Day 2 - Facilitation Guide
Variables and Operators

Index

(1.5 hrs) ILT

I. Recap

In our last introductory session we learned:

- Writing programs (or programming) is a very creative and rewarding activity. You can write programs for many reasons, ranging from making your living to solving a difficult data analysis problem, to having fun, to helping someone else solve a problem.
- Computer hardware architecture involves various components to execute and define a particular task.
- Every programming language has a vocabulary and Grammar.
- Interpreter executes code line by line, whereas the compiler executes the code as a whole.
- Program is a set of instructions used to execute a particular task.

In this session we are going to learn about Variables and constants,Operators and its Precedence.

II. Variables and Constants

Variables and constants are fundamental concepts in programming used to store and manage data. They serve distinct purposes and have different characteristics:

Variables

- **Mutable**: Variables are mutable, which means their values can change during the program's execution. You can assign different values to a variable as needed.
- **Storage**: Variables are used to store and manage data that may change or vary over time. For example, you can use variables to store user input, calculation results, or intermediate values in your program.
- **Declaration**: In many programming languages, you need to declare variables before using them. The declaration specifies the variable's name and data type

Data types

Data types are used to categorize and specify the type of data that a variable can hold or the type of value that an expression can produce. Data types are essential for efficient memory allocation, defining the operations that can be performed on the data, and ensuring type safety.

Common programming languages, including Python, Java, C++, and JavaScript, have their own data types. Here are some commonly used data types:(e.g., integer, string, float).

Example :

```
# Variable declaration and assignment
age = 25
name = "Alice"

# Updating variable values
age = 26
print(name)
print(age)
```

Constants:

- **Immutable**: Constants are immutable, meaning their values remain fixed and cannot be changed once they are assigned.
- **Storage**: Constants are used to represent values that are known and unchanging throughout the program. They provide a way to give meaningful names to such values, enhancing code readability.
- **Declaration**: Depending on the programming language, constants may require explicit declaration or follow specific naming conventions (e.g., all uppercase).

Example :

```
# Constant declaration (using uppercase convention)
PI = 3.14159265359
MAX_VALUE = 100

# Attempting to change a constant value (will result in an error)
PI = 3.14  # Error: Cannot assign to a constant
```

Note: Python does not have *built-in support for constants* in the same way they do for variables. In such cases, programmers use naming conventions (e.g., uppercase) to indicate that a variable's value should not be changed.

Syntax

Python syntax refers to the rules and structure that dictate how Python code should be written in order for it to be valid and interpretable by the Python interpreter. Proper adherence to Python syntax is crucial for writing correct and functional Python programs. Here are some key aspects of Python syntax:

Indentation: Python uses whitespace (indentation) to indicate blocks of code. Indentation is not optional; it's a fundamental part of Python's syntax. Typically, four spaces are used for each level of indentation.

Example:

```
if 10> 5:
        print("x is greater than 5")
```

Variable naming conventions

Food for thought:
Imagine a situation if you get a string without any demarcations.
Eg. whatyouseeiswhatyougetromewasnotbuiltinaday

Naming conventions for variables are vital in programming to create code that is readable, maintainable, and steady. Different programming languages and groups may

also have slightly various conventions, but a few popular concepts are observed across many languages. Here are common variable naming conventions.

1. Use Descriptive Names: Choose variable names that clearly and concisely describe the data they store or represent. Avoid single-letter or cryptic names whenever possible.

Good: `user_age`, `total_count`, `customer_name`

Bad: `x`, `temp1`, `var1`

2. Use CamelCase or Snake_case: Variable names can be written in CamelCase or snake_case, depending on the language and community conventions.

- **CamelCase:** Words are concatenated without spaces, and each word (except the first) begins with a capital letter. This convention is often used in languages like Java and JavaScript.

  Example: `userName`, `totalAmount`, `calculateTaxRate`

- **snake_case:** Words are separated by underscores, and all letters are typically lowercase. This convention is commonly used in Python and some other languages.

  Example: `user_name`, `total_amount`, `calculate_tax_rate`

3. **Start with a Letter:** Variable names must start with a letter (A-Z or a-z) or an underscore (_) in many programming languages. While some languages allow digits (0-9) after the first character, it's a good practice to start with a letter.

Example: `_private_variable`, `name`, `score_1`

4. Avoid Reserved Words: Do not use language keywords or reserved words as variable names, as they have special meanings in the language.

Bad: `if`, `for`, `while`, `int`, `float`

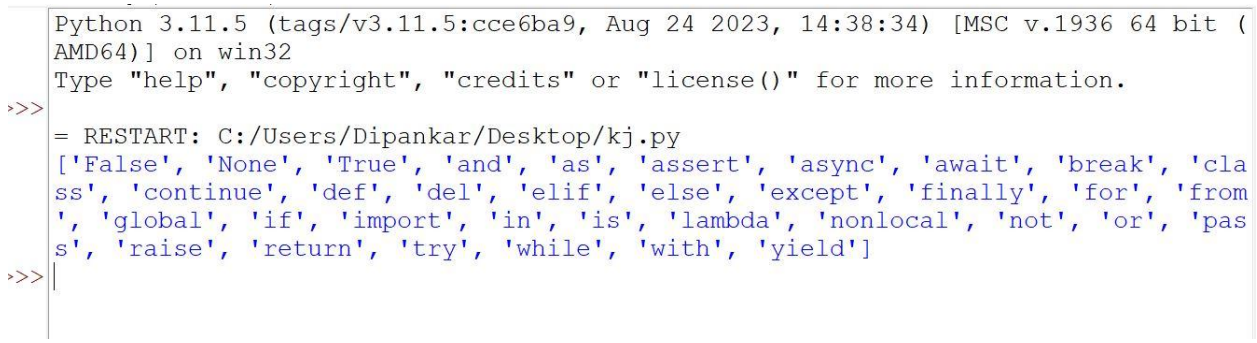
You can use Python's keyword module to programmatically check and list the reserved keywords in Python. Here's a Python program to do that:

```
import keyword
```

```
# Get the list of reserved keywords
reserved_keywords = keyword.kwlist


print(reserved_keywords)
```

`Output:



5. **Be Consistent:** Stick to a consistent naming convention throughout your codebase. If you choose one style (e.g., CamelCase or snake_case), use it consistently for all variables.

6. Use Meaningful Prefixes/Suffixes: For specific types of variables, consider adding prefixes or suffixes that indicate their purpose. This can make your code more self-explanatory.

- **Prefix for Booleans**: Prefix boolean variables with "is," "has," or a similar word to indicate their purpose.

  Example: `is_valid`, `has_permission`

- **Suffix for Arrays and Collections**: You can add a suffix like "_list," "_array," or "_set" to variables that store collections.

  Example: `names_list`, `scores_array`

7. Avoid Acronyms and Abbreviations: While some acronyms and abbreviations may be widely recognized, it's generally better to use full words for variable names to improve code readability.

Good: `maximum_value`, `user_interface`

Avoid: `max_val`, `UI`

8. Keep Names Concise but Not Too Short: Variable names should be concise and meaningful but not overly short.

9. Avoid One-Character Names: With rare exceptions (e.g., loop counters in small scopes), avoid using single-character variable names. Meaningful names are much more helpful for understanding your code.

Avoid: `i`, `x`, `j`

Remember that clear and consistent variable naming is a crucial aspect of writing clean, maintainable, and understandable code. Follow the conventions of the programming language you're using and consider any specific guidelines or standards used in your development team or community.

III. Python Operators and Precedence

Python Operators : In Python, operators are symbols or special keywords that are used to perform operations on values or variables. Python provides a wide range of operators that can be categorized into several groups based on their functionality. Here are some of the most commonly used operators in Python:

Arithmetic Operators:

- **+ (Addition):** Adds two values.
- **- (Subtraction):** Subtracts the right operand from the left operand.
- *** (Multiplication):** Multiplies two values.
- **/ (Division):** Divides the left operand by the right operand.
- **% (Modulus):** Returns the remainder of the division.
- **** (Exponentiation):** Raises the left operand to the power of the right operand.
- **// (Floor Division):** Returns the integer part of the division result (floor value).

Example:

```
x = 10
y = 3
addition = x + y
division = x / y
modulus = x % y
print(addition)
print(division )
print(modulus)
```

Comparison Operators

- **== (Equal to):** Checks if two values are equal.
- **!= (Not equal to):** Checks if two values are not equal.
- **< (Less than):** Checks if the left operand is less than the right operand.
- **> (Greater than):** Checks if the left operand is greater than the right operand.
- **<= (Less than or equal to):** Checks if the left operand is less than or equal to the right operand.
- **>= (Greater than or equal to):** Checks if the left operand is greater than or equal to the right operand.
-

Example:

```
a = 5
b = 10
is_equal = (a == b)
is_not_equal = (a != b)
is_less_than = (a < b)
print(is_equal)
print(is_not_equal)
print(is_less_than)
```

Logical Operators

- **and (Logical AND):** Returns True if both conditions are True.
- **or (Logical OR):** Returns True if at least one condition is True.
- **not (Logical NOT):** Inverts the result of the condition.

Example:

```
x = True
y = False
result_and = x and y
result_or = x or y
result_not = not x
print(result_and)
print(result_or)
print(result_not)
```

Assignment Operators:

- **= (Assignment):** Assigns a value to a variable.
- **+=, -=, *=, /=, %=, **=, //= (Augmented Assignment):** Performs an operation and assigns the result to the variable.

Example:

```
x = 5
x += 2  # Equivalent to x = x + 2
print(x)
```

Bitwise Operators (used for bitwise manipulation):

- & (Bitwise AND)
- | (Bitwise OR)
- ^ (Bitwise XOR)
- ~ (Bitwise NOT)
- << (Left Shift)
- >> (Right Shift)

Example:

```
a = 5  # Binary: 0101
b = 3  # Binary: 0011
bitwise_and = a & b  # Result: 0001 (Decimal: 1)
print(bitwise_and)
```

Membership Operators (used to test if a value is a member of a sequence):

- in (Membership)
- not in (Negated Membership)

Example:

```
#declare list
fruits = ["apple", "banana", "cherry"]
#Declare string
is_apple_in_list = "apple" in fruits
is_mango_not_in_list = "mango" not in fruits
print(is_apple_in_list)
print(is_mango_not_in_list)
```

Identity Operators (used to compare the memory addresses of objects):

- is (Identity)
- is not (Negated Identity)

Example:

```
x = [1, 2, 3]
y = x
is_same_object = x is y
print(is_same_object)
```

Ternary Operator (conditional expression):

- value_if_true if condition else value_if_false

Example:

```
age = 20
is_adult = True if age >= 18 else False
print(is_adult)
```

These are some of the commonly used operators in Python. Depending on your programming needs, you'll use various operators to perform operations, comparisons, and logical evaluations in your code.

Operator precedence

Operators in Python have a specific precedence, which determines the order in which they are evaluated in an expression. When an expression contains multiple operators, Python follows the operator precedence rules to determine which operations to perform first. If needed, you can use parentheses to override the default precedence and explicitly specify the order of operations.

Here is a summary of common operators in Python and their precedence, from highest to lowest:

- Parenthesis are always respected hence given first priority.
- Exponentiation (raise to a power).
- Multiplication, Division, and Remainder.
- Addition and Subtraction.

Here's an example that demonstrates operator precedence in Python:

```
# Example 1: Arithmetic Operators

result = 10 + 5 * 2  # Multiplication has higher precedence than addition
print(result)  # Output: 20
```

```
# Example 2: Parentheses can be used to change precedence

result = (10 + 5) * 2  # Parentheses force addition to be evaluated first
print(result)  # Output: 30
```

```
# Example 3: Comparison Operators

x = 5
y = 10
z = 15
result = x < y <= z  # Chained comparison operators have left-to-right precedence
print(result)  # Output: True
```

```
# Example 4: Logical Operators
a = True
b = False
```

```
c = True
result = a or b and c  # Logical AND has higher precedence than OR
print(result)  # Output: True
```

```
# Example 5: Assignment Operators
value = 5
value += 10 * 2  # Multiplication has higher precedence than assignment
print(value)  # Output: 25
```

BODMAS (or PEMDAS in some regions) is an acronym that represents the order of operations to evaluate mathematical expressions. It stands for:

Brackets (or Parentheses): Perform operations inside parentheses first.
Orders (or Exponents): Evaluate exponentiation (powers and roots) next.
Division and Multiplication: Perform multiplication and division from left to right.
Addition and Subtraction: Perform addition and subtraction from left to right.

Here's an example of how BODMAS is applied to evaluate a mathematical expression:

```
Expression: 2 * (3 + 4) - 5^2

Step 1: Brackets (Parentheses)
  = 2 * (7) - 5^2  # Evaluate the expression inside the parentheses.

Step 2: Orders (Exponents)
  = 2 * 7 - 25  # Evaluate the exponentiation (5^2).

Step 3: Division and Multiplication (from left to right)
  = 14 - 25  # Perform multiplication (2 * 7).

Step 4: Addition and Subtraction (from left to right)
  = -11  # Perform subtraction (14 - 25).

Result: -11
```

So, following the BODMAS rule, the value of the expression is -11. This rule ensures that mathematical expressions are evaluated in a consistent and unambiguous manner.

In each of these examples, the operators are evaluated based on their precedence, and the expressions are computed accordingly. Understanding operator precedence is essential to write correct and predictable code, as it helps you avoid unexpected results or the need for excessive parentheses to clarify the order of evaluation. You can refer to the documentation of your specific programming language to learn more about operator precedence rules, as they may vary between languages.

Exercise ChatGPT

1. Hi, can you help me to write a program in Python using a ternary operator to check whether the given number is even or odd. We need to specify a dummy number in the code.
2. Please help me with a program in Python to calculate the power of any number. We need to specify a dummy number in the code.