ST. JOSEPH'S
COLLEGE OF ENGINEERING
AND TECHNOLOGY,
- PALAI -
AUTONOMOUS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ALGORITHMIC THINKING WITH PYTHON

Prof. Sarju S

1 October 2024

Module 1

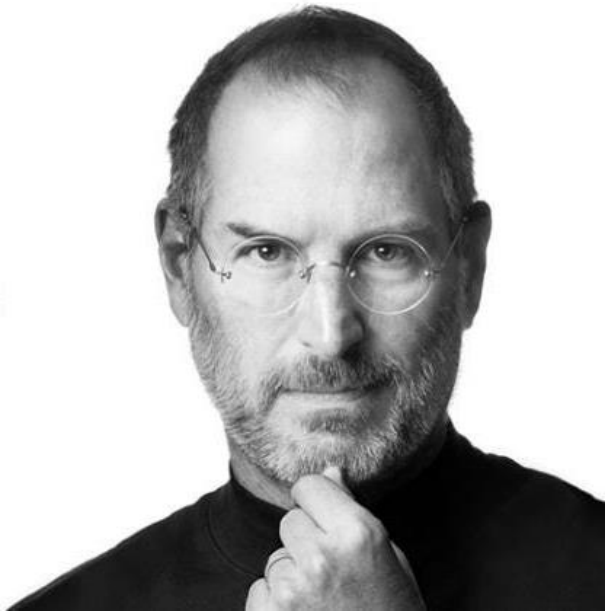
Module 1

- PROBLEM-SOLVING STRATEGIES:- Problem-solving strategies defined, Importance of understanding multiple problem-solving strategies, Trial and Error, Heuristics, Means-Ends Analysis, and Backtracking (Working backward).

- THE PROBLEM-SOLVING PROCESS:- Computer as a model of computation, Understanding the problem, Formulating a model, Developing an algorithm, Writing the program, Testing the program, and Evaluating the solution.

- ESSENTIALS OF PYTHON PROGRAMMING:- Creating and using variables in Python, Numeric and String data types in Python, Using the math module, Using the Python Standard Library for handling basic I/O - print, input, Python operators and their precedence.

"Everyone should learn how to program because it teaches you how to think" – Steve Jobs.



Python® – the language of today and tomorrow

Python® – the language of today and tomorrow

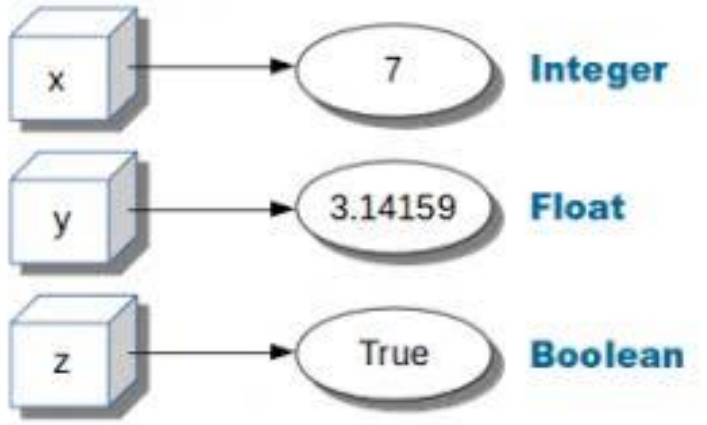
- Python was created by **Guido van Rossum**, and first released on **February 20, 1991**.
- The name of the Python programming language comes from an old **BBC television comedy** sketch series called **Monty Python's Flying Circus**.
- Python is maintained by the **Python Software Foundation**, a non-profit membership organization and a community

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." – Martin Fowler

Creating and using variables in Python

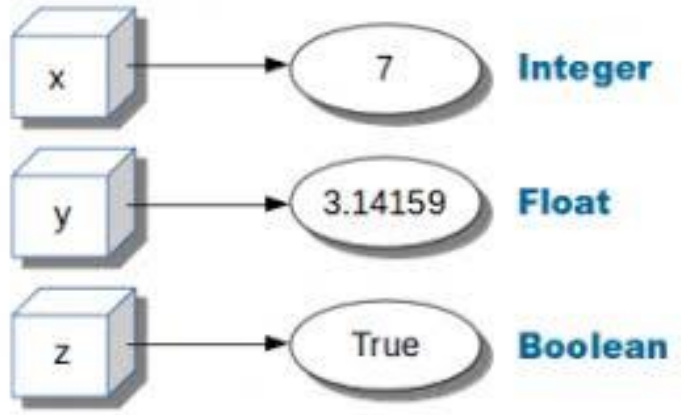
Introduction to Variables in Python

- Definition : Variables serves as a symbolic names for data stored in the memory allowing developers to reference and manipulate values easily.
- They facilitate the storage and retrieval of data, making code dynamic and adaptable to various inputs.



Understanding different Data Types

- **Integers and Floats**: Integers represent whole numbers, while float depict decimal numbers, both of which are essential for numeric computations.
- **Strings as Data Types**: Strings are sequence of characters used for storing and manipulating text, enclosed in either single or double quotes.
- **Booleans Explained**: Booleans only have two possible values: True or False, integral in decision-making process within the code.



Variable Naming Conventions

- **Rules for Naming:** Variable name must begin with a letter or underscore and can include letters, numbers and underscores, maintaining a clear structure.
- **Best Practices Overview**: Adhering to conventions such as using *lower_case_with_underscores* helps enhance the readability of code.
- **Meaningful Naming**: Descriptive names are critical, as they provide context and clarity about the variable's purpose, aiding code comprehension.
- **Case Sensitivity**: Python variables are case-sensitive, meaning *'Variable'* and *'variable'* refer to different entities, which is crucial to remember when coding.

Declaring and Initializing Variables

- **Syntax for Variable Declaration** : In Python, initializing a variable is as simple as assigning a value with the '=' operator, creating a dynamic relationship between name and the value.

- **Example**: x=5 for an integer or name="Alice" for a string, highlighting the simplicity in assigning data to variables.

Scope of Variables

- **Local vs Global Scope:** Local variables are accessible only within the block of code they are defined in, while global variables exist throughout the program, accessible from any function.
- **Lifetime of Variables**: The lifetime refers to the period during which the variable is accessible and usable, influenced by its scope characteristics.

Python data types

Strings

- String is a text data type.
- Python has powerful and flexible built-in string (str) processing capabilities.
- The value of a str object is a sequence of characters.
- You can delimit string values with either single quotes (' ') or double quotes (" ").
- All the characters between the opening and closing delimiter are part of the string.
- For multi-line strings with line breaks, you can use triple quotes.

```
sentence = 'This is a string in single quotes'
print(sentence)
```

```
sentence = "This is a string in double quotes"
print(sentence)
```

```
str_a = """This is a multiline string
Line 1 added
Line 2 added
Line 3 added
"""
print(str_a)
```

```
str_b = '''This is a multiline string using triple quote notation with single quote
Line 1 added
Line 2 added
Let's close this string
'''
print(str_b)
```

Strings

- You can also print selected characters from a string.
- You printed the characters at index 1, index 2, and index 10. Note that **the index count starts at 0**. That is, in 'This', index 0 is 'T'.

```
str_a = "This is a longer string"
print(str_a[1], str_a[2], str_a[10])
```

```
h i l
```

- Python strings are **immutable**; that is, you cannot modify a string without creating a new string variable, or assigning a completely new string value to the variable.

```
: #Let's try to modify str_a now, it will give a runtime error
str_a[10]='S'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_22232\1834719191.py in <module>
      1 #Let's try to modify str_b now, it will give a runtime error
----> 2 str_a[10]='S'

TypeError: 'str' object does not support item assignment
```

Concatenating strings

- Concatenation means to combine two or more strings.
- Once you assign string values to the new variables, you can join them.

```
first_name = 'Jane'
last_name = 'Doe'
name=first_name+last_name
print(name)
```

```
JaneDoe
```

```
first_name = 'Jane'
last_name = 'Doe'
name=first_name+' '+ last_name
print(name)
```

```
Jane Doe
```

Numeric types

- Integers (int) and floats (float) are numeric types, which means they hold numbers. You can perform mathematical operations with them.
- The Python interpreter can then evaluate these expressions to produce numeric values.
- You can use the **type()** function to return the type of a value of variable.

```
age=25
type(age)
```

```
int
```

```
interest_rate=9.75
type(interest_rate)
```

```
float
```

Python Standard Library for handling basic I/O - print, input

Input, Processing, and Output

- `print(<expression>)`
  - When running the print function, Python first evaluates the expression and then displays its value.

```
>>> print ("Hi there")
Hi there
```

  - The syntax for a print statement with two or more expressions looks like the following:

```
print(<expression>,..., <expression>)
```

```
>>> age=25
>>> print("I am",age,"years old")
I am 25 years old
```

Input, Processing, and Output

- Whether it outputs one or multiple expressions, the **print** function always ends its output with a **newline**.
- To begin the next output on the same line as the previous one, you can place the expression **end = “”**, which says *“end the line with an empty string instead of a newline,”*

```
print(<expression>, end = "")
```

```
print("Hello")
print("Python")
```

```
Hello
Python
```

```
print("Hello",end="")
print("Python")
```

```
HelloPython
```

Input, Processing, and Output

- The following example receives an input string from the user and saves it for further processing. The user's input is in black.

```
>>> name = input("Enter your name: ")
Enter your name: Ken Lambert
>>> name
'Ken Lambert'
>>> print(name)
Ken Lambert
```

Input, Processing, and Output

- The following example receives an input string from the user and saves it for further processing. The user's input is in black.

```
>>> name = input("Enter your name: ")
Enter your name: Ken Lambert
>>> name
'Ken Lambert'
>>> print(name)
Ken Lambert
>>>
```

- The input function does the following:
  - Displays a prompt for the input. In this example, the prompt is "Enter your name: ".
  - Receives a string of keystrokes, called characters, entered at the keyboard and returns the string to the shell.

Input, Processing, and Output

- The following example receives an input string from the user and saves it for further processing. The user's input is in black.

```
>>> name = input("Enter your name: ")
Enter your name: Ken Lambert
>>> name
'Ken Lambert'
>>> print(name)
Ken Lambert
>>>
```

- A **variable identifier**, or **variable** for short, is just a name for a value.

```
<variable identifier> = input(<a string prompt>)
```

Input, Processing, and Output

- The **input** function always builds a string from the user's keystrokes and returns it to the program.
- After inputting strings that represent numbers, the programmer must convert them from strings to the appropriate numeric types.
- In Python, there are two **type conversion** functions for this purpose, called int (for integers) and float (for floating point numbers).

```
>>> first = int(input("Enter the first number: "))
Enter the first number: 23
>>> second = int(input("Enter the second number: "))
Enter the second number: 44
>>> print("The sum is", first + second)
The sum is 67
```

Input, Processing, and Output

Function	What It Does
`float(<a string of digits>)`	Converts a string of digits to a floating-point value.
`int(<a string of digits>)`	Converts a string of digits to an integer value.
`input(<a string prompt>)`	Displays the string prompt and waits for keyboard input. Returns the string of characters entered by the user.
`print(<expression>, ...,<expression>)`	Evaluates the expressions and displays them, separated by one space, in the console window.
`<string 1> + <string 2>`	Glues the two strings together and returns the result.

Basic Python functions for input and output

Escape sequences

- An escape sequence refers to a combination of characters beginning with a backslash (\) followed by letters.

Table 6.2: Escape sequences in Python

Escape Sequence	Meaning
\b	Backspace
\n	Newline
\t	Horizontal tab
\v	Vertical tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark

```
Python Console>>> print("Hello world")
Hello world
>>> print("Hello\tworld")
Hello	world
>>> print("Hello\nworld")
Hello
world
```

```
>>> print("The teacher said, \"It\'s very easy to program with Python\"")
The teacher said, "It's very easy to program with Python"
```

Program Comments and Docstrings

- A **comment** is a piece of program text that the computer ignores but provides useful documentation to programmers.
- These comments begin with the **# symbol** and extend to the end of a line.
- Everything from the # to the end of the line is **ignored by the interpreter** while execution – it does not affect the program.

```
>>> sum = 5 + 7 # the variable sum contains the sum of 5 and 7
```

- Python also supports comments that extend multiple lines, one way of doing it is to use # in the beginning of each line.

```
>>> # This is a long comment
>>> # and it extends
>>> # to multiple line
```

Program Comments and Docstrings

- Just as comments are attached to individual statements, you can also include details about the program's purpose at the beginning of the program file.
- This type of comment called a docstring, is a multi-line string.

```
"""
Program name: areaRect.py
Version: 1.1
This program finds the area of a rectangle.
The inputs are two integers representing the length
and breadth of a rectangle, and the output is an
integer named area that represents the area of
the rectangle
"""
```

Operators in Python

Operators in Python

- Operands represent data items on which various operations are performed.
- The operations are denoted by operators. The operands can be constants or variables.
- Operators in Python
  - Arithmetic Operators
  - Comparison (Relational) Operators
  - Assignment Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators
  - Identity Operators

Arithmetic Operators

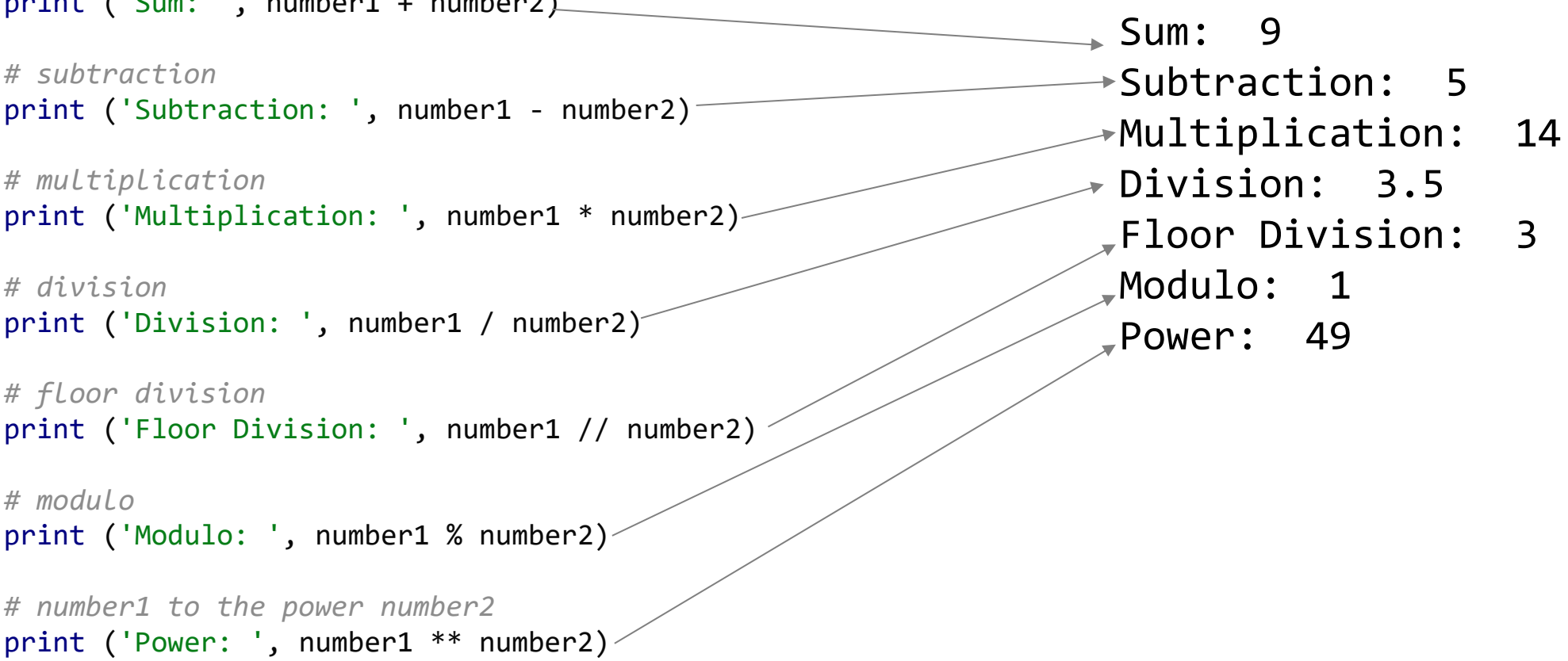
- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

Operator	Operation	Example
+	Addition	5 + 2 = 7
-	Subtraction	4 - 2 = 2
*	Multiplication	2 * 3 = 6
/	Division	4 / 2 = 2
//	Floor Division	10 // 3 = 3
%	Modulo	5 % 2 = 1
**	Power	4 ** 2 = 16

```
number1 = 7
number2 = 2

# addition
print ('Sum: ', number1 + number2)

# subtraction
print ('Subtraction: ', number1 - number2)

# multiplication
print ('Multiplication: ', number1 * number2)

# division
print ('Division: ', number1 / number2)

# floor division
print ('Floor Division: ', number1 // number2)

# modulo
print ('Modulo: ', number1 % number2)

# number1 to the power number2
print ('Power: ', number1 ** number2)
```

Arithmetic Operators

```
number1 = 7
number2 = 2

# addition
print ('Sum: ', number1 + number2)

# subtraction
print ('Subtraction: ', number1 - number2)

# multiplication
print ('Multiplication: ', number1 * number2)

# division
print ('Division: ', number1 / number2)

# floor division
print ('Floor Division: ', number1 // number2)

# modulo
print ('Modulo: ', number1 % number2)

# number1 to the power number2
print ('Power: ', number1 ** number2)
```



Assignment Operators

- Assignment operators are used to assign values to variables

```
# assign 5 to x
x = 5
```

Operator	Name	Example
`=`	Assignment Operator	`a = 7`
`+=`	Addition Assignment	`a += 1 # a = a + 1`
`-=`	Subtraction Assignment	`a -= 3 # a = a - 3`
`*=`	Multiplication Assignment	`a *= 4 # a = a * 4`
`/=`	Division Assignment	`a /= 3 # a = a / 3`
`%=`	Remainder Assignment	`a %= 10 # a = a % 10`
`**=`	Exponent Assignment	`a **= 10 # a = a ** 10`

```
# Initial value
x = 10
# Addition Assignment
x += 5
print("After += operation, x =", x)  # x = x + 5
# Subtraction Assignment
x -= 3
print("After -= operation, x =", x)  # x = x - 3
# Multiplication Assignment
x *= 2
print("After *= operation, x =", x)  # x = x * 2
# Division Assignment
x /= 4
print("After /= operation, x =", x)  # x = x / 4
# Remainder Assignment
x %= 3
print("After %= operation, x =", x)  # x = x % 3
# Exponent Assignment
x **= 3
print("After **= operation, x =", x)  # x = x ** 3
```

Assignment Operators

- Assignment operators are used to assign values to variables

```
# assign 5 to x
x = 5
```

Operator	Name	Example
`=`	Assignment Operator	`a = 7`
`+=`	Addition Assignment	`a += 1 # a = a + 1`
`-=`	Subtraction Assignment	`a -= 3 # a = a - 3`
`*=`	Multiplication Assignment	`a *= 4 # a = a * 4`
`/=`	Division Assignment	`a /= 3 # a = a / 3`
`%=`	Remainder Assignment	`a %= 10 # a = a % 10`
`**=`	Exponent Assignment	`a **= 10 # a = a ** 10`

```
# Initial value
x = 10
# Addition Assignment
x += 5
print("After += operation, x =", x)  # x = x + 5
# Subtraction Assignment
x -= 3
print("After -= operation, x =", x)  # x = x - 3
# Multiplication Assignment
x *= 2
print("After *= operation, x =", x)  # x = x * 2
# Division Assignment
x /= 4
print("After /= operation, x =", x)  # x = x / 4
# Remainder Assignment
x %= 3
print("After %= operation, x =", x)  # x = x % 3
# Exponent Assignment
x **= 3
print("After **= operation, x =", x)  # x = x ** 3
```

Assignment Operators

```
# Initial value
x = 10
# Addition Assignment
x += 5
print("After += operation, x =", x)  # x = x + 5
# Subtraction Assignment
x -= 3
print("After -= operation, x =", x)  # x = x - 3
# Multiplication Assignment
x *= 2
print("After *= operation, x =", x)  # x = x * 2
# Division Assignment
x /= 4
print("After /= operation, x =", x)  # x = x / 4
# Remainder Assignment
x %= 3
print("After %= operation, x =", x)  # x = x % 3
# Exponent Assignment
x **= 3
print("After **= operation, x =", x)  # x = x ** 3
```

```
After += operation, x = 15
After -= operation, x = 12
After *= operation, x = 24
After /= operation, x = 6.0
After %= operation, x = 0.0
After **= operation, x = 0.0
```

Comparison Operators

- Comparison operators compare two values/variables and return a **boolean result**: `True` or `False`

Operator	Meaning	Example
`==`	Is Equal To	`3 == 5` gives us `False`
`!=`	Not Equal To	`3 != 5` gives us `True`
`>`	Greater Than	`3 > 5` gives us `False`
`<`	Less Than	`3 < 5` gives us `True`
`>=`	Greater Than or Equal To	`3 >= 5` give us `False`
`<=`	Less Than or Equal To	`3 <= 5` gives us `True`

```
# Initial values
number1 = 10
number2 = 5
# Is Equal To (==)
print("Is number1 equal to number2? ", number1 == number2)  # False
# Not Equal To (!=)
print("Is number1 not equal to number2? ", number1 != number2)  # True
# Greater Than (>)
print("Is number1 greater than number2? ", number1 > number2)  # True
# Less Than (<)
print("Is number1 less than number2? ", number1 < number2)  # False
# Greater Than or Equal To (>=)
print("Is number1 greater than or equal to number2? ", number1 >=
number2)  # True
# Less Than or Equal To (<=)
print("Is number1 less than or equal to number2? ", number1 <=
number2)  # False
```

Logical Operators

- Logical operators are used to check whether an expression is `True` or `False`. They are used in decision-making.

Operator	Example	Meaning
`and`	a **and** b	**Logical AND:** `True` only if both the operands are `True`
`or`	a **or** b	**Logical OR:** `True` if at least one of the operands is `True`
`not`	**not** a	**Logical NOT:** `True` if the operand is `False` and vice-versa.

Logical Operators

```
# Initial values
number1 = 10
number2 = 5
number3 = 15

# Using 'and' operator
print("Is number1 greater than number2 and number1 less than number3? ", (number1 > number2) and (number1 < number3))
# True (because both conditions are True)

# Using 'or' operator
print("Is number1 greater than number2 or number1 greater than number3? ", (number1 > number2) or (number1 > number3))
# True (because one condition is True)

# Using 'not' operator
print("Is it not true that number1 is greater than number3? ", not (number1 > number3))
# True (because number1 is not greater than number3)
```

Bitwise operators

- Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.
- In the table below: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x 0010 1000)

Bitwise operators

```
# Initial values
number1 = 10  # In binary: 1010
number2 = 4   # In binary: 0100
# Bitwise AND
result = number1 & number2
print(f"Bitwise AND of {number1} & {number2} = {result}")  # 1010 & 0100 = 0000 (result = 0)
# Bitwise OR
result = number1 | number2
print(f"Bitwise OR of {number1} | {number2} = {result}")   # 1010 | 0100 = 1110 (result = 14)
# Bitwise XOR
result = number1 ^ number2
print(f"Bitwise XOR of {number1} ^ {number2} = {result}")  # 1010 ^ 0100 = 1110 (result = 14)
# Bitwise NOT
result = ~number1
print(f"Bitwise NOT of ~{number1} = {result}")             # ~1010 = -1011 (result = -11)
# Bitwise Left Shift
result = number1 << 2
print(f"Left Shift of {number1} << 2 = {result}")          # 1010 << 2 = 101000 (result = 40)
# Bitwise Right Shift
result = number1 >> 2
print(f"Right Shift of {number1} >> 2 = {result}")         # 1010 >> 2 = 0010 (result = 2)
```

Bitwise operators

```
# Initial values
number1 = 10  # In binary: 1010
number2 = 4   # In binary: 0100
# Bitwise AND
result = number1 & number2
print(f"Bitwise AND of {number1} & {number2} = {result}")  # 1010 & 0100 = 0000 (result = 0)
# Bitwise OR
result = number1 | number2
print(f"Bitwise OR of {number1} | {number2} = {result}")   # 1010 | 0100 = 1110 (result = 14)
# Bitwise XOR
result = number1 ^ number2
print(f"Bitwise XOR of {number1} ^ {number2} = {result}")  # 1010 ^ 0100 = 1110 (result = 14)
# Bitwise NOT
result = ~number1
print(f"Bitwise NOT of ~{number1} = {result}")             # ~1010 = -1011 (result = -11)
# Bitwise Left Shift
result = number1 << 2
print(f"Left Shift of {number1} << 2 = {result}")          # 1010 << 2 = 101000 (result = 40)
# Bitwise Right Shift
result = number1 >> 2
print(f"Right Shift of {number1} >> 2 = {result}")         # 1010 >> 2 = 0010 (result = 2)
```

```
Bitwise AND of 10 & 4 = 0
Bitwise OR of 10 | 4 = 14
Bitwise XOR of 10 ^ 4 = 14
Bitwise NOT of ~10 = -11
Left Shift of 10 << 2 = 40
Right Shift of 10 >> 2 = 2
```

Membership operators

- In Python, in and not in are the membership operators. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

Operator	Meaning	Example
`in`	`True` if value/variable is **found** in the sequence	`5 in x`
`not in`	`True` if value/variable is **not found** in the sequence	`5 not in x`

Membership operators

- In Python, in and not in are the membership operators. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

Operator	Meaning	Example
`in`	`True` if value/variable is **found** in the sequence	`5 in x`
`not in`	`True` if value/variable is **not found** in the sequence	`5 not in x`

```
# Example with a string
text = "Hello, welcome to Python programming!"
# Using 'in' operator with string
print("Is 'Python' in the text? ", 'Python' in text)  # True
# Using 'not in' operator with string
print("Is 'Java' not in the text? ", 'Java' not in text)  # True
```

```
Is 'Python' in the text?  True
Is 'Java' not in the text?  True
```

Identity operators

- In Python, **`is`** and **`is not`** are used to check if two values are located at the same memory location.
- It's important to note that having two variables with equal values doesn't necessarily mean they are identical.

Operator	Meaning	Example
`is`	`True` if the operands are identical (refer to the same object)	`x is True`
`is not`	`True` if the operands are not identical (do not refer to the same object)	`x is not True`

Identity operators

```
str1 = "hello"
str2 = "hello"
str3 = "world"

# Using 'is' operator
print("Does str1 is str2? ", str1 is str2)  # True (both refer to the
same object)
print("Does str1 is str3? ", str1 is str3)  # False (different objects)
```

- `str1` and `str2`: Both are assigned the value **"hello".** In Python, strings that **are identical and immutable are often interned**, meaning **they reference the same memory location**. Therefore, `str1` is `str2` returns True.
- `str1` and `str3`: `str3` is assigned the value "world", which is different from "hello". Thus, `str1` and `str3` reference different objects in memory, so `str1` is `str3` returns False.

Identity operators

```
str1 = "hello"
str2 = "hello"
str3 = "world"

# Using 'is not' operator
print("Does str1 is not str2? ", str1 is not str2)  # False (both refer to the same object)
print("Does str1 is not str3? ", str1 is not str3)  # True (different objects)
```

- `str1` is not `str2` returns False because `str1` and `str2` reference the same object.
- `str1` is not `str3` returns True because `str1` and `str3` reference different objects.

Precedence of Python Operators

- To evaluate expressions, there is a rule of precedence in Python that guides the order in which these operations are carried out

```
>>> 10 - 4 * 2
2
```

- Why does the result show 2 instead of 12?

Precedence of Python Operators

Precedence group	Operators	Associativity
Parenthesis	`()`	L → R
Exponentiation	`**`	R → L
Unary plus, Unary minus, One's complement	`+, -, ~`	R → L
Multiplication, Division, Floor division, Modulus	`*, /, //, %`	L → R
Addition, Subtraction	`+, -`	L → R
Bitwise shift operators	`<<, >>`	L → R
Bitwise AND	`&`	L → R
Bitwise XOR	∧	L → R
Bitwise OR	|	L → R
Comparisons, Identity and Membership operators	`==, !=, <, <=, >=, > is, is not, in, not in`	L → R
Logical NOT	`not`	R → L
Logical AND	`and`	L → R
Logical OR	`or`	L → R
Assignment operators	`=, +=, -=, *= /=, //=, %=`	R → L

Precedence of Python Operators

- Evaluate the following expressions

```
a) result = 2 + 3 * 4
b) result = (2 + 3) * 4
c) result = 10 / 2 + 5
d) result = 10 // 3 * 3
e) result = 2 ** 3 ** 2
```

The math module

The math module

- The math module in Python provides a wide range of mathematical functions and constants.
- It’s a built-in module, so you don’t need to install anything extra to use it.
- To use a function or a constant of the math module, you need to do two things:
  - import the module
  - access the function or the constant by prefixing its name with ”math.” (math followed by a dot)

The math module

- Trigonometric Functions:
  - `math.sin(x)`: Returns the sine of x (x is in radians).
  - `math.cos(x)`: Returns the cosine of x (x is in radians).
  - `math.tan(x)`: Returns the tangent of x (x is in radians).
- Exponential and Logarithmic Functions:
  - `math.exp(x)`: Returns (e^x).
  - `math.log(x, base)`: Returns the logarithm of x to the given base. If the base is not specified, it returns the natural logarithm.
  - `math.log10(x)`: Returns the base-10 logarithm of x.
- Power and Root Functions:
  - `math.sqrt(x)`: Returns the square root of x.
  - `math.pow(x, y)`: Returns (x^y).

The math module

- Rounding Functions:
  - `math.ceil(x)`: Returns the smallest integer greater than or equal to x.
  - `math.floor(x)`: Returns the largest integer less than or equal to x.
- Other Useful Functions:
  - `math.factorial(x)`: Returns the factorial of x.
  - `math.gcd(a, b)`: Returns the greatest common divisor of a and b.
  - `math.fabs(x)`: Returns the absolute value of x.
- Constants
  - `math.pi`: The mathematical constant π (approximately 3.14159).
  - `math.e`: The mathematical constant e (approximately 2.71828).

The math module - Examples

- Calculating the square root of a number.

```
# This program will show the calculation of square root using the math module
# importing the math module
import math
print(math.sqrt( 9 ))
```

- Calculating the factorial of a number

```
import math  # Import the math module to use mathematical functions

n = int(input())  # Take an integer input from the user and store it in variable 'n'

# Calculate the factorial of 'n' using the factorial function from the math module
print(math.factorial(n))  # Print the result
```

The math module - Examples

- Python program that calculates the circumference and area of a circle

```
import math  # Import the math module to access mathematical functions and constants

# Input: radius of the circle
radius = float(input("Enter the radius of the circle: "))

# Calculate the circumference of the circle
circumference = 2 * math.pi * radius

# Calculate the area of the circle
area = math.pi * radius ** 2

# Output the results
print(f"The circumference of the circle is: {circumference}")
print(f"The area of the circle is: {area}")
```

References

- https://peps.python.org/pep-0008/
- Algorithmic Thinking with Python – Ajeesh Ramanujan, Narasimhan T
- https://www.javatpoint.com/python-math-module
- https://www.programiz.com/python-programming/operators
- Microsoft Copilot in Bing
- https://chatgpt.com/

ST.JOSEPH'S
COLLEGE OF ENGINEERING
AND TECHNOLOGY,
- PALAI -
AUTONOMOUS

Thank You

Prof. Sarju S
Department of Computer Science and Engineering
St. Joseph's College of Engineering and Technology, Palai (Autonomous)
sarju.s@sjcetpalai.ac.in