes: - after PEMDAS order goes left to right - use parentheses to override order The basic numerical data types - integer - float (floating point, decimal) Notes: - any input of float results in a float - in Python 3 division results in a float - integers only result in integers (except for division) notes/video0023.txt (END) It details the order of operations, and also note, in Python we have three types of division, regular division, floor division and finding the remainder, using what's called modulo. Step 2 - Python PEMDAS Order of Operation Let's head to the the python3 Interpreter and cover the rest of this. The first way to interact with Python, is like this, one line at a time. Second is a script, or text file, which is the focus of a future Project. So the three greater-than symbols >>> are a Python signature, like the command line in Linux, meaning it's waiting for us. paul@fullstack:~$ python3 Python 3.4.2 (default, Oct 8 2014, 10:45:20) [GCC 4.9.1] on linux Type "help", "copyright", "credits" or "license" for more information. >>> #PEMDAS backwards ... The hash symbol #, or comment, too is similar, meaning the rest of the line is ignored. Three dots ... is a continuation prompt, meaning Python doesn't have enough to act on. Python Subtraction So let's start with the last letter of PEMDAS, subtraction, give it a simple 1 - 2. ... 1 - 2 -1 >>> And voilà -1. Python Addition Next, A for addition. We could enter 1+1 like this, and it works. >>> 1+1 2 Or 1 +1 like this, and it too works. >>> 1 +1 2 Or 1 + four spaces 1. >>> 1 + 1 2 And it also works, but the preferred form is to put a space between each. >>> 1 + 1 2 Next, D, division, and here I should mention the data types: integers and floats. We've been playing with integers so far. Let's try a floating point number like 6.0 divided by the integer 2. >>> 6.0 / 2 3.0 And we get 3.0, a float. Python will automatically Interpret the format for the resulting number based on how you input numbers. So input of any float results in a float. >>> 6 / 2 3.0 Also, with division, in Python 3, at least, something like integer 6 divided by integer 2, equals float 3.0. Floor division rounds down, so 7 floor, divided by 3 is two and a third (2.3333333333333335). >>> 7 / 3 2.3333333333333335 >>> 7 // 3 2 And 7, floor divided by 3 is 2, which ignores the remainder. Python modulo division To see the remainder only, we use the modulo, or percent sign %. >>> 7 % 3 1 So 7 % 3 is 1 because 3 goes into 7 two times with one left over. Modulo can help us determine if a number is positive or negative. Python multiplication Next, M, multiplication is pretty straightforward, so two integer 5s equals 25. >>> 5 * 5 25 And one integer 5 and one float 5.0 equals float 25.0. >>> 5 * 5.0 25.0 Next, E, for exponent, is entered with two stars **, so 2 to the second power is 4. >>> 2 ** 2 4 And to the third power is 8. >>> 2 ** 3 8 What would be 9 to the power of .5, or one half power? >>> 9 ** .5 3.0 Remember it (a number to the 1/2 power) is the square root. Python parentheses Next, P, for parentheses. So after following the rules of operations Python will work from left to right. So we know 1 + 2 * 4 is going to give us a different answer than (1 + 2) * 4. Right? >>> 1 + 2 * 4 9 >>> (1 + 2) * 4 12 I'll ask you to do the last one for homework. >>> 5 + (4 - 2) * 2 + 4 % 2 - 4 // 3 - (5 - 3) / 1 Do it by hand on paper before using Python, and there are a couple little tricks in there that will reinforce your understanding. Feel free to pause now, or rewind it. And to leave the Python Interpreter type exit() followed by Enter. >>> exit() paul@fullstack:~$ You are welcome to join our journey to Data Science as we take one step at a time and build out our Full Stack. Client : HTML, CSS, JavaScript Software : Python Scientific Stack Data : PostgreSQL, MySQL / MariaDB OS : Linux (command line), Debian And our next step is to talk about Relational Operators. Have a nice day. The order of operations in Python follows the PEMDAS/BODMAS acronym, which stands for:P/B - Parentheses/Brackets: Evaluate expressions inside parentheses or brackets first.E/O - Exponents/Orders: Evaluate exponentiation (or orders, such as square roots) next.MD - Multiplication and Division: Perform multiplication and division from left to right.AS - Addition and Subtraction: Perform addition and subtraction from left to right.Here's an example to illustrate the order of operations:result = 5 + 3 * 2 ** 2 / 4 Following the order of operations:Exponentiation: 2 ** 2 equals 4Multiplication: 3 * 4 equals 12Division: 12 / 4 equals 3Addition: 5 + 3 equals 8So, the result is 8. If you want to change the order of evaluation, you can use parentheses to group expressions and force a specific order:result = (5 + 3) * 2 ** (2 / 4) In this case:Parentheses: 5 + 3 equals 8Exponentiation: 2 ** 0.5 equals 2Multiplication: 8 * 2 equals 16So, the result is 16. Using parentheses helps to clarify the intended order of operations and ensures that the expressions are evaluated as desired.Uses of order of opreationsThe order of operations in Python is crucial for determining the correct sequence in which expressions are evaluated. Understanding and applying the order of operations ensures that mathematical expressions are calculated accurately. Here are some common use cases for the order of operations in Python:Arithmetic Expressions: result = 5 + 3 * 2 Here, the multiplication (3 * 2) is performed before the addition (5 + ...), following the of operations.Formulas and Equations: area = 2 * 3.14 * radius ** 2 This example involves multiplication (2 * 3.14), exponentiation (radius ** 2), and the overall multiplication of these factors. total_cost = unit_price * quantity + shipping_cost The multiplication (unit_price * quantity) is done before addition (... + shipping_cost). print("Condition satisfied.") The expression 2 * y is calculated before the comparison (x > ...). result = math.sqrt(4 * x + y) The expression 4 * x + y is evaluated before being passed as an argument to the math.sqrt function. Using parentheses helps to explicitly define the order of operations and avoids ambiguity.Algorithmic Calculations: In various algorithms, mathematical operations are performed in a specific order to achieve the desired computation.complex_result = (a + b) * c - d ** 2 / (e + f) The order of operations is crucial for correctly evaluating complex expressions.Order of Operations in Python:Follows the PEMDAS/BODMAS acronym: Parentheses/Brackets, Exponents/Orders, Multiplication and Division, Addition and Subtraction.Purpose:Crucial for accurately evaluating mathematical expressions in the correct sequence.Parentheses:Expressions inside parentheses are evaluated first in Python.Changing Order:Use parentheses to override the default order of operations and specify a different sequence.PEMDAS/BODMAS Acronym:Stands for Parentheses/Brackets, Exponents/Orders, Multiplication and Division, Addition and Subtraction.Exponents:Exponentiation is performed before other operations.Multiplication and Division:Performed from left to right in Python expressions.Conditional Statements:Order of operations applies to expressions within conditional statements.Complex Expressions:Order of operations is crucial for correctly evaluating complex mathematical expressions.Preventing Ambiguity:Use parentheses to clarify the intended order of operations and prevent ambiguity in expressions.Numeric Calculations:Multiplication and division precede addition and subtraction in numeric calculations.Algorithmic Calculations:Order of operations is vital in various algorithms involving mathematical computations.Comments in Python:Use comments sparingly; focus on explaining why something is done rather than what the code is doing.Triple-Quoted Strings:Triple-quoted strings can be used for multiline comments in Python.Best Practices:Good code organization, meaningful names, and well-documented code reduce the need for excessive comments.Financial Calculations:Order of operations is applicable to financial calculations, ensuring accuracy in interest rates, payments, etc.Consistency Across Versions:The order of operations is consistent between Python 2 and Python 3; however, use the appropriate version, as Python 2 is no longer supported.What is the order of operations in Python?The order of operations in Python follows the PEMDAS/BODMAS acronym: Parentheses/Brackets, Exponents/Orders, Multiplication and Division, Addition and Subtraction.Why is the order of operations important in Python?The order of operations is crucial for accurately evaluating mathematical expressions and ensuring that calculations are performed in the correct sequence.How does Python handle parentheses in expressions?Expressions inside parentheses are evaluated first in Python, following the "Parentheses" rule of the order of operations.Can I change the default order of operations in Python?Yes, you can use parentheses to group expressions and force a specific order of evaluation in Python.What is the purpose of the acronym PEMDAS/BODMAS? PEMDAS/BODMAS stands for Parentheses/Brackets, Exponents/Orders, Multiplication and Division, Addition and Subtraction, serving as a mnemonic for the order of operations.Give an example of using exponents in Python.result = 2 ** 3 calculates 2 raised to the power of 3, resulting in 8.How are multiplication and division handled in Python expressions?Multiplication and division are performed from left to right in Python expressions.How can I use parentheses to override the default order of operations?By placing expressions inside parentheses, you can ensure that those expressions are evaluated first.Can I use the order of operations in conditional statements?Yes, expressions in conditional statements follow the order of operations. For example, if x > 2 * y: evaluates the multiplication before the comparison.Why is the order of operations essential in mathematical calculations?The order of operations ensures that mathematical expressions are evaluated consistently and accurately, preventing ambiguity.Are there exceptions to the order of operations in Python?No, the order of operations is fundamental in Python, and expressions are always evaluated following the PEMDAS/BODMAS rules.How does Python handle complex mathematical expressions?Python evaluates complex expressions by following the order of operations, which helps in correctly calculating the result.What happens if I don't use parentheses in my expressions?Without parentheses, Python follows the default order of operations to evaluate expressions.Is the order of operations specific to Python, or is it a general mathematical rule?The order of operations is a general mathematical rule that applies to various programming languages, not just Python.Can I use the order of operations in algorithms?Yes, algorithms often involve mathematical operations, and the order of operations is applied to achieve the desired computation.Are there built-in functions in Python that rely on the order of operations?Functions like math.sqrt may involve expressions that follow the order of operations when calculating results.How does Python handle expressions involving both addition and multiplication?Python performs addition and multiplication from left to right in expressions, following the order of operations.Can I use the order of operations in mathematical formulas and equations in Python?Yes, mathematical formulas and equations in Python should adhere to the order of operations.What is the purpose of using triple-quoted strings in multiline comments?Triple-quoted strings can be used for multiline comments to achieve a similar effect, even though Python does not have a specific multiline comment syntax.Why should comments be used sparingly in Python code?Comments should focus on explaining why something is done, as the code itself should be clear and self-explanatory through good organization and naming conventions.In what situations can the order of operations prevent calculation errors?The order of operations prevents errors by ensuring that expressions are evaluated in a predictable and correct sequence, avoiding unintended results.How can I handle numeric calculations involving variables in Python?Numeric calculations involving variables should be done following the order of operations, taking into account the values of the variables.What are some best practices for using parentheses in Python expressions?Use parentheses to clarify the order of operations, especially in complex expressions, and to ensure that calculations are performed as intended.Can I use the order of operations in Python for financial calculations?Yes, the order of operations is applicable to financial calculations, ensuring accuracy in calculations involving interest rates, payments, etc.Is there a difference between the order of operations in Python 2 and Python 3?The order of operations is consistent between Python 2 and Python 3. However, it's essential to use the appropriate version of Python for your code, as Python 2 is no longer supported.In summary, the order of operations is fundamental in Python and other programming languages to ensure that mathematical expressions are evaluated correctly. Adhering to the order of operations is essential for writing accurate and predictable code when dealing with mathematical calculations and expressionsThe fundamental role of the order of operations in writing accurate and predictable Python code, particularly in mathematical and algorithmic contexts Finally, just like in mathematics, we must also be aware of the order that these operators are applied, especially if they are combined into a single expression. Thankfully, the same rules we learned in mathematics apply in programming as well. Specifically, operators in Python are applied in this order:Operations in parentheses are resolved first, moving from left to right.** is resolved second, moving from left to right*, /, // and % are resolved third, moving from left to right.+ and - are resolved fourth, moving from left to right.You might recall the "PEMDAS" acronym for remembering the order of operations in math, and thankfully it still applies here. Of course, this means that there are now 4 operators that all fit in the "multiplication and division" portion, so we have to carefully make sure they are all taken care of in the correct way.Also, as you've probably already learned in math, it is always best to add extra parentheses to any expression to make the intent very clear instead of relying on the order of operations. So, when in doubt, use extra parentheses wherever needed!Let's work through a quick example just to see the order of operations in practice. Here's a complex expression in Python that we can try to evaluate:x = 8 / 4 + 5 * (3 + 1) - 7 % 4Looking at our order of operations, the first step is to handle any expressions inside of parentheses. So, we'll first start with the expression (3 + 1) and evaluate it to 4.x = 8 / 4 + 5 * 4 - 7 % 4Then, we'll go left to right and perform any multiplication, division, and modulo operations. This means we'll evaluate 8 / 4, 5 * 4 and 7 % 4 and replace them with the resulting values:Notice that 8 / 4 was reduced to 2.0 instead of just 2. This is because the division operator is the one exception to the rule where an operator applied to two integers will result in an integer. The division operator always produces a floating-point value.Finally, we'll perform addition and subtraction from left to right. So, we'll evaluate 2.0 + 20 first, and then subtract 3 from the result of that operation. At the end, we'll have this statement:So, we were able to use our knowledge of the order of operations to evaluate that complex expression to a single value, $ 19.0 $, which will be stored in the variable x.Page 2Let's try some simple practice problems. These problems are not graded - they are just for you to practice before doing the real exercises in the lab itself. You can find the answers below each question by clicking the button below each question.2.1 Reading CodeWrite the output that is displayed to the user after running the following Python code:x = 13 z = "5" y = int(z) var = (x + y) % y * y - x var = var / 2 print(var)Pay special attention to data types! Make sure the answer is presented as the correct type. AnswerThe correct answer isFollowing order of operations, we do parentheses first:(x + y) % y * y - x (13 + 5) % 5 * 5 - 13 18 % 5 * 5 - 13Then we will do multiplication and modulo left to right. Recall that % is the modulo operation, so 18 % 5 is the remainder of 18 / 5, which is 3:18 % 5 * 5 - 13 3 * 5 - 13 15 - 13Finally, we do subtraction:The last line will compute 2 / 2, which is just 1.0. Recall that the division operation always produces a float, not an integer.2.2 Reading CodeWrite the output that is displayed to the user after running the following Python code:x = 9 y = 3.0 z = 5 ans = (x - z) % 2 // y + z ** (y - 1) print(ans)Pay special attention to data types! Make sure the answer is presented as the correct type. AnswerThe correct answer is:Following order of operations, we do parentheses first:(x - z) % 2 // y + z ** (y - 1) (9 - 5) % 2 // 3.0 + 5 ** (3.0 - 1) 4 % 2 // 3.0 + 5 ** 2.0Then, we'll perform all multiplication, division, modulo, and integer division from left to right:4 % 2 // 3.0 + 5 ** 2.0 0 // 3.0 + 5 ** 2.0 0.0 + 5 ** 2.0 0.0 + 25.0Finally, we'll perform addition to find the answer of 25.0.2.3 Writing CodeYou are teaching a class and would like to put your students into a number of groups. You know how many students are in the class and the number of groups to create, but you aren't sure how many students should be in each group.We'll assume that these values are stored in the students and groups variables, respectively.Write a Python program to compute the ideal group size for each group in the class. When divided, the groups in the class should have no fewer than size people, and no more than size+1 people, and there should be exactly groups groups total. For example, if there are $ 15 $ people and the desired number of groups is $ 4 $, then code should start with the following two variable assignments.students = 15 groups = 4 # more code goes hereYour code should produce the following output for these values:This is because the ideal group size for $ 4 $ groups out of $ 15 $ people is $ 3 $, and all groups should have either $ 3 $ or $ 4 $ members (in this case, one group of $ 3 $ and the rest groups of $ 4 $).Write the rest of this Python program. Try different values for the students and groups variables to make sure that your answers are correct! PEMDAS is an acronym in mathematics that describes the order in which operations should be performed when evaluating an expression. It stands for: P - Parentheses E - Exponents M - Multiplication D - Division A - Addition S - Subtraction Python strictly adheres to the PEMDAS order of operations, making it consistent and predictable when evaluating complex expressions. This follows standard mathematical conventions and allows Python to give the expected result. Here's a quick overview of how it works: Parentheses Expressions inside parentheses are evaluated first. For example: Python evaluates the expression inside the parentheses (3 + 4) first, which equals 7. It then multiplies the result by 2, giving 14. Exponents Exponents are evaluated next. For example: Here, 3 is raised to the 2nd power, giving 9. Multiplication and Division Multiplication and division are performed next, from left to right. For example: Python first multiplies 2 * 3 which is 6. It then divides 6 by 6, giving 1. Division and multiplication have equal precedence. Addition and Subtraction Finally, addition and subtraction are performed. For example: Python first adds 2 + 3 which is 5. It then subtracts 4, giving a result of 1. Like multiplication and division, addition and subtraction have equal precedence. PEMDAS is an acronym for a mathematical rule that defines the order to solve arithmetic problems effectively.The PEMDAS rule tells us the sequence in which the expression with multiple operations is solved. The order is PEMDAS: Parentheses, Exponents, Multiplication, Division (from left to right), Addition, and Subtraction (from left to right).PEMDAS Full Form PEMDAS is a collection of rules in mathematics that help us solve problems in the correct sequence. It's similar to a recipe for solving arithmetic problems.P stands for Parentheses, which are essentially containers that house numbers and operations. First, we deal with what is inside them.E stands for Exponents, which are integers that inform us how many times we can multiply a given number. M stands for Multiplication D stands for Division, AS stands for Addition and Subtraction, performed from left to right direction.Using the PEMDAS rule, one will always get the right answer.PEMDAS Rule- Order of OperationsPEMDAS is a set of recommendations that define the order of operations for solving mathematical equations properly.It begins with brackets, which prioritize the processes encompassed inside them. Following that, exponents and powers are discussed. Then, multiplication and division are done from left to right. Finally, addition and subtraction are done the same way. P{{ ( ) }}ParenthesesEx2ExponentsM or Dx or ÷Multiplication or DivisionA or S+ or -Addition or SubtractionSolving Problems with the PEMDAS RuleWhen you have a math problem with multiple operations such as addition, subtraction, multiplication, and division, the PEMDAS rule helps you to perform the operations in a correct order, it instructs you to begin with parentheses, then exponents, then multiplication and division (whichever comes first from left to right), and finally addition and subtraction. Note: PEMDAS or PEDMAS is same as BODMAS both rules correctly define the order of operation to solve a mathematical operation. The use of PEMDAS is explained using the example below:Suppose in a class, two students A and B are asked to solve 11 - 2 × 2Student A solved it as 11 - 2 × 2 (Incorrect way)= 9 × 2= 18Student B solved it as: 11 - 2 × 2 (Correct way)= 11 - 4= 7Both students solved correctly according to their understanding but only the second one is correct as it is done in the correct order. We will learn the correct way of solving such expressions is explained using PEMDAS in this article.PEMDAS Rule Solved ExampleLet us explain PEMDAS with an example.5 + 2[10 - 3(4 - 2)] ÷ 2We will begin by working from the inside of the brackets. We will begin by solving the innermost bracket and then proceed to the outermost bracket.Step 1: Solve for 4 - 2, which equals 2. The equation becomes 5 + 2[10 - 3(2)] ÷ 2.Step 2: Compute 3(2), which equals 6. The equation becomes 5 + 2[10 - 6] ÷ 2.Step 3: Now, between the parentheses, answer 10 - 6 = 4. Our equation is now 5 + 2[4] ÷2.Step 4: Then, address what's between the brackets 2[4] = 8. Our expression now looks like 5 + 8 ÷ 2.Step 5: Following PEMDAS, we divide first 8 ÷ 2 = 4. The equation becomes 5 + 4.Step 6: Finally, add 5 and 4, and we have our final answer: 9.By using PEMDAS, you may effortlessly solve hard arithmetic problems and thrive in your math career. Applications of PEMDAS RuleUnderstanding the sequence of operations in mathematics is essential for many fields and daily situations. Let's look at some significant uses of PEMDAS in various sectors and how they help to ensure precision and dependability. Engineering: When designing structures such as bridges, engineers utilize PEMDAS to properly calculate loads and stresses, assuring the structure's safety and stability.Computer Science: PEMDAS helps verify that algorithms provide proper results while coding. For example, in a program that calculates distances between points on a map, the sequence of operations guarantees that the results are accurate.Economics and Finance: Financial analysts utilize PEMDAS to calculate financial indicators like net present value and internal rate of return, which are critical for making sound investment decisions.Pharmacists: Pharmacists use PEMDAS to precisely determine medicine doses. Using the exact order of procedures is crucial to avoid giving patients wrong dosages of medication, which might be dangerous.Architecture and Construction: When planning structures, architects utilize PEMDAS to compute dimensions, angles, and structural integrity. Following the proper sequence of procedures ensures that structures are both visually beautiful and structurally stable.PEMDAS Rule vs BODMAS RuleThe following table compares PEMDAS with BODMASPEMDASBODMASUsed for the systematic simplification of mathematical operations such as division, multiplication, addition, and subtraction. It is also used to simplify arithmetic operations like division, multiplication, addition, and subtraction in an orderly fashion.P = ParenthesisE = ExponentsM = MultiplicationD = DivisionA = AdditionS = SubtractionB = BracketsO = OrdersD = DivisionM = MultiplicationA = AdditionS = SubtractionPeople Also Read:Solve Simplifications Using BODMAS RulesBODMAS Rule in MathPEMDAS Rule Solved ExamplesExample 1: Simplify the equation with the PEMDAS rule: [18 + {12 – (4 x 8)}]Solution:=> [18 + {12 – (4 x 8)}] = [18 + {12 – 32}]=> [18 + {-20}]=> [18 - 20]=> -2Example 2: Calculate: 3 × 3 - 3 ÷ 3 + 3.Solution:=> 3 × 3 - 3 ÷ 3 + 3=> 9 - 3 ÷ 3 + 3=> 9 - 1 + 3=> 9 + 2=> 11Example 3: Solve: 5 + 8 × (3 + 8) ÷ 4 – 6 using PEMDAS.Solution:Step 1 (Parentheses): 5 + 8 × (3 + 8) ÷ 4 – 6 = 5 + 8 × 11 ÷ 4 – 6Step 2 (Multiplication): 5 + 8 × 11 ÷ 4 – 6 = 5 + 88 ÷ 4 – 6Step 3 (Division): 5 + 88 ÷ 4 – 6 = 5 + 22 – 6Step 4 (Addition): 5 + 22 – 6 = 27 - 6Step 5 (Subtraction): 27 - 6 = 21