101 Workshop: Python Programming Basics

Presented by:
Nobel Wong
TM Technology Sdn Bhd

< / > TM Technology
Software | Cloud | Automation

Content

1. Welcome & Setup
2. Programming and Python
3. Intro to Programming Concepts
4. Variables & Data Types
5. Input & Output
6. Break
7. Control Flow - Conditions
8. Loops
9. Mini Project: Interactive Script

Background

- Technical Director and Software Engineer for TM Technology Sdn Bhd.
- Bachelor and Masters of Engineering from University of Melbourne.
- 8+ years experience programming
  - Data processing
  - Cloud architecture & computing
  - Web & Mobile app development
  - DevOps
- US & AUS software patent co-author

Welcome & Setup

- Theory and Practical workshop about the fundamental of Python programming.
- Google Collab - Google's Jupyter Notebook to run Python script in the cloud.
- No installation required - https://colab.research.google.com/
- By the end of this 3 hours workshop, everyone should be able to understand basic Python syntax and write a simple interactive Python script using Google Collab.

Welcome & Setup

- https://colab.research.google.com/
- Sign in using Gmail account, notebooks will be synced across different devices.
- Create a new notebook
    - print(“Hello, world!”)

Programming

- The process of providing a computer with a set of instructions, or code, to perform specific tasks or solve problems.
- Instruction comes in different forms and languages.
- Different programming language specialise in different domains.
  - AI & Data Processing - Python
  - Web Development - HTML, CSS, Javascript
  - Database - SQL
  - Engineering - Matlab, R, C
  - Game Development - C++, C#, Java

Programming

- Different programming languages have different syntax, but the fundamentals are still largely the same.
- Python
  - Most used programming language
  - Easiest to learn
  - Most versatile
  - AI, Data Processing, Web & Game Dev, IoT, Cloud, Automation
- Big community online
  - Free resources for self learning, projects, career opportunities

Projects

- Sample project
  - AI & Machine Learning
  - Smart home
  - Websites
  - Web scrapper
  - Simple games
    - Snake game, Sudoku
- Tips for choosing a project
  - Start small
  - Focus on your interest and learning

Python Syntax

```python
# This is a comment
name = "Alice"  # Variable assignment
age = 30

if age > 18:
    print(f"{name} is an adult.")  # Indented block for the if statement
else:
    print(f"{name} is a minor.")

for i in range(3):
    print(f"Loop iteration: {i}") # Indented block for the for loop

def greet(person):
    print(f"Hello, {person}!") # Indented block for the function

greet(name)
```

Errors

```python
# SyntaxError - Missing colon after if statement
if 5 > 2
    print("Five is greater than two!")

# NameError - 'my_variable' is not defined
print(my_variable)

# TypeError - Attemtping to add a string and an integer
result = "hello" + 5

# IndexError - Attempting to access an index beyond the list's
length
my_list = [1, 2, 3]
print(my_list[5])
```

Variables

- Symbolic names that act as containers for storing data values.
- Label and reference information within a program.
- Does not require explicit declaration of a variable’s data type.
- Variable is created the moment a value is assigned to it.
- Dynamic Typing
  - Type of data determined at runtime, variable can even be reassigned to a different data type

Variables

- Assignment
  - Created and assigned values using the assignment operator (=)
  - my_variable = 10
- Case-Sensitivity
  - my_variable and My_Variable treated as two distinct variables.
- Reusability
  - Allow for the reuse of data throughout a program without re-entering each time.

Variables

```python
# Assigning an integer to a variable
age = 30

# Assigning a string to a variable
name = "Alice"

# Assigning a boolean to a variable
is_student = True

# Reassigning a variable to a different type
age = "thirty"

print(age)
print(name)
print(is_student)
```

Types

- Numeric Types
  - **int**: Represents whole numbers (integers), e.g., 5, -10
  - **float**: Represents real numbers with a decimal point (floating-point numbers), e.g., 3.14, -2.5
  - complex: Represents complex numbers, e.g., 1+2j
- Text Type
  - **str**: Represents sequences of characters (strings), enclosed in single or double quotes, e.g., “**Hello**”, ‘**Python**’
- Boolean Type:
  - **bool**: Represents truth values, either **True** or **False**

Types

- Sequence Types:
  - **list**: Represents ordered, mutable collection of items, e.g., [1, 2, 3], [“apple”, “banana”]
  - **range**: Represents an immutable sequence of numbers, often used in loops, e.g., range(5)
- Mapping Type:
  - **dict**: Represents unordered collections of key-value pairs (dictionaries), e.g., {”name”: “John”, “age”: 30}
- None Type:
  - NoneType: Represents the absence of a value, with a single value None

Types

- Type casting
  - Explicit conversion of a value from one data type to another.
  - Achieved using built-in functions that correspond ot the desired data type.
- Common Type Casting Functions
  - **int()**: Converts a value to an integer
  - **float()**: Converts a value to a floating-point number
  - **str()**: Converts a value to a string
  - **bool()**: Converts a value to a boolean. Zero, empty sequences (like empty strings, lists, tuples), and **None** evaluate to **False**. All other values evaluate to **True**

Types

- Important Considerations:
  - Data Loss:
    - Casting a float to an integer truncates the decimal part, leading to potential data loss
  - ValueError
    - Attempting to cast a value to an incompatible type will result in a ValueError
  - Temporary vs Permanent Change:
    - Type casting within an expression is temporary. To permanently change the data type of a variable, you must reassign the variable with the casted value.

Types

- The type() function can be used to determine the data type of a variable at runtime, for example:

```
x = 10
print(type(x))  # Output: <class
'int'>
y = "Hello"
print(type(y))  # Output: <class
'str'>
```

Hands-on Task

- Create and print variables for name, age, favourite number
- Use name, age, fav_number as variable names
  - name should be **string**
  - age should be **int**
  - fav_number can be **int** or **float**
- print type and value of variables

Variables

Input & Output

- Input and output operations are fundamental for program interaction.
- Input
  - The input() function is used to obtain data from the user.
  - Pauses program execution and waits for the user to type something and press Enter,
  - By default, the input() function returns the user's input as a string.

Input & Output

- In this example, the program prompts the user to enter their name, stores the input in the name variable, and then prints a greeting using the entered name.

Input & Output

- Output
  - The print() function is used to display information to the console. It can display text, variables, and expressions.
  - Multiple items can be printed by separating them with commas within the print() function, which will add spaces between them by default.

Input & Output

- Using input() to get data from user
- Using string concatenation and f-strings

```
name = input("What's your name? ")
print(f"Hello, {name}!")
```

Mini Task

- Ask for birth year and calculate/display age

Flow & Conditions

- Flow of control dictates the order in which code statements are executed.
- Python scripts are **mostly** executed in sequential order.
- This sequential execution can be altered based on conditions and iterations.
- Conditional statements allow specific blocks of code to execute only if certain conditions are met.
- Python uses if, elif (else if), and else keywords for this purpose.

Flow & Conditions

- **if** statement: Executes a block of code if a condition is True.

```python
if (condition):
  # do something

# example code
age = 20

if age >= 18:
  print("You are eligible to vote!")
```

Flow & Conditions

- **elif** statement: Provides an alternative condition to check if the preceding if or elif conditions were False.

```
if (condition1):
  # do something
elif (condition 2):
  # do something else

# example
temperature = 25
if temperature > 30:
    print("It's hot outside.")
elif temperature > 20:
    print("It's warm outside.")
```

Flow & Conditions

- **else** statement: Executes a block of code if none of the preceding if or elif conditions were True.

```python
# example
temperature = 25
if temperature > 30:
  print("It's hot outside.")
elif temperature > 20:
  print("It's warm outside.")
else:
  print("It's cool outside")
```

Logical Operators

- Three primary logical operators: **and**, **or**, and **not**.
- These operators are used to combine or modify Boolean expressions and return a Boolean result (True or False) and are fundamental for controlling program flow in conditional statements and loops.
- Short-Circuit Evaluation
  - Python's and and or operator utilize short-circuit evaluation.
- These logical operators are fundamental for controlling program flow in Python, particularly within conditional statements (if, elif, else) and loop control.

Logical Operators

- **and** operator
  - Returns **True** if both operands are **True**.
  - Returns **False** if either or both operands are **False**.
  - Example: (x > 5) and (y < 10) will be **True** only if x is greater than 5 and y is less than 10.
  - If the first operand is **False**, the second operand is not evaluated because the result will already be **False**.

Logical Operators

- **or** operator
  - Returns **True** if at least one of the operands is **True**.
  - Returns **False** only if both operands are **False**.
  - (age < 18) or (is_student) will be **True** if age is less than 18 or is_student is **True** (or both).
  - If the first operand is **True**, the second operand is not evaluated because the result will already be **True**.

Logical Operators

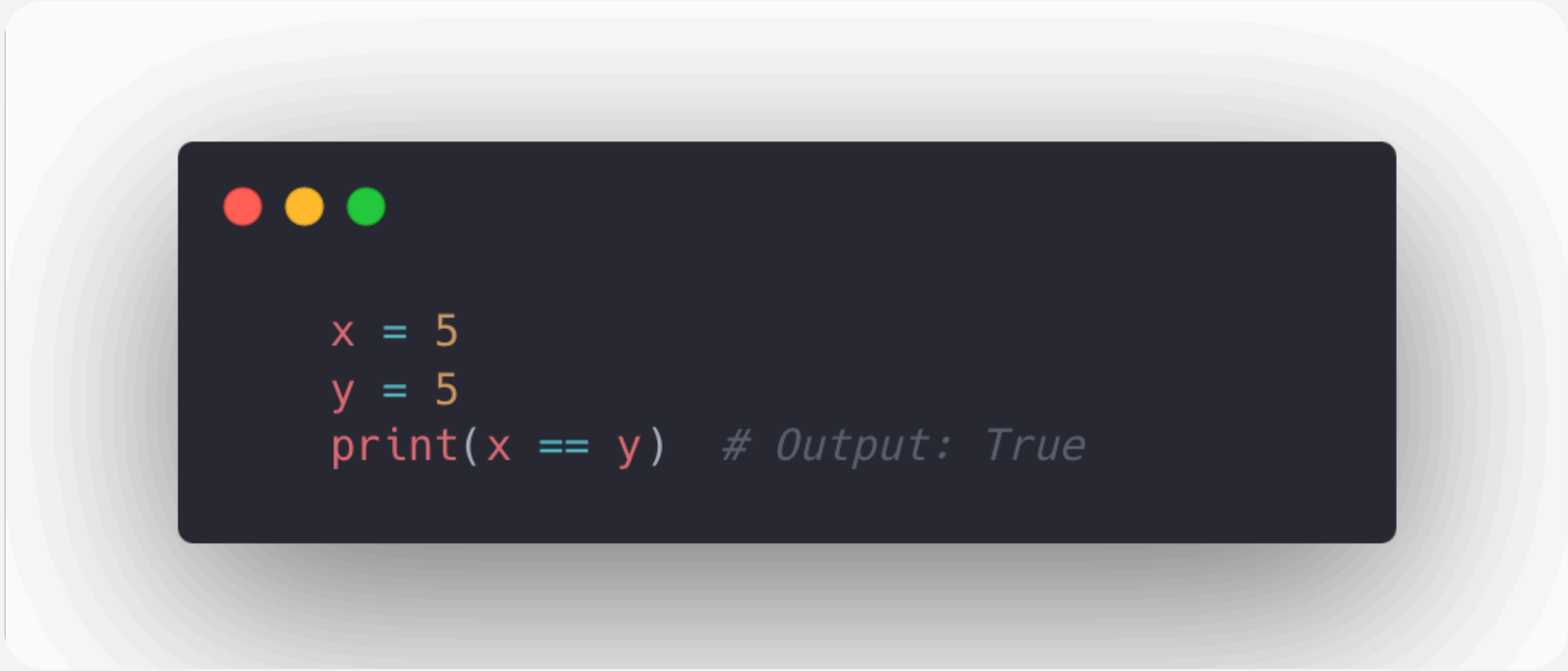
- **not** operator
  - This is a unary operator, meaning it operates on a single operand.
  - Reverses the truth value of the operand: **True** becomes **False**, and **False** becomes **True**.
  - not (is_active) will be **True** if is_active is **False**, and **False** if is_active is **True**

Comparison Operators

- Python’s comparison operators, also know as relational operators, are used to compare two values and return a Boolean result (either **True** or **False**).
- These operators are fundamental for controlling program flow through conditional statements.
- There are six primary comparison operators in Python.
- These operators can be used to compare various data types, including numbers, strings, and other objects.

Comparison Operators

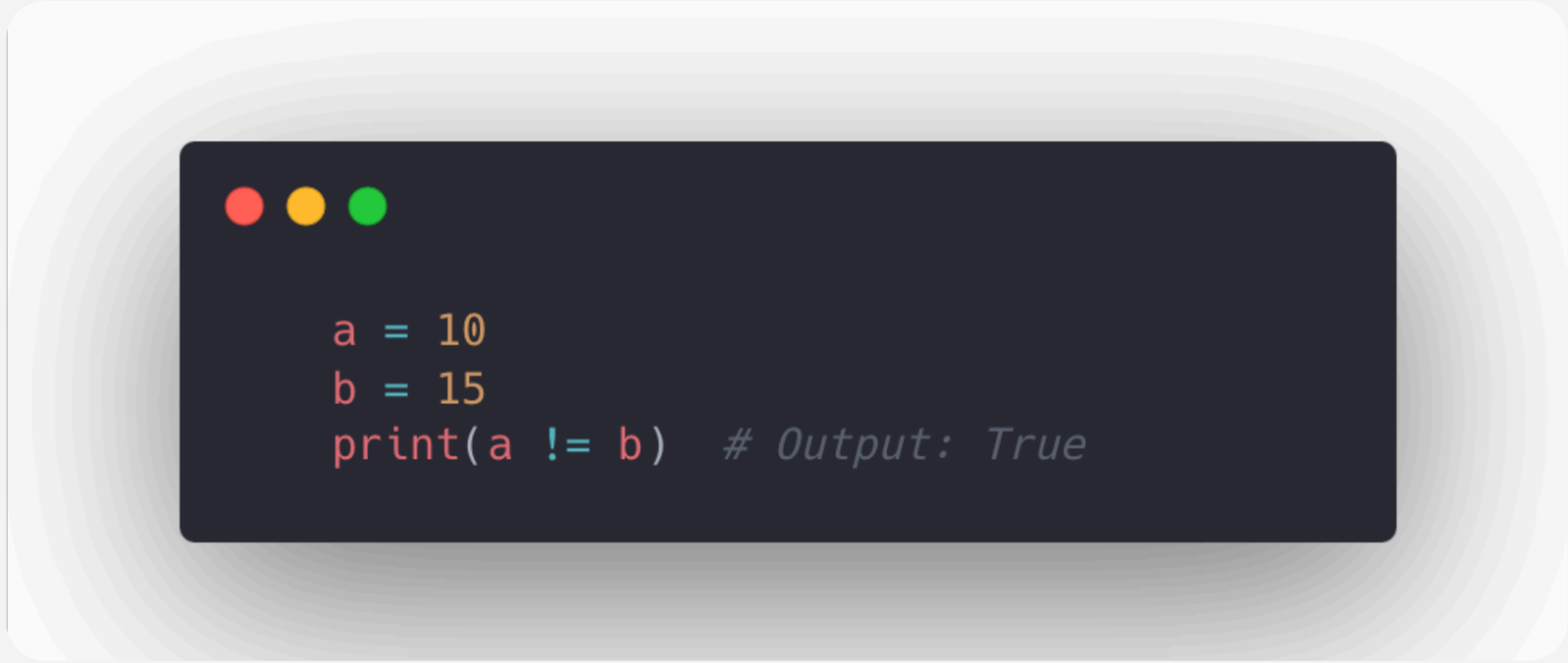
- Equal to (==): Checks if two values are equal.



```
x = 5
y = 5
print(x == y)  # Output: True
```

Comparison Operators

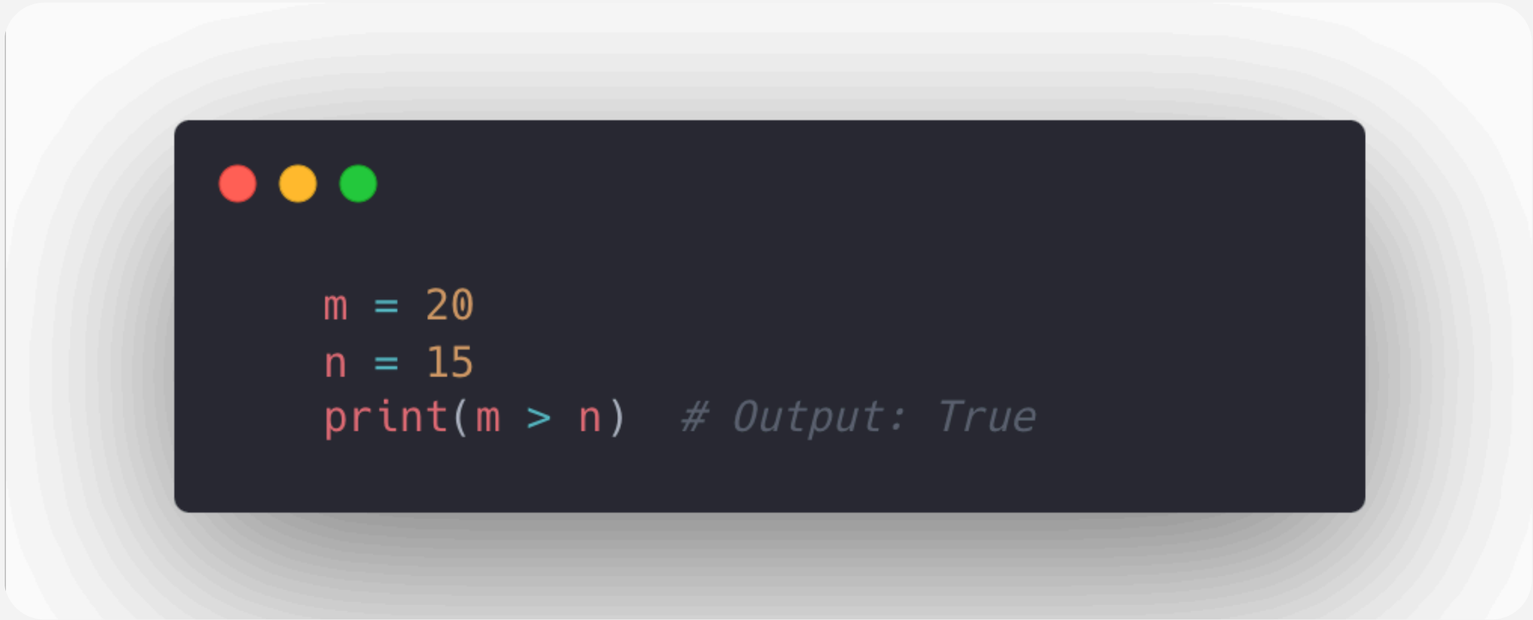
- Not equal to (!=): Checks if two values are not equal.



```
a = 10
b = 15
print(a != b)  # Output: True
```

Comparison Operators

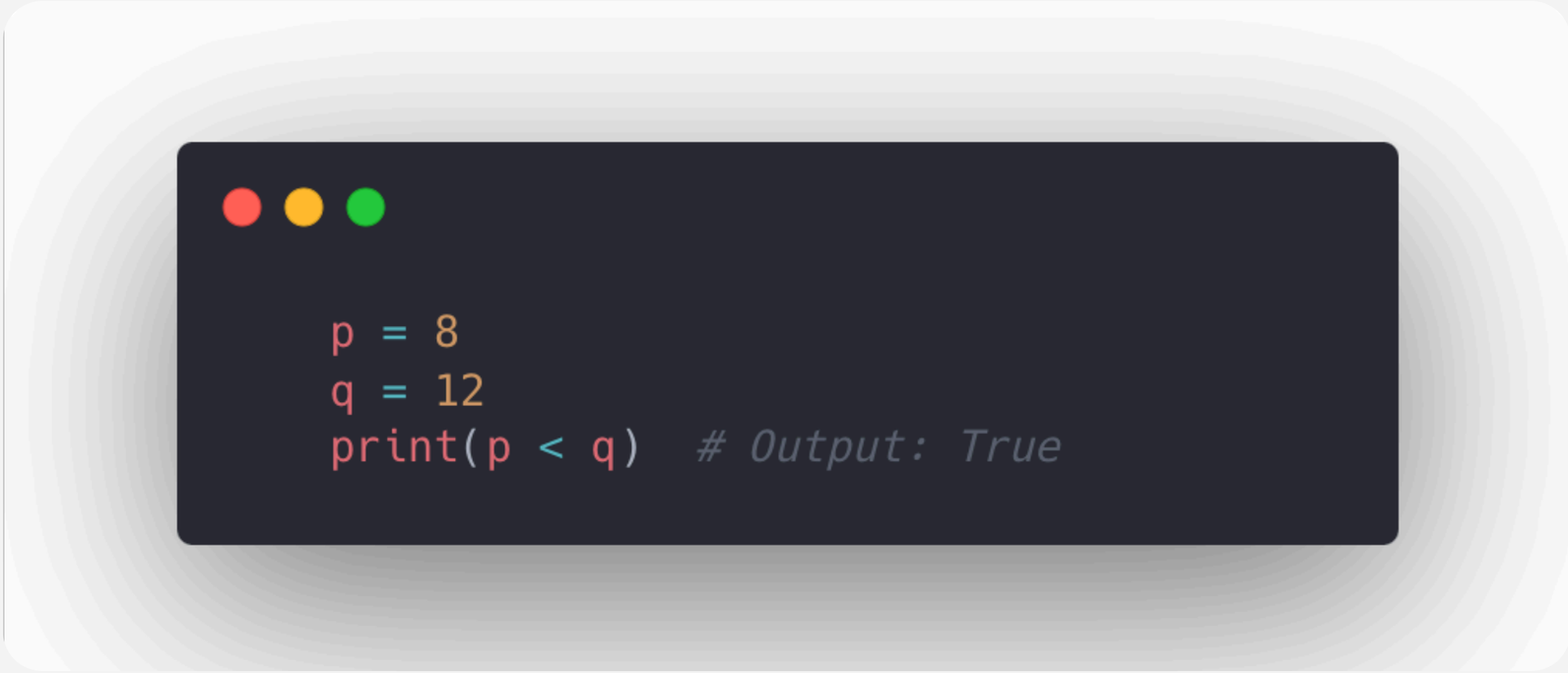
- Greater than (>): Checks if the left operand is greater than the right operand.



```
m = 20
n = 15
print(m > n)  # Output: True
```

Comparison Operators

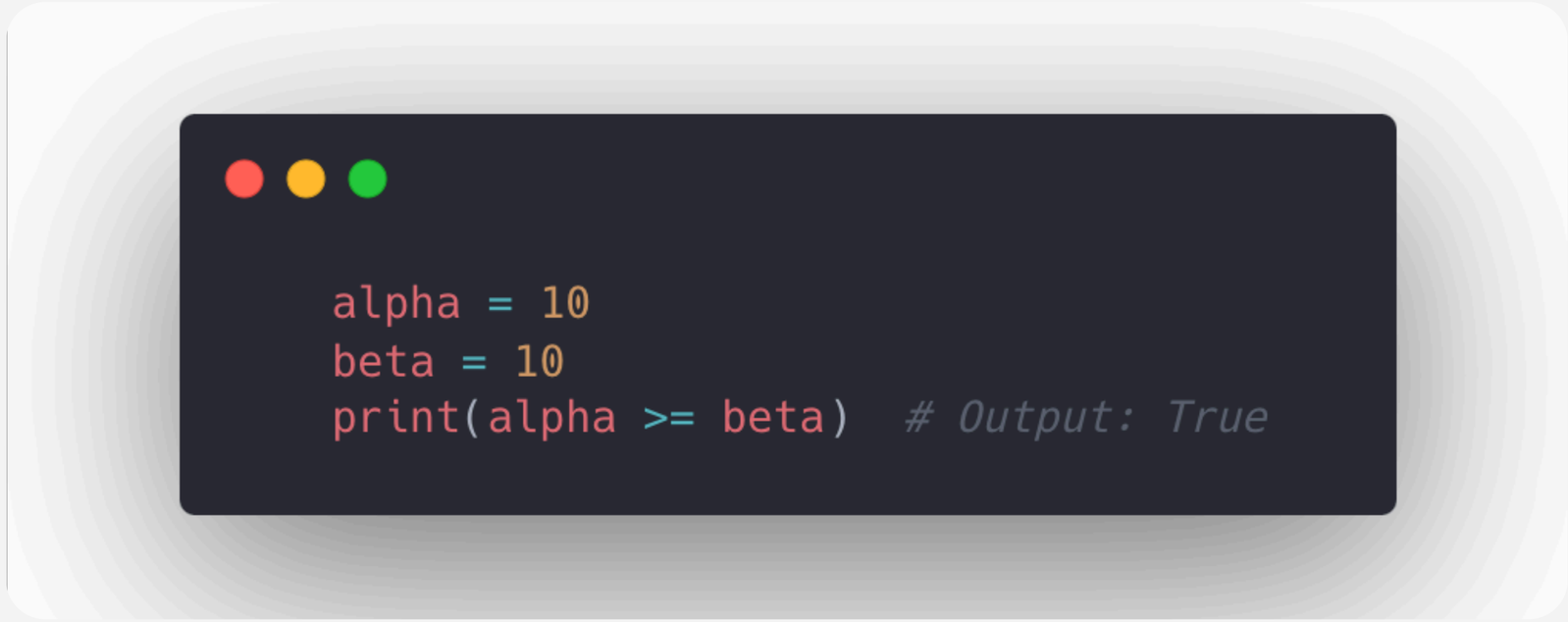
- Less than (<): Checks if the left operand is less than the right operand.



```
p = 8
q = 12
print(p < q)  # Output: True
```

Comparison Operators

- Greater than or equal to (>=): Checks if the left operand is greater than or equal to the right operand.



```
alpha = 10
beta = 10
print(alpha >= beta)  # Output: True
```

Comparison Operators

- Less than or equal to (<=): Checks if the left operand is less than or equal to the right operand.

```
gamma = 5
delta = 7
print(gamma <= delta)  # Output: True
```

Indentation

- Indentation refers to the spaces at the beginning of a code line.
- In other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.
- Inconsistent indentation within the same block lead to an IndentationError.
- Indentation is achieved using whitespace at the beginning of a line.
- Indentation levels indicate the hierarchical structure of the code.

Indentation

```python
def my_function():
    # This line is indented, belonging to my_function
    print("Hello from the function!")

if True:
    # These lines are indented, belonging to the if block
    print("This condition is true.")
    x = 10
else:
    # These lines are indented, belonging to the else block
    print("This condition is false.")

for i in range(3):
    # This line is indented, belonging to the for loop
    print(f"Loop iteration: {i}")
```

Mini Task

- Task: Create a voting eligibility checker
- If voter is 18 years old and above, they are eligible to vote.
- Use input() to get age and output() to display message

```
if (condition):
    # do something
```

Loops

- Loops are control flow statements used to repeatedly execute a block of code.
- Python offers two primary types of loops: **for** loops and **while** loops.
- **for** loops:
  - Used for iterating over a sequence (such as a list, tuple, string, or range) or other iterable objects.
- **while** loops:
  - Execute a block of code as long as a specified condition remains True. The loop continue until the condition evaluates to False.

For Loop

- for loops:
  - Used for iterating over a sequence (such as a list, tuple, string, or range) or other iterable objects.

```python
# Iterating through a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# Iterating through a string
for char in "Python":
    print(char)

# Iterating through a range of numbers
for i in range(5):  # Iterates from 0 up to (but not including) 5
    print(i)
```

While Loop

- while loops:
  - Execute a block of code as long as a specified condition remains True. The loop continue until the condition evaluates to False.

```python
# Example of a while loop
count = 0
while count < 3:
    print(f"Count is: {count}")
    count += 1
```

Loop Control

- **break:**
  - Terminates the current loop prematurely, exiting the loop and continuing execution at the statement immediately following the loop.

```python
# Example of break
for i in range(5):
    if i == 3:
        break
    print(i)
```

Loop Control

- **continue:**
  - Skips the rest of the current iteration of the loop and proceeds to the next iteration.

```python
# Example of continue
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Mini Task

- Option 1: Calculate sum of all numbers from 1 to a given number
- Option 2: Print even numbers from 1 to 20
  - % returns the remainder for e.g., 4 % 2 returns 0 (use to check for even numbers)
- Option 3: Print multiplication table of a given number (2)

Project: Interactive Script

- Task 1: Number Guessing Game
- Task 2: Simple Quiz Game

Wrap-Up

- Variables & Data Types
- Input & Output
- Flow & Condition
- Logical and comparison operators
- Try to work on some simple projects.
- Practice coding questions on leetcode.com
- Follow tutorials on Youtube to build more complex projects.

Q&A Session

Thank you

Presented by:
Nobel Wong
TM Technology Sdn Bhd