

Let's evaluate the given expression according to Python's operator precedence:

$$45/5 + 3 \times 4 - 7 \times (44\%5)$$

Operator Precedence in Python:

1. **Parentheses**: Evaluate expressions inside `()`
2. **Modulo (%) and Division (/)**: Left to right
3. **Multiplication (*)**: Left to right
4. **Addition (+) and Subtraction (-)**: Left to right

Step-by-step evaluation:

1. Modulo (%) first:

$$44\%5 = 4$$

Now the expression becomes:

$$45/5 + 3 \times 4 - 7 \times 4$$

2. Division (/):

$$45/5 = 9.0$$

Now:

$$9.0 + 3 \times 4 - 7 \times 4$$

3. Multiplication (*):

$$3 \times 4 = 12$$

$$7 \times 4 = 28$$

Now:

$$9.0 + 12 - 28$$

4. Addition and Subtraction (+, -):

$$9.0 + 12 = 21.0$$

$$21.0 - 28 = -7.0$$

Final Result:

-7.0

Since Python follows floating-point division by default (`/` returns a float), the result is `-7.0`.

75 - 5 ** 2 + 3 * 2 ** 4 // 2

Let's evaluate the given expression step by step according to Python's precedence rules:

$$75 - 5 * 2 + 3 * 2 * 4 // 2$$

Operator Precedence in Python:

1. **Exponentiation (**)** (right to left)
2. **Multiplication (*), Integer Division (//)** (left to right)
3. **Addition (+), Subtraction (-)** (left to right)

Step-by-step evaluation:

1. Exponentiation (**) first:

- - - - -

$$5 * 2 = 25$$

$$2 * 4 = 16$$

Now the expression becomes:

$$75 - 25 + 3 * 16 // 2$$

2. Multiplication (*):

$$3 * 16 = 48$$

Now:

$$75 - 25 + 48 // 2$$

3. Integer Division (//):

$$48 // 2 = 24$$

Now:

$$75 - 25 + 24$$

4. Addition and Subtraction (+, -):

$$75 - 25 = 50$$

$$50 + 24 = 74$$

Final Result:

74

$$8 + (15 / 3) * 4 - 6^2 + (10 - 3) / 7$$

Let's evaluate the given expression step by step according to Python's precedence rules:

$$8 + (15/3) * 4 - 6^2 + (10 - 3)/7$$

Operator Precedence in Python:

1. Parentheses (()) first
2. Exponentiation (**)(right to left)
3. Multiplication (*), Division (/), Integer Division (//(left to right)
4. Addition (+), Subtraction (-)(left to right)

Step-by-step evaluation:

1. Parentheses (()) first:

$$(15/3) = 5.0$$

$$(10 - 3) = 7$$

Now the expression becomes:

$$8 + 5.0 * 4 - 6^2 + 7/7$$

2. Exponentiation (^ is NOT used for power in Python; it should be ** instead):

If ^ is used, it represents bitwise XOR in Python, not exponentiation.

Assuming you meant exponentiation:

$$6 * 2 = 36$$

Now:

$$8 + 5.0 * 4 - 36 + 7/7$$

3. Multiplication (*) and Division (/):

$$5.0 * 4 = 20.0$$

$$7/7 = 1.0$$

Now:

$$8 + 20.0 - 36 + 1.0$$

4. Addition and Subtraction (+, -):

$$8 + 20.0 = 28.0$$

$$28.0 - 36 = -8.0$$

$$-8.0 + 1.0 = -7.0$$

Final Result:

$$-7.0$$

If you actually meant bitwise XOR (`6 ^ 2`), the result would be different. Let me know if you want that evaluated!

Assume `^` bitwise xor

Let's evaluate the given expression assuming `^` is the **bitwise XOR** operator in Python:

$$8 + (15/3) * 4 - 6^2 + (10 - 3)/7$$

Operator Precedence in Python:

1. Parentheses (`()`) first
2. Division (`/`), Multiplication (`*`) (left to right)
3. Bitwise XOR (`^`) (after multiplication/division, before addition/subtraction)
4. Addition (`+`), Subtraction (`-`) (left to right)

Step-by-step evaluation:

1. Parentheses (`()`) first:

$$(15/3) = 5.0$$

$$(10 - 3) = 7$$

Now the expression becomes:

$$8 + 5.0 * 4 - 6^2 + 7/7$$

2. Multiplication (`*`) and Division (`/`):

$$5.0 * 4 = 20.0$$

$$7/7 = 1.0$$

Now:

$$8 + 20.0 - 6^2 + 1.0$$

3. Bitwise XOR (`^`):

- In Python, `6 ^ 2` performs bitwise XOR between `6` (binary: `110`) and `2` (binary: `010`):

```
sql
      110  (6 in binary)
      010  (2 in binary)
      ----
      100  (4 in decimal)
```

- So, `6 ^ 2 = 4`.

Now:

$$8 + 20.0 - 4 + 1.0$$

4. Addition and Subtraction (+, -):

$$8 + 20.0 = 28.0$$

$$28.0 - 4 = 24.0$$

$$24.0 + 1.0 = 25.0$$

Final Result:

25.0

t = 5 + 3 * 2 ** 2 // 4 - 1.

Let's evaluate the given expression step by step:

$$t = 5 + 3 * 2 * 2 // 4 - 1$$

Operator Precedence in Python:

1. Exponentiation (**) (right to left)
2. Multiplication (*), Integer Division (//) (left to right)
3. Addition (+), Subtraction (-) (left to right)

Step-by-step evaluation:

1. Exponentiation (2 ** 2):

$$2 * 2 = 4$$

Now the expression becomes:

$$t = 5 + 3 * 4 // 4 - 1$$

2. Multiplication (*) and Integer Division (//):

$$3 * 4 = 12$$

$$12 // 4 = 3$$

Now:

$$t = 5 + 3 - 1$$

3. Addition and Subtraction (+, -):

$$5 + 3 = 8$$

$$8 - 1 = 7$$

Final Result:

7

< 2/2 >

12//3

The expression:

$$12 // 3$$

Explanation:

- // is the integer division operator in Python.
- It divides the number and truncates the decimal part, returning an integer.

Calculation:

$$12 \div 3 = 4$$

Since the result is already an integer, integer division gives:

4

What floor division does give example

What is Floor Division (//) in Python?

Floor division (//) divides two numbers and rounds down the result to the nearest integer. It always returns an integer by truncating the decimal part.

Example 1: Positive Numbers

python

Copy Edit

```
print(10 // 3) # Output: 3
```

Explanation:

- $10 / 3 = 3.3333$
- Floor division rounds it down to 3.

Example 2: Negative Numbers

python

Copy Edit

```
print(-10 // 3) # Output: -4
```

Explanation:

- $-10 / 3 = -3.3333$
- Floor division rounds down to the nearest lower integer (-4 instead of -3).

Example 3: Floating-Point Numbers

python

Copy Edit

```
print(7.5 // 2) # Output: 3.0
```

Explanation:

- $7.5 / 2 = 3.75$
- Floor division rounds down to 3.0 (still a float if any operand is float).

Key Points

- ✓ Always rounds down (toward negative infinity).
- ✓ Works with integers and floats.

- Result is integer if both operands are integers, otherwise a float.

Try with bigger number

Sure! Let's try floor division (`//`) with bigger numbers.

Example 1: Large Positive Numbers

python

Copy Edit

```
print(1000000 // 123) # Output: 8130
```

Explanation:

- `1000000 / 123 ≈ 8130.0813`
- Floor division rounds down to `8130`.

Example 2: Large Negative Numbers

python

Copy Edit

```
print(-1000000 // 123) # Output: -8131
```

Explanation:

- `-1000000 / 123 ≈ -8130.0813`
- Floor division rounds down to `-8131` (towards negative infinity).

Example 3: Large Floating-Point Numbers

python

Copy Edit

```
print(9876543210.0 // 54321) # Output: 181875.0
```

Explanation:

- `9876543210 / 54321 ≈ 181875.225`
- Floor division rounds down to `181875.0`.

Would you like to test with even larger numbers?

153//10

The expression:

153//10

Explanation:

- `//` is floor division, which divides and rounds down to the nearest integer.
- `153 / 10 = 15.3`
- Floor division removes the decimal part, so the result is:

Gross Salary

Gross salary is the total earnings an employee receives before any deductions like taxes, provident fund (PF), and other withholdings.

Formula:

$$\text{Gross Salary} = \text{Basic Salary} + \text{Allowances}$$

Where allowances may include:

- ✓ House Rent Allowance (HRA)
- ✓ Dearness Allowance (DA)
- ✓ Travel Allowance (TA)
- ✓ Medical Allowance
- ✓ Bonus, Overtime, etc.

Example:

If an employee's Basic Salary is ₹30,000 and receives:

- HRA = ₹10,000
- DA = ₹5,000
- Other Allowances = ₹2,000

Then,

$$\text{Gross Salary} = 30,000 + 10,000 + 5,000 + 2,000 = \text{₹47,000}$$

Net Salary vs. Gross Salary

- Gross Salary = Salary before deductions.
- Net Salary (Take-Home Salary) = Salary after all deductions (like tax, PF, etc.).

Let me know if you need further clarification!

24. Write an algorithm for grade calculation of a student. Grade of a student will be assigned as per below table

Grade	Marks
> 90	S
Between 81 and 90	A
Between 71 and 80	B
Between 61 and 70	C
Between 51 and 60	D
Between 41 and 50	E
< 41	Fail

Here is an algorithm for grade calculation based on the provided table:

Algorithm: Grade Calculation

1. Start
2. Input the marks of the student (marks).
3. Check the range of marks and assign a grade:
 - If marks > 90, assign Grade S.
 - Else if marks is between 81 and 90, assign Grade A.
 - Else if marks is between 71 and 80, assign Grade B.
 - Else if marks is between 61 and 70, assign Grade C.
 - Else if marks is between 51 and 60, assign Grade D.
 - Else if marks is between 41 and 50, assign Grade E.

- Else if `marks < 41`, assign Fail.
4. Display the assigned grade.
 5. End
-

?

Python Code Implementation

```
python

def calculate_grade(marks):
    if marks > 90:
        return "S"
    elif 81 <= marks <= 90:
        return "A"
    elif 71 <= marks <= 80:
        return "B"
    elif 61 <= marks <= 70:
        return "C"
    elif 51 <= marks <= 60:
        return "D"
    elif 41 <= marks <= 50:
        return "E"
    else:
        return "Fail"

# Example usage:
marks = int(input("Enter student marks: "))
print("Grade:", calculate_grade(marks))
```

Copy Edit

?

Would you like any modifications or explanations?

-
1. Write a python program to print the following pattern.

```
1
121
12321
1234321
123454321
```

2. Write an iterative program to find the factorial of a number.
3. Describe how character to number conversion is performed in Python. Write a Python program that takes a string of digits as input and returns the corresponding integer value.
4. Write a Python program to print the following picture.

6 / 8



Solve q:1 and 4

?

Solution for Question 1: Number Pattern

To print the given pattern:

```
1
121
12321
1234321
123454321
```

Copy Edit

?

Python Code:

```
python

def number_pattern(n):
    for i in range(1, n+1):
```

Copy Edit

?

```

# Generate the first half (ascending numbers)
for j in range(1, i+1):
    print(j, end="")
# Generate the second half (descending numbers)
for j in range(i-1, 0, -1):
    print(j, end="")
print() # Move to next Line

# Call function with required number of rows
number_pattern(5)

```

Solution for Question 4: Diamond Star Pattern

To print the diamond shape:

markdown

Copy Edit

```

*
 ***
 *****
 ******
 *****
 ****
 ***
 *

```

Python Code:

python

Copy Edit

```

def diamond_pattern(rows):
    # Upper part of diamond
    for i in range(1, rows + 1, 2):
        print(" " * ((rows - i) // 2) + "*" * i)

    # Lower part of diamond
    for i in range(rows - 2, 0, -2):
        print(" " * ((rows - i) // 2) + "*" * i)

# Call function with required number of rows
diamond_pattern(9)

```

Would you like modifications or explanations?

Q5. Write a python program to make a calculator with four functions. two non-zero input from user and calculate addition, subtraction, multiplication, and division. The program should incorporate with four different functions and call them by passing arguments.

Python Program: Simple Calculator with Functions

This program will take two non-zero inputs from the user and perform **addition, subtraction, multiplication, and division** using separate functions.

Python Code:

python

Copy Edit

```

# Function for addition
def add(a, b):

```

```

        return a + b

    # Function for subtraction
    def subtract(a, b):
        return a - b

    # Function for multiplication
    def multiply(a, b):
        return a * b

    # Function for division
    def divide(a, b):
        if b != 0: # Ensure denominator is not zero
            return a / b
        else:
            return "Error! Division by zero is not allowed."

    # Taking user input
    num1 = float(input("Enter first non-zero number: "))
    num2 = float(input("Enter second non-zero number: "))

    # Ensuring inputs are non-zero
    if num1 == 0 or num2 == 0:
        print("Error! Inputs must be non-zero.")
    else:
        # Performing calculations
        print(f"Addition: {add(num1, num2)}")
        print(f"Subtraction: {subtract(num1, num2)}")
        print(f"Multiplication: {multiply(num1, num2)}")
        print(f"Division: {divide(num1, num2)}")

```

Example Output:

yaml

Copy Edit

```

Enter first non-zero number: 10
Enter second non-zero number: 5
Addition: 15.0
Subtraction: 5.0
Multiplication: 50.0
Division: 2.0

```

Would you like any modifications?

Create a list with four elements in python. How you can access the list items by referring to the index number? How you can specify a range of indexes by specifying where to start and where to end the range? Show with an example.

Creating a List and Accessing Elements by Index in Python

Step 1: Creating a List with Four Elements

python

Copy Edit

```
my_list = ["Apple", "Banana", "Cherry", "Mango"]
```

Step 2: Accessing List Items by Index

In Python, list indices start from 0.

- First element: `my_list[0]` → "Apple"

- Second element: `my_list[1]` → "Banana"
- Third element: `my_list[2]` → "Cherry"
- Fourth element: `my_list[3]` → "Mango"

Example:

```
python
```

Copy Edit

```
print(my_list[0]) # Output: Apple
print(my_list[2]) # Output: Cherry
```

Step 3: Accessing a Range of Indexes

You can specify a range using list slicing:

- `list[start:end]` → Includes start index but excludes end index

Example:

```
python
```

Copy Edit

```
print(my_list[1:3]) # Output: ['Banana', 'Cherry']
```

Explanation:

- Starts from index `1` → "Banana"
- Stops before index `3` → "Cherry" (does not include "Mango")

Step 4: Omitting Start or End in Slicing

- Omit start index (starts from beginning): `my_list[:3]`
- Omit end index (till last element): `my_list[2:]`

Example:

```
python
```

Copy Edit

```
print(my_list[:3]) # Output: ['Apple', 'Banana', 'Cherry']
print(my_list[2:]) # Output: ['Cherry', 'Mango']
```

Would you like a specific example or explanation on list operations?

```
n = int(input("Enter the number:"))
factors_of_n = 0
for i in range(1,n+1):
    if n%i == 0:
        factors_of_n = factors_of_n+1
if factors_of_n == 2:
    print("The Number Is A prime Number")
else :
    print("The Number is Not a prime Number")
Using this approach print all the prime numbers from 1 to 100
```

To print all the prime numbers from 1 to 100 using your approach, we can modify the code to check

each number in the range 1 to 100 and determine if it is prime.

Modified Code to Print Prime Numbers from 1 to 100

```
python

for n in range(1, 101): # Loop from 1 to 100
    factors_of_n = 0 # Reset factor count for each number
    for i in range(1, n + 1): # Check divisibility from 1 to n
        if n % i == 0:
            factors_of_n += 1 # Count the factors

    if factors_of_n == 2: # A prime number has only 2 factors (1 and itself)
        print(n, end=" ") # Print prime numbers in a single line
```

Copy Edit

Explanation:

1. Loop through numbers 1 to 100.
2. Count the number of divisors for each number.
3. If a number has exactly two factors (1 and itself), it's a prime number.
4. Print the prime numbers.

Output:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Copy Edit

Would you like an optimized approach using the square root method for better performance?

```
n = int(input("Enter The Number: "))

# Extract individual digits
d1 = n % 10
d2 = (n // 10) % 10
d3 = (n // 100) % 10

# Calculate the sum of cubes of digits
s = (d1 ** 3) + (d2 ** 3) + (d3 ** 3)

# Check if the sum is equal to the original number
if s == n:
    print("The Number Is Armstrong")
else:
    print("The Number Is Not Armstrong")

#Method 3 : using while Loop

n = int(input("Enter The Number: "))
i = n # Store original number in i
s = 0 # Initialize sum

while n > 0:
    d = n % 10 # Get last digit
    s = s + d ** 3 # Cube the digit and add to sum
    n = n // 10 # Remove last digit and the loop continues with this
    number

# Check if the sum is equal to the original number
if s == i:
    print("The Number Is Armstrong")
else:
    print("The Number Is Not Armstrong")
Print all armstrong number between 101 to 999 using this
```

To print all **Armstrong numbers** between **101** and **999**, we can modify your approach to iterate through each number in this range and check if it is an Armstrong number.

Python Code to Print All Armstrong Numbers from 101 to 999

```
python Copy Edit

for n in range(101, 1000): # Loop from 101 to 999
    i = n # Store original number
    s = 0 # Initialize sum

    while n > 0:
        d = n % 10 # Get Last digit
        s += d ** 3 # Cube the digit and add to sum
        n //= 10 # Remove Last digit

    # Check if the sum is equal to the original number
    if s == i:
        print(i, end=" ") # Print Armstrong number
```

Explanation:

1. Loop through numbers **101** to **999**.
2. Extract individual digits using the `while` loop.
3. Compute the **sum of the cubes** of the digits.
4. Check if the sum is **equal to the original number**.
5. If true, print the Armstrong number.

Output:

```
Copy Edit

153 370 371 407
```

Would you like a more optimized approach using **list comprehension** or **functions**?

```
n = int(input("Enter The Number: "))
i = n
s = 0
while(n>0):
    d=n%10
    s=s*10+d
    n=n//10
if(i == s):
    print("palindrome")
else:
    print("Not Palindrome")
Print all palindrome number from 100 to 999
```

To print all **palindrome numbers** between **100** and **999**, we can modify your approach to iterate through each number in this range and check if it is a palindrome.

Python Code to Print All Palindrome Numbers from 100 to 999

```
python Copy Edit

for n in range(100, 1000): # Loop from 100 to 999
    i = n # Store original number
    s = 0 # Initialize reversed number
```

```

while n > 0:
    d = n % 10 # Extract last digit
    s = s * 10 + d # Build the reversed number
    n //= 10 # Remove last digit

# Check if the reversed number is equal to the original
if i == s:
    print(i, end=" ") # Print palindrome number

```

?

Explanation:

1. Iterate through numbers 100 to 999.
2. Reverse each number using the `while` loop.
3. Compare the reversed number with the original number.
4. If they are `equal`, print the number.

Output:

```
101 111 121 131 141 151 161 171 181 191 202 212 222 232 242 252 262 272 282 292 303 313 323 333
```

Copy Edit

?

Would you like an optimized approach using string manipulation?

According to precedence multiplication or division is done in python 1st

Yes, according to operator precedence in Python, multiplication (`*`) and division (`/`, `//`, `%`) have the same priority and are performed before addition (`+`) and subtraction (`-`).

Operator Precedence Order (from highest to lowest):

1. Parentheses: `()` (Highest precedence, evaluated first)
2. Exponentiation: `**`
3. Unary operators: `+x`, `-x`
4. Multiplication (`*`), Division (`/`), Floor Division (`//`), Modulus (`%`) → evaluated left to right
5. Addition (`+`), Subtraction (`-`)
6. Comparison operators: `==`, `!=`, `>`, `<`, `>=`, `<=`
7. Logical operators: `not` `>` `and` `>` `or` (Lowest precedence)

?

Example:

```
python

result = 10 + 5 * 2 # Multiplication (*) is performed first
print(result) # Output: 20
```

Copy Edit

?

Explanation:

- `5 * 2` is calculated first (`10`), then `10 + 10 = 20`.

Example with Division and Multiplication:

```
python

result = 20 / 4 * 2
print(result) # Output: 10.0
```

Copy Edit

?

Explanation:

- Division (`20 / 4 = 5.0`) is performed first.

- Then, multiplication ($5.0 * 2 = 10.0$).

Would you like a deeper explanation of any specific case?

