مكتب

Quick Reference Notes

Week 5: Expressions and Operators

7th Grade Computer Science

1 Introduction

1.1 This Week's Big Question

How do computers "think" mathematically and make logical decisions? This week, we'll discover how Python evaluates expressions, follows mathematical rules, and understands True/False logic - the foundation for making smart programs!

Prerequisites

Before starting this week, you should be comfortable with:

- Creating and using variables
- Understanding different data types (`int`, `float`, `str`, `bool`)
- Knowing that Python reads code top to bottom

1.2 What You Already Know

You've created variables to store numbers and text. You know that `age = 13` puts number 13 into the variable `age`. Now we'll learn how to make Python do calculations and comparisons with those values!

1.3 What You'll Be Able to Do

By the end of this week, you'll:

- Use Python as a powerful calculator with expressions
- Understand why `2 + 3 * 4` equals 14, not 20
- Create `True`/`False` expressions to compare values
- Trace through complex expressions step-by-step

2 VIDEO 1: Expressions and Operators - Python's Math Power

2.1 What Are Expressions?

An expression is a combination of values, variables, and operators that Python can evaluate (i.e solve) to produce a single value. Think of it like a math problem that Python solves for you!

2.2 Expressions vs Statements: Two Types of Python Code

Python code comes in two fundamental forms:

Rule

1. **Expressions** - Code that evaluates to a value *(can be computed)*
2. **Statements** - Code that performs an action *(does something)*

Expressions evaluate to values:

```
5 + 3                # Evaluates to 8
score * 2            # Evaluates based on score's value
len("Hello")         # Evaluates to 5
10 > 5               # Evaluates to True
```

Statements perform actions:

```
age = 13             # Assignment statement
print("Hello")       # Print statement
```

2.3 The Assignment Statement

Assignment is a statement, not an expression. It doesn't evaluate to a value - it performs the action of storing a value in a variable.

Rule

The Assignment Rule:

1. Evaluate the expression on the right-hand side (RHS)
2. Bind the resulting value to the variable name on the left-hand side (LHS)

Syntax: `variable_name = expression`

Assignment statement structure:

```
score = 85 + 10
^^^^^   ^^^^^^^
  |         |
  |         +-- Expression (RHS)
  |
  +----------- Variable (LHS)
```

Step 1: Evaluate RHS (85 + 10 = 95)
Step 2: Store value 95 in the variable `'score'` i.e LHS

2.4 Basic Arithmetic Operators

Python knows all the basic math operations:

```
# Addition
total = 10 + 5       # 15
print(total)

# Subtraction
```

```
change = 100 - 75    # 25

# Multiplication
area = 7 * 8         # 56

# Division (always gives a float!)
result = 15 / 3      # 5.0 (not 5)

# Integer Division (drops the decimal)
pizzas = 17 // 5     # 3 (how many whole pizzas per person?)

# Modulo (remainder)
leftover = 17 % 5    # 2 (pieces left over)

# Exponentiation (power)
square = 4 ** 2      # 16 (4 squared)
```

Rule

Arithmetic Operators:

- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division (always returns float)
- `//` Integer division (returns quotiend, whole number only)
- `%` Modulo (returns remainder)
- `**` Exponentiation (power)

2.5 Working with Different Types

Operators work differently with different data types:
When used with numbers operators perform math operations

```
price = 50 + 30          # 80
```

With strings operators act as concatenators (join text together) or repeaters (repeat text)

```
greeting = "Hello" + " " + "Pakistan"    # "Hello Pakistan"
repeated = "Ha" * 3                      # "HaHaHa"
```

With mixed types → Error!

```
# result = "Age: " + 25  # TypeError!
result = "Age: " + str(25)  # "Age: 25" (must convert!)
```

Common Error

You can't mix strings and numbers in math!

These will crash:

```
"5" + 3          # Can't add string + number
"Price" * 2.5    # Can't multiply string by float
```

These work:

```
5 + 3            # Both numbers
"Ha" * 3         # String repetition: "HaHaHa"
```

Tip

Division (/) always returns a float, even for whole numbers:

- `10 / 2` gives `5.0` not `5`
- Use `//` for integer division: `10 // 2` gives `5`, which is the quotient when dividing 10 by 2

Quick Check

What will each expression evaluate to?

1. `15 + 10`
2. `20 / 4`
3. `13 % 5`
4. `"CS" * 2`

2.6 The len() Function - Counting Characters

Python has a built-in function called `len()` that counts characters in a string. It returns a number you can use in expressions!

Get the length of strings:

```
name = "Ahmed"
name_length = len(name)                # 5
print(name_length)
```

Use `len()` directly in expressions:

```
message = "Hello Pakistan!"
half_length = len(message) / 2         # 15 / 2 = 7.5
```

Checking password requirements:

```
password = "mySecret123"
is_valid = len(password) >= 8          # True (11 >= 8)
```

Empty string has length 0L

```
empty = ""
print(len(empty))                      # 0
```

Common Error

`len()` only works with strings, not numbers!

These will crash:

```
age = 13
len(age)            # TypeError: object of type 'int' has no len()
price = 99.50
len(price)          # TypeError: object of type 'float' has no len()
```

To count digits in a number, convert to string first:

```
age = 13
age_digits = len(str(age))      # 2 (converts 13 to "13")
phone = 923001234567
phone_length = len(str(phone))  # 12 digits
```

Tip

`len()` counts all characters including spaces and punctuation:

- `len("Hi!")` returns `3`
- `len("A B")` returns `3` *(space counts!)*

3 VIDEO 2: Order of Operations - Python Follows Math Rules!

3.1 Why Order Matters

Look at this expression: `2 + 3 * 4`

Suppose, for the time being, if we go left to right: `2 + 3 = 5`, and then `(5) * 4 = 20` !!

But Python gives us 14! Why? Because Python follows the same order of operations you learned in math class.

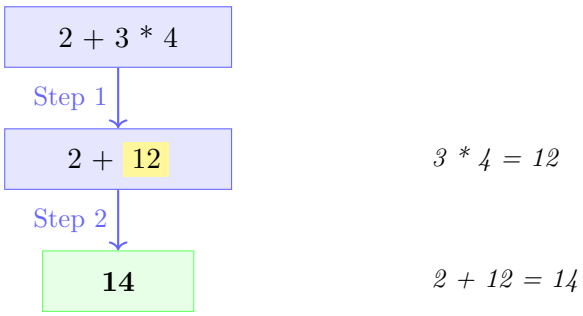
3.2 BODMAS/PEMDAS in Python

Python follows this order (same as your math class!):

1. **B**rackets/**P**arentheses: `( )`
2. **O**rders/**E**xponents: `**`
3. **D**ivision and **M**ultiplication: `/ // * %` (left to right)
4. **A**ddition and **S**ubtraction: `+ -` (left to right)

3.3 Step-by-Step Evaluation

Let's trace through `2 + 3 * 4`:



3.4 Using Parentheses to Control Order

Want addition first? Use parentheses.

Without parentheses - multiplication first:

```
result1 = 2 + 3 * 4              # 2 + 12 = 14
```

With parentheses - addition first:

```
result2 = (2 + 3) * 4            # 5 * 4 = 20
```

Complex example:

```
answer = 100 - (20 + 10) * 2     # 100 - 30 * 2 = 100 - 60 = 40
```

Rule

Order of Operations Quick Reference:

1. Do what's in `( )` first
2. Then `**` (powers)
3. Then `* / // %` (left to right)
4. Finally `+ -` (left to right)

When in doubt, use parentheses to make it clear!

3.5 Real-World Example

```
# Calculate total bill at a restaurant
meal_cost = 250
drinks = 50
tip_percent = 0.15

# Wrong way (order matters!)
wrong_total = meal_cost + drinks * (1 + tip_percent)
# This does: 250 + 50 * 1.15 = 250 + 57.5 = 307.5

# Right way (use parentheses)
total = (meal_cost + drinks) * (1 + tip_percent)
# This does: 300 * 1.15 = 345
print(f"Total bill: {total} rupees")
```

Common Error

Common Order of Operations Mistakes:

```
# Averaging two numbers - WRONG
avg = 10 + 20 / 2     # This gives 10 + 10 = 20, not 15!

# CORRECT - use parentheses
avg = (10 + 20) / 2   # This gives 30 / 2 = 15
```

Quick Check

Evaluate these expressions (show your work!):

1. `10 + 2 * 5`
2. `(10 + 2) * 5`
3. `100 / 10 * 2`
4. `2 ** 3 + 1`

4 VIDEO 3: Boolean Expressions - True or False?

4.1 Understanding Boolean Logic

Boolean expressions evaluate to either `True` or `False`, just these two values! They're how Python makes decisions and comparisons - essential for creating smart programs!

4.2 Comparison Operators

Python can compare values:

```
age = 13
height = 5.5

# Equal to
print(age == 13)         # True

# Not equal to
print(age != 10)         # True

# Greater than
print(age > 15)          # False

# Less than
print(height < 6.0)      # True

# Greater than or equal to
print(age >= 13)         # True

# Less than or equal to
print(height <= 25.5)    # False
```

Rule

Comparison Operators:

- `==` Equal to *(two equals)*
- `!=` Not equal to
- `>` Greater than
- `<` Less than
- `>=` Greater than or equal to
- `<=` Less than or equal to

Remember: `=` assigns, `==` compares

4.3 Boolean Operators: and, or, not

The `and`, `or`, and `not` operators let you combine or modify boolean expressions:

```
age = 13
grade = 7

# AND - both must be True
can_join = (age >= 12) and (grade >= 7)    # True

# OR - at least one must be True
weekend = (day == "Saturday") or (day == "Sunday")

# NOT - flips True/False
school_day = not weekend
```

4.4 Truth Tables

Understanding `and`, `or`, and `not`:

AND Truth Table

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

OR Truth Table

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

NOT Table

A	not A
True	False
False	True

Tip

Memory tricks:

- `and` = Both must be True (strict parent rule)
- `or` = Either can be True (lenient parent rule)
- `not` = opposite day!

4.5 Complex Boolean Expressions

```python
# Checking multiple conditions
temperature = 28
is_raining = False

# Good weather for cricket?
good_weather = (temperature>20) and (temperature<35) and (not is_raining)
print(f"Play cricket? {good_weather}")  # True

# School admission check
age = 12
test_score = 85
eligible = (age >= 11) and (age <= 14) and (test_score >= 80)
```

Common Error

Common Boolean Mistakes:

Warning - do not chain comparisons like math

```
x = 5
result = 1 < x < 10
```

The above chaining actually works in Python but it is not recommended as it can lead to confusion. Instead, use logical operators. Test each condition separately:

Correct - use logical operators

```
if x > 1 and x < 10:  # Check each condition
```

But this doesn't mean what you think:

```
if x == 5 or 6:  # WRONG! Always True
# CORRECT:
if x == 5 or x == 6:  # Check each condition
```

Quick Check

What do these evaluate to? (Assume `x = 10`, `y = 5`)

1. `x > y`
2. `x == 10 and y == 5`
3. `x < 5 or y < 10`
4. `not (x == y)`

5 VIDEO 4: Summary

5.1 Python as a Calculator and Decision Maker

This week, we transformed Python into:

- A powerful calculator using arithmetic operators
- A rule-follower that respects order of operations
- A decision-maker using boolean expressions

- A logical thinker with and, or, and not
- A character counter with the len() function

5.2 Key Syntax Patterns

The operators we mastered:

```
# Arithmetic operators
result = 10 + 5 - 3 * 2 / 1  # Order: *, /, +, -
power = 2 ** 3                # 8
remainder = 17 % 5            # 2

# The len() function
text_length = len("Hello")    # 5
long_enough = len(password) >= 8  # Use in boolean

# Comparison operators
is_equal = (x == y)           # False if different
is_greater = (x > y)          # True if x larger

# Boolean operators
both_true = (x > 5) and (y > 0)     # Both conditions
either_true = (x > 15) or (y > 0)   # At least one
opposite = not (x == y)              # Flip the result
```

6 Quick Reference

Quick Reference

Week 5 Patterns:

- Arithmetic: `+ - * / // % **`
- Order: Parentheses → Powers → Multiply/Divide → Add/Subtract
- Compare: `== != > < >= <=`
- Logic: `and` (both), `or` (either), `not` (opposite)
- Count: `len(string)` returns number of characters in a string
- Division `/` always returns float, use `//` for integer division
- Use parentheses when order isn't clear

6.1 Reflection

Think about your learning this week:

1. Write an expression that calculates your age in months
2. Why does `10 / 2` give `5.0` instead of `5`?
3. Create a boolean expression that checks if a temperature is comfortable (between 20-30°C)
4. Write an expression using `len()` to check if a name has more than 10 characters

Next week, we'll learn to plan our programs before coding using pseudocode and flowcharts!