# Python for Data Science, AI & Development

## Module 1: Python Basics

# Table of Contents


Introduction....4

What is Python used for...? ....4

Introduction to Jupyter ....5

Why Choose Jupyter? ....5

Python Basics ....6

1. Syntax....6

Example of a simple program: ....6

2. Variables and Data Types ....6

3. Control Structures ....6

Conditional Statements (if-else / if-elif-else) ....7

Loops ....7

4. Functions ....7

5. Libraries and Modules....7

Python in Data Science ....8

Python in AI....8

Python in Development ....8

Why Python for Data Science, AI & Development? ....8

Python History and Development ....9

Key Differences Between Python 2 and Python 3....9

Modules and pip in Python! ....10

The pip command....11

Our First Program with print function....12

Other Parameters of Print Statement....13

Quick Quiz ....14

Comments and Escape sequence ....14

Python Comments....14

Escape Sequences in Python: ....15

Errors in Python ....15

How to Handle Errors in Python....17

Tips for Error Handling ..... 19
Variables and Data Types In Python ..... 20
Variables in Python ..... 20
Data Types In Python ..... 21
Common Data Types: ..... 21
Summary ..... 22
Working with Expressions and Variable In Python ..... 23
Variables ..... 23
Expressions ..... 23
Common Operators on Variables ..... 23
Combining Variables and Expressions ..... 26

# Introduction

Python is a **high-level programming language** known for its **simplicity, versatility, and readability**, which makes it an ideal choice for beginners and professionals alike. It is widely used in **data science, artificial intelligence (AI), and software development**, due to its extensive libraries, frameworks, and supportive community.

## What is Python used for...?

- Python is used in Data Visualization to create plots and graphical representations.
- Python helps in Data Analytics to analyze and understand raw data for insights and trends.
- It is used in AI and Machine Learning to simulate human behavior and to learn from past data without hard coding.
- It is used to create web applications.
- It can be used to handle databases.
- It is used in business and accounting to perform complex mathematical operations along with quantitative and qualitative analysis.

# Introduction to Jupyter

Jupyter is a free web application that ***allows you to create and share documents combining live code, equations, visualizations, and text***. It ***provides an interactive environment supporting multiple programming languages, including Python***, R, and Julia, but is especially effective with Python. ***Jupyter focuses on "notebooks,"*** which are documents that mix code, charts, explanations, equations, and multimedia. These notebooks make it easy to work on, share, and collaborate on computational projects.

## Why Choose Jupyter?

Jupyter is popular because it's flexible and easy to use, catering to both beginners and experts. Whether you're new to coding or experienced in data science, Jupyter offers a simple platform for writing, testing, and sharing code. Its interactive interface lets you explore data, try out algorithms, and visualize results all in one place.

**Key Features of Jupyter**

- **Interactive Coding**: Write and run code in sections (cells) and see results instantly, making it great for experimenting and refining your work.
- **Supports Many Languages**: Originally for Python, now supports R, Julia, and more, thanks to its kernel system.
- **Rich Output**: You can include interactive graphs, charts, images, and even videos within Jupyter notebooks, which makes data visualization and storytelling much easier.
- **Data Science Integration**: Works with libraries like NumPy, Pandas, Matplotlib, scikit-learn, TensorFlow, and PyTorch for data analysis and machine learning.
- **Collaboration and Sharing**: You can easily share Jupyter notebooks via email, GitHub, or Jupyter Notebook Viewer. This promotes collaboration, peer review, and sharing of ideas, making it perfect for group projects or interdisciplinary work.

# Python Basics

## 1. Syntax

Python's syntax is **clean, readable**, and uses indentation to define blocks of code rather than curly braces {} like many other languages. This makes Python very beginner-friendly.

### Example of a simple program:

```python
# This is a comment
print("Hello, World!")  # This prints text to the console
```

- No need for semicolons (;) at the end of lines.
- Indentation (spaces or tabs) defines code blocks

## 2. Variables and Data Types

Variables are used to store data in Python. You don't need to declare the type of variable; Python infers it automatically.

**Common Data Types**

| Data Type | Example | Description |
|---|---|---|
| String | `"Hello"` | Text data |
| Integer | `25` | Whole numbers |
| Float | `3.14` | Decimal numbers |
| Boolean | `True/False` | Logical values |
| List | `[1, 2, 3]` | Collection of items (mutable) |
| Tuple | `(1, 2, 3)` | Immutable collection |
| Dictionary | `{"key": "value"}` | Key-value pairs |

## 3. Control Structures

Python uses control structures to manage the flow of a program. It uses indentation (not braces {}) to define codes.

### Conditional Statements (if-else / if-elif-else)

```
if age > 18:
    print("Adult")
elif age == 18:
    print("Just turned adult!")
else:
    print("Minor")
```

### Loops

- **For Loop**: Iterate over a sequence.

```
for hobby in hobbies:
    print(hobby)
```

- **While Loop**: Repeat until a condition is false.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

## 4. Functions

Functions help you organize code into reusable blocks. You can create a function on your own and there are may built-in functions like len(), type(), and range() in python.

```
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))
```

## 5. Libraries and Modules

Python has an extensive standard library, and you can install additional libraries using **pip** (Python Package Index).

### Importing Libraries

```
import math
print(math.sqrt(16))  # Output: 4.0
```

## Python in Data Science

Python is extremely popular in data science because it provides powerful tools for working with data. Let's explore:

1. **Data Manipulation**: The **Pandas** library is essential for data manipulation and analysis.
2. **Data Visualization**: Python offers powerful tools like **Matplotlib** and **Seaborn** for creating data visualizations.
3. **Machine Learning**: **Scikit-learn** simplifies tasks such as regression, classification, and clustering.

## Python in AI

Python is a leading language for artificial intelligence because of its flexibility and strong libraries.

1. **Deep Learning**: Libraries like **TensorFlow** and **PyTorch** simplify building and training neural networks.
2. **Natural Language Processing (NLP)**: Tools like **spaCy** and **NLTK** assist in analyzing and processing text data.
3. **Computer Vision**: Python's **OpenCV** library is ideal for handling image and video processing tasks.

## Python in Development

Python is used extensively for creating applications:

1. **Web Development**: Frameworks like **Django** and **Flask** are widely used for creating dynamic web applications.
2. **Game Development**: **Pygame** is perfect for building simple 2D games.
3. **Automation**: Python excels at automating repetitive tasks, including file handling, data scraping, and testing.

## Why Python for Data Science, AI & Development?

- **Ease of Use**: Python's simple syntax reduces the learning curve.
- **Community Support**: A vast community helps with troubleshooting and learning.

- **Rich Libraries**: Libraries like NumPy, Pandas, TensorFlow, and Flask cater to specific needs.
- **Cross-Platform**: Works on Windows, macOS, and Linux.
- **Versatility**: Works across web development, AI, and data science.

# Python History and Development

- **Python 2** and **Python 3** are two major versions of the Python programming language. Python 3 was introduced to address limitations and inconsistencies in Python 2, leading to improved functionality and modernized syntax. **Python 2** was released in 2000. It's not updated anymore and reached the "end of life" in 2020, meaning no more official support or fixes.
- **Python 3** was released in 2008. It's the current version and actively maintained with new features and updates.

## Key Differences Between Python 2 and Python 3

1. **Printing text is different:**
   - **In Python 2**, you write print "Hello" without parentheses.
   - **In Python 3**, you must use parentheses: print("Hello").
2. **Dealing with text (Unicode vs. ASCII):**
   - **Python 2** uses ASCII by default, which is good for English but struggles with other languages. So, Unicode handling requires a special prefix (u).
   - **Python 3** uses Unicode by default, which handles all languages and special characters better. (Think emojis! 😊 )
3. **Division behaves differently:**
   - **In Python 2**, dividing integers (e.g., 5/2) gives a whole number: 2.
     - You'd need to write 5.0 / 2.0 to get 2.5.
   - **In Python 3**, dividing integers (5/2) gives a float result: 2.5 by default.
4. **Libraries are mostly Python 3-friendly now:**
   - Many modern libraries and tools only support **Python 3**.
   - If you use Python 2, you might miss out on new technology.

5. **Input function is clearer in Python 3:**
    - **In Python 2**, you use raw_input() to get input as a string, but input() evaluates it like code (can be risky).
    - **In Python 3**, you just use input(), which always takes input as a string (simpler and safer).
6. **Syntax Improvements**
    - Python 3 introduces many new features such as type hints, f-strings for formatted strings, and async programming.

## Modules and pip in Python!

Module is like a code library which can be used to borrow code written by somebody else in our python program. There are two types of modules in python:

- Built in Modules - These modules are ready to import and use and ships with the python interpreter. there is no need to install such modules explicitly.
- External Modules - These modules are imported from a third-party file or can be installed using a package manager like pip or conda. Since this code is written by someone else, we can install different versions of a same module with time.

A module is like a container that holds related pieces of code, which can include functions, classes, and variables. You can think of a module as a toolbox. Each module has a specific purpose, like a toolbox containing tools for a particular task.

**Real-life Example:**

Imagine you have a kitchen. The kitchen has different drawers for different items: one drawer for spoons, another for knives, and another for forks. These drawers organize your tools in a way that makes it easier to find the one you need.

Similarly, in Python, a module is like a drawer that contains tools (functions, classes, etc.) you might need. You can import a module into your program to use its tools.

For example:

python Copy code

```python
import math   # Importing the math module

result = math.sqrt(16)  # Using the sqrt function from the math module
print(result)  # Output: 4.0
```

Here, math is the module, and sqrt is one of the tools (functions) it contains.

## The pip command

**pip** is a tool used to **install** and manage Python packages (which are like modules). It's like going to a **store** to buy new tools or equipment. When you need a tool that isn't in your current toolbox (module), you can use pip to go to the store and get it.

**Real-life Example:**

Imagine you need a **specialized tool** that you don't have in your toolbox, like a **wrench** for a very specific type of bolt. You can go to the **store** (the internet, in this case) and buy the wrench (using pip), bringing it back to your toolbox to use whenever you need it.

In Python, **pip** helps you get packages (modules) from the **Python Package Index (PyPI)** so you can use them in your program.

It can be used as a package manager pip to install a python module. Lets install a module called pandas using the following command

pip install pandas

**Using a module in Python (Usage)**

We use the import syntax to import a module in Python. Here is an example code:

```
import pandas

# Read and work with a file named 'words.csv'
df = pandas.read_csv('words.csv')
print(df) # This will display first few rows from the words.csv file
```

**Summary:**

- **Module**: Like a **toolbox** that contains different tools (functions, classes) to help you perform tasks.
- **pip**: Like a **store** where you can go to **buy** (install) new tools (packages/modules) that you need.

So, when you use pip, you're essentially going to the store to get new tools that you can later use in your Python programs by importing them as modules!

## Our First Program with print function

Today we will write our first ever python program from scratch. It will consist of a bunch of print statements. print can be used to print something on the console in python.

***Note: REPL: - Read Evaluate Print Loop***

In order to print in python, a valid object should be given. We use print function in python to pass a value (string, number, etc.) directly to it.

```python
print("Hello, World!")  # Output: Hello, World!
```

print()

We use ***single ‘ ‘ or double “ “*** quotation in python to ***print string*** however number and variable doesn’t need them. We can also print multiple values.

```python
name = "Bob"
age = 30
print("Name:", name, "Age:", age)  # Output: Name: Bob Age: 30
```

We can ***also format Output with f-strings*** in more readable ways using python. Note we need to ***use curly brackets {}*** for each variable.

```python
name = "Charlie"
age = 22
print(f"Name: {name}, Age: {age}")  # Output: Name: Charlie, Age: 22
```

Here, `f"..."` is used to embed the values of `name` and `age` into the string directly.

### Controlling Line Breaks

By default, print() adds a newline (a **line break**) after each call. This means that each print() statement will show its output on a new line.

```python
print("Hello")
print("World")
```

**Output:**

```
Hello
World
```

If you don't want a new line after the output, you can ***use the end argument*** to control what happens after the output. By default, ***end="\n***" adds a new line, but you can change this behavior.

```python
print("Hello", end=" ")
print("World")  # Output: Hello World
```

## Other Parameters of Print Statement

1. object(s): Any object, and as many as you like. Will be converted to string before printed
2. sep = 'separator': Specify how to separate the objects, if there is more than one. Default is ' '
3. end='end': Specify what to print at the end. Default is '\n' (line feed)
4. file: An object with a write method. Default is sys.stdout

## Quick Quiz

1. **Write a program to print a poem in Python. Choose the poem of your choice and publish your repl.**

```python
print("I opened the door, and there it was,\n"
      "A world of code, with no big pause.\n"
      "Lines of logic, clean and bright,\n"
      "Python’s power, pure delight\n"
      "\n"
      "From loops to lists, functions to flow,\n"
      "Each lesson learned helps me grow.\n"
      "I write my script, the code runs free,\n"
      "A language simple, as it should be.\n\n")

print('Errors come, but I don’t despair,')
print('I debug and fix with steady care.\nWith Python, I can dream and build,\nCreating wonders, with mind fulfilled.')
```

```
I opened the door, and there it was,
A world of code, with no big pause.
Lines of logic, clean and bright,
Python’s power, pure delight

From loops to lists, functions to flow,
Each lesson learned helps me grow.
I write my script, the code runs free,
A language simple, as it should be.


Errors come, but I don’t despair,
I debug and fix with steady care.
With Python, I can dream and build,
Creating wonders, with mind fulfilled.
```

# Comments and Escape sequence

## Python Comments

A comment is a part of the coding file that the programmer does not want to execute, rather the programmer uses it to either explain a block of code or to avoid the execution of a specific part of code while testing.

- # is used for disabling single line in python
- """ or ''' is used for disabling block of codes and it needs to have same closing quotes
- Or we can select lines of codes and press ctrl + / to disables codes

python Copy code

```python
# This is a comment. It explains the code below.
print("Hello, world!")  # Prints a greeting message.
```

## Escape Sequences in Python:

Escape sequences are special characters in a string that perform actions like starting a new line, adding a tab space, or using quotation marks within a string.

Escape sequences start with a backslash (\).

Common Escape Sequences:

- \n - New Line
- \t - Tab Space
- \' - Single Quote
- \" - Double Quote

```
print("Hello,\nWelcome to Python!")  # \n starts a new line
print("Item\tPrice")                 # \t adds a tab space
print("She said, \"Python is fun!\"")  # \" allows double quotes inside a string
```

**Output**

Item	Price
She said, "Python is fun!"
p is greater than 5.

## Errors in Python

Errors occur when the Python interpreter encounters something it cannot process, preventing the program from running or completing its task. Understanding errors helps debug code effectively and ensures smoother program execution.

**Types of Errors in Python**

1. **Syntax Errors**:
    - These occur when Python's syntax rules are broken. These are detected during parsing before execution.

      python  Copy code

      ```
      print("Hello World"  # Missing closing parenthesis
      ```

    - **Error Message**: `SyntaxError: unexpected EOF while parsing`

2. **Indentation Errors**:
   - Python relies on indentation to define code blocks. Misaligned or missing indentation causes this error.

```python
def greet():
print("Hello")  # Indentation missing for the print statement
```

Error Message: `IndentationError: expected an indented block`

3. **Name Errors**:
   - Raised when attempting to access a variable, function, or module that is not defined.

```python
print(name)  # 'name' is not defined anywhere in the code
```

Error Message: `NameError: name 'name' is not defined`

4. **Type Errors:**
   - Occur when an operation is applied to an object of an inappropriate type.

```python
print(5 + "5")  # Cannot add integer and string
```

Error Message: `TypeError: unsupported operand type(s) for +: 'int' and 'str'`

5. **Value Errors**:
   - Occur when a function receives the right type of argument but an invalid value.

```python
int("abc")  # Cannot convert non-numeric string to integer
```

6. **Index Errors**:
   - Raised when trying to access an index that doesn't exist in a list, string, or tuple.

```python
my_list = [1, 2, 3]
print(my_list[5])  # Index 5 is out of range
```

7. **Key Errors**:
   - Occur when trying to access a dictionary key that doesn't exist.

```python
my_dict = {"name": "Alice"}
print(my_dict["age"])  # Key 'age' is not in the dictionary
```

Error Message: `KeyError: 'age'`

8. **ZeroDivisionError**:
   - Raised when dividing by zero, which is mathematically undefined.

```python
print(10 / 0)  # Cannot divide by zero
```

Error Message: `ZeroDivisionError: division by zero`

9. **Attribute Errors**:
   - Raised when an invalid attribute is accessed for an object.

```python
number = 10
number.append(5)  # Integers don't have an 'append' method
```

Error Message: `AttributeError: 'int' object has no attribute 'append'`

## How to Handle Errors in Python

Python provides the **try-except** mechanism to gracefully handle errors and prevent program crashes.

### Basic Error Handling Syntax

```python
try:
    # Code that might raise an error
except SpecificError:
    # Code to handle the error
finally:
    # Code that always executes (optional)
```

1. **Handle Division by Zero:**

```python
try:
    print(10 / 0)
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

2. **Handle Multiple Errors**:

```python
try:
    x = int("abc")
    y = 5 / 0
except ValueError:
    print("Invalid value!")
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

3. **Catch All Errors**: Exception Handling (Not recommended for debugging purposes):

```python
try:
    print(5 / 0)
except Exception as e:
    print(f"An error occurred: {e}")
```

4. **Using finally** (Code that always runs):

```python
try:
    file = open("data.txt", "r")
except FileNotFoundError:
    print("File not found!")
finally:
    print("Execution complete!")
```

5. **else Block in Try-Except:**

```python
try:
    result = 10 / 2
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print(f"Result: {result}")
```

6. **Raising Exceptions**

```python
try:
    age = -5
    if age < 0:
        raise ValueError("Age cannot be negative!")
except ValueError as e:
    print(e)
```

## Tips for Error Handling

- Be specific about the error type in except to avoid catching unrelated errors.
- Use finally for cleanup (e.g., closing files or releasing resources).
- Test your code thoroughly to minimize errors in the first place.
- Avoid overusing try-except

With proper error handling, your programs become more robust and user-friendly

# Variables and Data Types In Python

## Variables in Python

A **variable** is like a container that stores data in a computer's memory. Imagine it as a labeled box where you can keep something, and you can use the label to find or change what's inside.

**Key Points:**

1. Variables are used to store values (data).
2. You can assign values to variables using the = operator.
3. Variable names should be meaningful and follow certain rules:
    - Must start with a letter or an underscore (_).
    - Can contain letters, numbers, or underscores (but no spaces or special characters) but cannot use Python keywords (like if, while, etc.).

In Python it's as easy as writing:

- a = 1
- b = True
- c = "Harry"
- d = None

These are four variables of different data types.

**How Variables Work**

When you create a variable:

- Python **stores the value in memory**.
- The variable acts as a **reference** to that value.

```
name = "Alice"  # "name" is the variable storing "Alice"
age = 25        # "age" is the variable storing the number 25
```

**How It Works:**

- The computer puts "Alice" in a box in memory and labels the box name.
- It puts 25 in another box and labels it age.

**Note:** Variables can be updated or changed anytime.

## Data Types In Python

Data types tell Python what kind of data you're working with. Just like a box can hold different things (books, clothes, toys), variables can store different types of data. Python automatically assigns the data type based on the value you give to a variable. This is called **dynamic typing**.

### Common Data Types:

1. **Numeric Types**:
   - **int**: Whole numbers (e.g., 1, -5, 100).
   - **float**: Numbers with decimal points (e.g., 3.14, -2.5).
   - **complex**: Numbers with a real and imaginary part (e.g., 3+4j).
2. **String (str)**: Text or characters surrounded by quotes (' ' or " ").
3. **Boolean (bool)**: Represents True or False.

```python
is_light_on = True  # Boolean type
is_raining = False
print("Is the light on?", is_light_on)
print("Is it raining?", is_raining)
```

4. **List**: An ordered collection of **data** with elements separated by a comma and enclosed within square brackets [ ]. Lists are mutable and can be modified after creation.

```python
fruits = ["apple", "banana", "cherry"]
print(fruits[1])  # Output: banana
```

5. **Tuple:** A tuple is a data type in Python used to store multiple items in a single, ordered, and immutable (unchangeable) collection. Tuples are similar to lists, but there's a key difference: *you cannot change, add, or remove elements in a tuple once it's created*.

```python
coordinates = (10, 20)  # A tuple storing x and y coordinates
print(coordinates)  # Output: (10, 20)
```

6. **Dictionary: dict:** A dictionary in Python is an unordered collection of data containing key-value pairs. Each key is unique, and it is used to access its corresponding value. Dictionaries are similar to real-life dictionaries, where you look up a word (key) to find its meaning (value). They are mutable in the sense that we can change, add or remove value or items however keys must be immutable and must be unique.

```python
# Example of a dictionary
student = {
    "name": "Alice",
    "age": 25,
    "grade": "A"
}
```

7. **Sets:** A set in Python is an unordered collection of unique elements. Sets are used when you need to store multiple items, ensure there are no duplicates, and perform mathematical operations like unions, intersections, and differences. They are unordered, unique, mutable but immutable elements.
   - Sets are created using curly braces {} or the set() function.
   - You **cannot access elements by index** because sets are unordered. However, you can loop through a set.

```python
colors = {"red", "green", "blue"}
```

## Summary

- A **variable** is a container to store data.
- **Data types** define the type of data a variable can hold.
- Python is dynamically typed, so you don't need to explicitly mention the data type.
- Use meaningful variable names to make your code easier to understand.

# Working with Expressions and Variable in Python

## Variables

Variables are names that store data values in your program. Think of them as containers where you can put and retrieve data. Variables can hold any type of data and can be reassigned.

In Python, you don't need to specify the type of a variable explicitly. Just assign a value to a name:

```python
x = 10          # Integer
name = "Alice"  # String
pi = 3.14       # Float
is_active = True # Boolean
```

## Expressions

An expression in Python is a combination of values, variables, operators, and function calls that produce a result. They are evaluated to produce a single value.

```python
# Arithmetic Expression
result = 5 + 3 * 2  # Evaluates to 11

# Using Variables in Expressions
x = 10
y = 5
total = x + y  # Evaluates to 15

# Logical Expression
is_valid = (x > y) and (x < 20)  # Evaluates to True
```

## Common Operators on Variables

1. **Arithmetic Operators**:
    - +: Addition
    - -: Subtraction
    - *: Multiplication
    - /: Division
    - //: Floor Division (removes the decimal part)
    - %: Modulus (remainder)
    - **: Exponentiation (power)

```python
x = 15
y = 4

print(x + y)  # Output: 19
print(x - y)  # Output: 11
print(x * y)  # Output: 60
print(x / y)  # Output: 3.75
print(x // y) # Output: 3 (floor division)
print(x % y)  # Output: 3 (remainder)
print(x ** y) # Output: 50625 (15 to the power of 4)
```

2. **Comparison Operators**:

   Used to compare two values:

   - ==: Equal
   - !=: Not equal
   - >: Greater than
   - <: Less than
   - >=: Greater than or equal to
   - <=: Less than or equal to

```python
x = 10
y = 5

print(x == y)  # Output: False (10 is not equal to 5)
print(x != y)  # Output: True (10 is not equal to 5)
print(x > y)   # Output: True (10 is greater than 5)
print(x <= y)  # Output: False (10 is not less than or equal to 5)
```

3. **Logical Operators**:
   Used for combining conditions:
   - and: True if both conditions are true
   - or: True if at least one condition is true
   - not: Reverses the condition

```python
x = 10
y = 5

print(x > 5 and y < 10)  # Output: True (both conditions are true)
print(x > 15 or y < 10)  # Output: True (at least one condition is true)
print(not(x > 15))       # Output: True (x > 15 is false, but `not` makes it true)
```

4. **String Operations**

   Python allows you to manipulate strings easily like concatenate, repeat strings or fint string length.

5. **Bitwise Operators**
   Operate on binary numbers.

| Operator | Description | Example |
|---|---|---|
| `&` | Bitwise AND | `5 & 3` → `1` |
| ` | ` | Bitwise OR |
| `^` | Bitwise XOR | `5 ^ 3` → `6` |
| `~` | Bitwise NOT | `~5` → `-6` |
| `<<` | Left shift | `5 << 1` → `10` |
| `>>` | Right shift | `5 >> 1` → `2` |

6. **Assignment Operators**
   Used to assign values to variables.

| Operator | Description | Example |
|---|---|---|
| `=` | Assign | `x = 5` |
| `+=` | Add and assign | `x += 3` → `x = x + 3` |
| `-=` | Subtract and assign | `x -= 2` → `x = x - 2` |
| `*=` | Multiply and assign | `x *= 4` → `x = x * 4` |
| `/=` | Divide and assign | `x /= 2` → `x = x / 2` |
| `%=` | Modulus and assign | `x %= 3` → `x = x % 3` |

7. **Membership Operators**
   Used to check if a value is in a sequence.

| Operator | Description | Example |
|---|---|---|
| `in` | Checks membership | `'a' in 'apple'` → `True` |
| `not in` | Checks non-membership | `'x' not in 'apple'` → `True` |

**Real-Life Example**

Checking if an item is in a shopping list:

python Copy code

```python
shopping_list = ['milk', 'bread', 'eggs']
if 'milk' in shopping_list:
    print("Buy milk!")
else:
    print("No need for milk.")
```

8. **Identity Operators**
   Used to check if two variables are the same object.

| Operator | Description | Example |
|---|---|---|
| `is` | Same identity | `x is y` |
| `is not` | Not the same identity | `x is not y` |

**Real-Life Example**

Verifying if two users are the same:

```python
user1 = "Alice"
user2 = "Alice"

if user1 is user2:
    print("Same user.")
else:
    print("Different users.")
```

## Combining Variables and Expressions

Variables can store results of expressions for reuse.

```python
base_salary = 50000
bonus = 0.1 * base_salary
total_salary = base_salary + bonus

print(f"Total Salary: {total_salary}")  # Outputs: Total Salary: 55000.0
```

## Type Casting in Python

### What is Type Casting in Python?

Type casting in Python refers to the process of converting a value from one data type to another. This is useful when you need a value in a specific format to perform certain operations. Python provides built-in functions for type casting.

### Types of Type Casting

1. **Implicit Type Casting**:
    - Done automatically by Python.
    - Converts a smaller data type to a larger data type to avoid data loss.
    - Example: Converting an integer to a float during arithmetic operations.

```python
x = 5     # Integer
y = 2.5   # Float
z = x + y   # Integer + Float
print(z)  # Output: 7.5 (Float)
```

2. **Explicit Type Casting**:
    - Done manually by the programmer.
    - Uses type conversion functions like int(), float(), str(), etc.

```python
shopping_list = ["eggs", "milk", "bread"]
shopping_tuple = tuple(shopping_list)
print(shopping_tuple)  # Output: ('eggs', 'milk', 'bread')
```

```python
age = 25  # Integer
message = "I am " + str(age) + " years old."
print(message)
```

- `str(age)` converts the integer `25` to a string so it can be concatenated with other strings.