📚 Variables in Python Basics

Brief Overview

This note covers **Variables in Python** and was created from the Lecture 2 : What is a variable? | Operations on variables in Python | #python#datasciencecourse#2025 YouTube video. It introduces how to declare and reassign variables, explores common data types, demonstrates string operations, and explains arithmetic rules and compound assignments.

Key Points

- Variable declaration and dynamic reassignment
- Integer, float, and string types with basic operations
- String concatenation, repetition, and invalid arithmetic
- Arithmetic rules, modulo, PEMDAS, and compound assignments

📚 Definition of a Variable

> *A **variable** is a named location in memory that holds a **value**. The name is an identifier you can use to refer to the stored value.*

- Syntax: variable_name = value
- The name is arbitrary (e.g., apple), the value can be an integer, float, string, etc.

🖨️ Printing a Variable

> *Printing a variable displays its current **value**, not its name.*

Example (Python):

```
apple = 2
print(apple)   # Output: 2
```

🔄 Dynamic Change of a Variable

- A variable's value can be **reassigned** at any time.
- After reassignment, the previous value is forgotten.

Example:

```
apple = 2
apple = 4
print(apple)   # Output: 4
```

📊 Data Types That Can Be Assigned

Data Type	Example Assignment	Typical Operations
Integer	apple = 4	arithmetic (+, *, /, **, %)
Float	price = 3.14	same as integers, results may be non-integer
String	greeting = "hello"	concatenation (+), repetition (*)

🔤 String Operations

- **Concatenation** (+): joins two strings.

  ```
  result = "Hello" + "World"   # "HelloWorld"
  ```

- **Repetition** (*): repeats a string *n* times.

  ```
  result = "telephone" * 3     # "telephonetelephonetelephone"
  ```

- Multiplying a string by **0** yields an empty string ("").

❌ Invalid String Arithmetic

- Multiplying a string by another string (e.g., "apple" * "apple") causes an error.

➕ Arithmetic Operations with Numbers

- **Addition**: a + b
- **Multiplication**: a * b
- **Exponentiation**: a ** b (e.g., $64^{0.5} = 8$)
- **Modulo** (%): returns the remainder.
  - Example: $57 \bmod 2 = 1$
  - Example: $50 \bmod 2 = 0$

📈 Compound Assignment Operators

Operator	Meaning	Example
+=	Add and assign	i = 4 → i += 1 → i becomes 5
*=	Multiply and assign	i = 2 → i *= 3 → i becomes 6
-=	Subtract and assign	i = 4 → i -= 1 → i becomes 3

These operators are shorthand for variable = variable value.

🔁 Swapping Variable Values

- Simultaneous assignment lets you exchange values without a temporary variable.

```
a = 1
b = 3
a, b = b, a    # a becomes 3, b becomes 1
print(a, b)    # Output: 3 1
```

📐 Order of Operations (PEMDAS)

1. Parentheses ()
2. Exponents **
3. Multiplication * and Division / (left-to-right)
4. Addition + and Subtraction - (left-to-right)

Example Calculation

Expression: 4 \times 3 / 2 + 8 - 18

Step-by-step:

1. $4 \times 3 = 12$
2. $12 \div 2 = 6$
3. $6 + 8 = 14$
4. $14 - 18 = -4$

Result: -4 (a float if any division yields a non-integer).

📚 Key Takeaways

- Variables store values; the name is just a reference.
- Values can change dynamically via reassignment.
- Strings support concatenation (+) and repetition (*), but not arithmetic multiplication.
- Numeric operations follow standard arithmetic rules; % gives the remainder.
- Compound assignments (+=, *=, etc.) simplify common update patterns.
- Simultaneous assignment enables easy swapping of values.
- Follow the order of operations to evaluate complex expressions correctly.