MODULE 1: CORE PYTHON & DATA

WEEK: 1 LECTURE: 3

DATE: 19/08/2025

INSTRUCTOR: ORANGZAIB RAJPOOT

THE ENGINE OF LOGIC: PYTHON'S BASIC OPERATORS

Today's Agenda

- **The Calculators - Arithmetic & Assignment Operators**
  - Arithmetic Operators: Beyond Basic Math
  - Python's Division Nuances (/ vs. //)
  - Assignment Operators: The Efficient Shorthand
  - Interactive Exercises throughout
- **The Decision Makers - Comparison & Logical Operators**
  - Comparison Operators: Asking Questions
  - The Critical Difference: = vs. ==
  - Logical Operators: and, or, not (The English Advantage)
  - Chaining Comparisons for Readability
  - Interactive Exercises for each data type
- The Rules of Engagement - Precedence & Hands-On Lab
  - Operator Precedence: Python's Order of Operations
  - The Power of Parentheses ()
  - Hands-On Lab: The Ultimate Simple Calculator
  - Q&A and Wrap-up

OPERATORS

We've learned how to store data in variables. Today, we'll learn how to operate on that data. Operators are the symbols that perform computations, comparisons, and logical checks. For those with experience in other languages, the concepts will be familiar, but Python's clean syntax and specific behaviors—especially with division and logical operators—are key to writing Pythonic code.

THE CALCULATORS - ARITHMETIC & ASSIGNMENT ARITHMETIC OPERATORS: MORE THAN JUST + AND -

Operator	Name	Description	Example	Result
`+`	Addition	Adds two numbers	`10 + 5`	`15`
`-`	Subtraction	Subtracts the second number from the first	`10 - 5`	`5`
`*`	Multiplication	Multiplies two numbers	`10 * 5`	`50`
`**`	Exponent	Raises the first number to the power of the second	`10 ** 2`	`100`
`/`	**True Division**	Divides and **always returns a float**. This is a key Python feature.	`10 / 4`	`2.5`
`//`	**Floor Division**	Divides and rounds down to the nearest whole number (returns `int` or `float`).	`10 // 4`	`2`
`%`	Modulus	Returns the remainder of a division	`10 % 3`	`1`

IN-CLASS EXERCISE: AREA AND PERIMETER

- You are given length = 15.5 and width = 8.2.
  - Calculate the area of the rectangle (length * width).
  - Calculate the perimeter (2 * (length + width)).
  - Print both results in a formatted string.

ASSIGNMENT OPERATORS: EFFICIENT SHORTHAND

The basic assignment operator is =, which assigns the value on the right to the variable on the left. Compound assignment operators provide a concise way to perform an operation and an assignment in one step.

Operator	Example	Equivalent to...
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
**=	x **= 2	x = x ** 2
//=	x //= 3	x = x // 3
%=	x %= 3	x = x % 3

IN-CLASS EXERCISE: LEVELING UP A GAME CHARACTER

- A game character starts with health = 100 and mana = 50.
  - The character finds a health potion, which adds 25 health. Update the health using +=.
  - The character casts a spell that costs 30 mana. Update the mana using -=.
  - Print the final health and mana.

THE DECISION MAKERS - COMPARISON & LOGICAL

COMPARISON OPERATORS: ASKING QUESTIONS

These operators compare two values and **always return a boolean (True or False).**

Operator	Description	Example	Result
`==`	Equal to	`5 == 5`	`True`
`!=`	Not equal to	`5 != 6`	`True`
`>`	Greater than	`5 > 4`	`True`
`<`	Less than	`5 < 6`	`True`
`>=`	Greater than or equal to	`5 >= 5`	`True`
`<=`	Less than or equal to	`5 <= 4`	`False`

- **CRITICAL DISTINCTION: = vs. ==**
  This is one of the most common bugs for new programmers.
- = is the **assignment operator**. It *puts a value into a variable*. (my_variable = 10)
- == is the **comparison operator**. It *asks if two values are equal*. (my_variable == 10)

LOGICAL OPERATORS: COMBINING QUESTIONS

Python uses plain English words for logical operations, which is a major win for readability compared to &&, ||, and ! in other languages.

Operator	Description	Example
`and`	Returns `True` if **both** statements are true.	`age > 18 and has_license == True`
`or`	Returns `True` if **at least one** of the statements is true.	`is_weekend or is_holiday`
`not`	Reverses the result, returns `False` if the result is true.	`not is_raining`

- **Chaining Comparisons:** Python allows for a beautifully readable way to check for a range.

```
# The common way in other languages
# if (age >= 13 && age <= 19) { ... }

# The Pythonic way
age = 15
if 13 <= age <= 19:
    print("You are a teenager.")
```

IN-CLASS EXERCISE: ACCESS CONTROL LOGIC

- Write a single boolean expression that determines if a user can access a secure area. The conditions are:
  - The user must be an 'admin' OR have a 'security_clearance' level greater than 4.
  - The user must NOT be on the 'watch_list'.
- Create variables user_role, security_clearance, and on_watch_list to test your expression.

THE RULES OF ENGAGEMENT - PRECEDENCE

When you have multiple operators in one expression, Python follows a specific order of operations, similar to PEMDAS in math.

- () (Parentheses) - Always evaluated first.
- ** (Exponentiation)
- *, /, //, % (Multiplication, Divisions, Modulus) - Evaluated left-to-right.
- +, - (Addition, Subtraction) - Evaluated left-to-right.
- <, <=, >, >=, !=, == (Comparison Operators)
- not (Logical NOT)
- and (Logical AND)
- or (Logical OR)

- **The Golden Rule:** You don't need to memorize the entire list. **When in doubt, use parentheses () to make your intentions clear.** Code should be readable first and foremost.

```
# Ambiguous:
result = 5 + 10 * 2 # result is 25

# Clear and explicit:
result = (5 + 10) * 2 # result is 30
```

IN-CLASS EXERCISE: PREDICT THE OUTPUT

- Without running the code, what will be the final boolean value of can_proceed?

```
score = 85

time_remaining = 10

has_bonus = False

can_proceed = score > 80 and time_remaining > 0 or has_bonus and not score > 90

print(can_proceed)
```

HANDS-ON LAB: THE ULTIMATE SIMPLE CALCULATOR

- Create a new file named calculator.py. Your script will take two numbers from the user and perform a series of operations and checks on them.

PART 1: SETUP AND ARITHMETIC

- Prompt the user to enter the first number and store it in a variable num1.
- Prompt the user to enter the second number and store it in a variable num2.
- **Important:** Convert both inputs to floats to handle decimal values.
- Perform all arithmetic operations (+, -, *, /, //, %, **) on num1 and num2.
- Print the result of each operation with a descriptive label. For example:

```
--- Arithmetic Results ---
Sum:        15.0
Difference: 5.0
...etc.
```

PART 2: COMPARISON CHECKS

- Perform a series of comparisons between num1 and num2.
- Print the boolean result of each comparison. For example:

```
--- Comparison Results ---
Is num1 equal to num2?      False
Is num1 greater than num2?   True
...etc.
```

PART 3 (CHALLENGE): LOGICAL SCENARIOS

- Write a boolean expression to check if **both** numbers are positive. Store the result in a variable both_positive and print it.
- Write a boolean expression to check if **at least one** of the numbers is greater than 100. Store the result in one_is_large and print it.
- Write a boolean expression to check if num2 is not zero. This is a crucial check before performing division. Store the result in is_division_safe and print it.
- Combine the previous check: print the result of the true division (/) **only if** is_division_safe is True.