Numbers, Arithmetic Operators & Order of Operations

AiCE Software Engineering Foundations — Python Programming

Numbers are at the heart of every real program — prices, scores, distances, ages, percentages, exchange rates. This lesson covers Python's arithmetic operators, the order Python evaluates them in, and the shorthand operators that make updating a number's value quick and clean.

1. Integers and Floats — A Quick Recap

Python has two primary numeric data types: int, whole numbers with no decimal point, and float, numbers with a decimal point. The distinction matters because calculations can behave differently depending on which type is involved:

```
population = 220000000
gdp_growth = 3.7

print(type(population))              # <class 'int'>
print(type(gdp_growth))              # <class 'float'>

# Integer division produces a float in Python 3
print(type(10 / 3))                  # <class 'float'>

# Floor division stays an integer
print(type(10 // 3))                 # <class 'int'>
```

> **Regular Division Always Returns a Float**
> *10 / 2 gives 5.0, not 5 — regular division (/) always returns a float in Python 3, even when the result is a whole number. Floor division (//), covered below, always returns an integer by discarding the decimal portion entirely.*

2. Arithmetic Operators

Python supports the full standard set of arithmetic operations:

Operator	Operation	Example	Result
+	Addition	15 + 7	22
-	Subtraction	100 - 45	55
*	Multiplication	8 * 6	48
/	Division (float)	10 / 4	2.5
//	Floor Division (int)	10 // 4	2
%	Modulo (remainder)	10 % 3	1
**	Exponentiation	2 ** 8	256

Understanding Floor Division and Modulo Together

Floor division (//) tells you how many times a number fits completely into another. Modulo (%) tells you what's left over. Together, they're a genuinely useful pair:

```
total_laptops = 250
students_per_group = 8

full_groups = total_laptops // students_per_group
leftover = total_laptops % students_per_group

print(f"Complete groups : {full_groups}")
print(f"Leftover laptops: {leftover}")
```

This produces: Complete groups: 31, and Leftover laptops: 2 — 250 // 8 is 31 complete groups, and 250 % 8 is the 2 laptops left over.

One of the most common uses of modulo in all of programming is checking whether a number is even or odd:

```
student_id = 2026

if student_id % 2 == 0:
    print(f"Student ID {student_id} is even.")
else:
    print(f"Student ID {student_id} is odd.")
```

If dividing by 2 leaves no remainder, the number is even. This pattern will come up constantly once conditionals are covered in an upcoming lesson.

> **Modulo Has Many Uses**
> *Beyond even/odd checks and remainders, modulo is used for cycling through items repeatedly, checking divisibility, time calculations (minutes remaining after hours), and creating patterns. Whenever you think about remainders, think about modulo.*

3. Order of Operations — BODMAS in Python

When an expression contains multiple operators, Python evaluates them in the same order you learned in mathematics: BODMAS — Brackets, Orders/Exponents, Division, Multiplication, Addition, Subtraction.

```
# Without brackets — multiplication before addition
result1 = 5 + 3 * 2
print(f"5 + 3 * 2 = {result1}")     # 11, not 16

# With brackets — brackets evaluated first
result2 = (5 + 3) * 2
print(f"(5 + 3) * 2 = {result2}")   # 16

# Exponent before multiplication
result3 = 2 * 3 ** 2
print(f"2 * 3 ** 2 = {result3}")    # 18, not 36
```

A real formula puts this into practice — compound interest, $A = P \times (1 + r)^n$:

```
principal = 100000    # Naira invested
rate = 0.08           # 8% annual interest
years = 5
amount = principal * (1 + rate) ** years
print(f"Investment grows to: {amount:.2f} NGN")
```

The brackets ensure (1 + rate) is computed first, then ** applies the exponent, then * multiplies by the principal — exactly the order the formula requires.

> **When In Doubt, Add Brackets**
> *Brackets make your intent explicit and your code easier to read, with zero downside. Write code that a tired version of yourself can read at midnight — use brackets whenever arithmetic gets even slightly complex.*

4. Augmented Assignment Operators

Augmented assignment operators combine an arithmetic operation with assignment in one concise expression. Instead of writing x = x + 5, you write x += 5:

Operator	Equivalent To	Result if x = 10
x += 5	x = x + 5	15
x -= 3	x = x - 3	7
x *= 2	x = x * 2	20
x /= 4	x = x / 4	2.5
x //= 3	x = x // 3	3
x %= 3	x = x % 3	1
x **= 2	x = x ** 2	100

A running score tracker demonstrates the pattern naturally:

```
score = 0
score += 15    # Question 1
score += 20    # Question 2
```

```
score -= 5     # Late penalty
score *= 2     # Bonus
print(f"Final score: {score}")
```

You'll Use These Constantly

Augmented assignment operators show up everywhere once loops, counters, and accumulators enter the picture in later lessons. Getting comfortable with += and -= now pays off quickly.

Key Takeaways

Python's seven arithmetic operators — +, -, *, /, //, %, ** — cover everything from basic addition to exponentiation, with // and % working as a natural pair for quotient and remainder. Python evaluates expressions using BODMAS, and brackets are always worth adding for clarity. Augmented assignment operators like += and -= are shorthand you'll rely on constantly. The next lesson covers Python's built-in math functions and how to format numbers cleanly for professional-looking output.