python™

PYTHON NOTES

- **What is Python?**
  Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used in web development, data analysis, machine learning, automation, and more.
- **Why Learn Python?**
  - Beginner-friendly syntax
  - Extensive libraries and frameworks
  - Strong community support
  - Versatile for various applications

Setting Up Python on Linux and Windows: Step-by-Step Guide

1. Install Python

On Windows

- **Download Python**
  - Visit python.org and download the Windows installer.
  - Choose the appropriate version (32-bit or 64-bit) based on your system.
- **Install Python**
  - Run the installer.
  - Check the box **"Add Python to PATH"** to make Python accessible from the Command Prompt.
  - Choose **Customize Installation** for optional features like pip, IDLE, and development tools.
  - Complete the installation process.
- **Verify Installation**

    - Open Command Prompt.

Run:

```
python --version

pip --version
```

On Linux

Update System Packages

```
sudo apt update && sudo apt upgrade -y  # For Debian/Ubuntu
```

- **Install Python**

For Debian/Ubuntu:

```
sudo apt install python3 python3-pip -y
```

For Red Hat/CentOS:

```
sudo yum install python3 python3-pip -y
```

Verify Installation

```
python3 --version

pip3 --version
```

---

2. Choose an Editor

Recommended Editors for Both Linux and Windows

- **VS Code (Visual Studio Code)**
    - Download from code.visualstudio.com.

    - Install the **Python Extension** for debugging, syntax highlighting, and more.

Command to install on Linux (Debian/Ubuntu):

```
sudo apt install code
```

- **PyCharm**
    - Download from jetbrains.com/pycharm.
    - Offers a free Community Edition.

- **Jupyter Notebook**

Install via pip:

```
pip install notebook
```

Launch:

```
jupyter notebook
```

---

3. Verify Python and Pip Installation

On Windows

Open Command Prompt and run:

```
python --version

pip --version
```

On Linux

Open a terminal and run:

```
python3 --version
```

```
pip3 --version
```

---

4. Set Up Virtual Environments (Optional but Recommended)

On Windows

Create a virtual environment:

```
python -m venv myenv
```

Activate the environment:

```
myenv\Scripts\activate
```

Deactivate with:

```
deactivate
```

On Linux

Create a virtual environment:

```
python3 -m venv myenv
```

Activate the environment:

```
source myenv/bin/activate
```

Deactivate with:

```
deactivate
```

---

5. Install Essential Libraries

Use pip to install Python libraries.

Example Commands

Install libraries:

```
pip install numpy pandas matplotlib
```

Upgrade pip:

```
python -m pip install --upgrade pip  # Windows
```

```
python3 -m pip install --upgrade pip  # Linux
```

---

6. Additional Tips

- **Linux Users**
  - Use a package manager like apt or yum to install Python dependencies.

Install build tools if needed:

```
sudo apt install build-essential -y
```

- **Windows Users**
  - Use PowerShell or Command Prompt for Python commands.
  - Use Windows Subsystem for Linux (WSL) for a Linux-like development environment.

Script Mode: Save a file as script.py and run it with:

```
python script.py
```

Basic Syntax

Hello World

```
print("Hello, World!")
```

Python Comments

Single-line Comments
Comments in Python begin with a # symbol, and Python will ignore everything following the # on that line:

```
# This is a comment


print("Hello, World!")
```

Inline Comments
Comments can also be placed at the end of a line, and Python will ignore the rest of the line:

```
print("Hello, World!")  # This is a comment
```

Multiline Comments

Using Multiple # Symbols
Python does not have a specific syntax for multiline comments. However, you can use multiple # symbols, one per line:

```
# This is a comment

# written in

# more than just one line

print("Hello, World!")
```

Using Triple Quotes for Multiline Comments

Since Python ignores string literals that are not assigned to a variable, you can use triple quotes (""" or ''') to create multiline comments:

```
"""

This is a comment

written in

more than just one line

"""

   print("Hello, World!")
```

1. Operators

Arithmetic Operators

These operators perform mathematical operations like addition, subtraction, etc.

- + : Addition
- - : Subtraction
- * : Multiplication
- / : Division
- % : Modulus (remainder of division)
- // : Floor division (returns the integer part of the division)
- ** : Exponentiation (raising to a power)

Example:

```
a = 10

b = 5

print(a + b)  # Output: 15

print(a - b)  # Output: 5
```

Comparison Operators

These operators compare two values and return True or False.

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

Logical Operators

These operators are used to combine conditional statements.

- and : Returns True if both conditions are true
- or : Returns True if at least one condition is true
- not : Reverses the result (returns True if the condition is false)

---

2. Variables and Data Types:

In Python, variables are used to store data. You can think of a variable as a box where you store something (like a number or a name). Data types tell Python what kind of data you're storing.

- **Integer**: Whole numbers (e.g., 5)
- **Float**: Numbers with decimal points (e.g., 3.14)
- **String**: Text (e.g., "Alice")
- **Boolean**: True or False values (e.g., True)

```
x = 5        # Integer
y = 3.14     # Float
name = "Alice" # String
is_active = True # Boolean
```

---

3. Lists:

A list is like a collection of items. You can store multiple items in a list, and each item can be accessed by its position (index). Lists are very flexible and allow you to change, add, or remove items.

```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])  # Output: apple
fruits.append("orange")  # Adding an item
```

```
Output: ['apple', 'banana', 'cherry', 'orange']
```

Lists (Advanced Operations):

Lists are one of the most important data structures in Python. You can perform various operations on them.

- **Slicing**: Extract a portion of a list.

```
numbers = [1, 2, 3, 4, 5]
print(numbers[1:4])  # Output: [2, 3, 4]
```

- **List Comprehension**: A compact way to create lists.

```
squares = [x**2 for x in range(5)]
print(squares)  # Output: [0, 1, 4, 9, 16]
```

- **Sorting and Reversing**:

```
numbers = [5, 2, 9, 1]
numbers.sort()  # Sorting the list in ascending order
print(numbers)  # Output: [1, 2, 5, 9]

numbers.reverse()  # Reversing the list
print(numbers)  # Output: [9, 5, 2, 1]
```

---

4. Tuples:

A **tuple** is similar to a list but with one important difference: **it is immutable**. This means that once you create a tuple, you cannot change, add, or remove elements from it. Tuples are useful when you want to store a collection of values that should not be modified, like the coordinates of a point or days of the week.

Creating a Tuple

Tuples are defined using parentheses (), and they can contain elements of different data types (e.g., integers, strings, booleans).

```
# Creating a tuple
my_tuple = (1, 2, 3, "apple", True)
print(my_tuple)  # Output: (1, 2, 3, 'apple', True)
```

Tuple Immutability

Since tuples are immutable, you cannot change their values after creation. Trying to do so will raise an error.

```
my_tuple[1] = 10  # ❌ This will raise an error
```

Tuple with One Element

To create a tuple with just one element, you need to add a trailing comma.

```
single_tuple = (5,)  # ✅ Correct
not_a_tuple = (5)    # ❌ This is just an integer
print(type(single_tuple))  # Output: <class 'tuple'>
print(type(not_a_tuple))   # Output: <class 'int'>
```

---

5. Dictionaries:

A dictionary is like a collection of key-value pairs. Each value is associated with a unique key. You can use the key to access the value.

```
person = {"name": "Alice", "age": 25}
print(person["name"])  # Output: Alice
```

---

6. Conditional Statements:

Conditional statements let you check if something is true or false and then take different actions based on that. It's like asking a question: "Is this true?" If yes, do one thing; if no, do something else.

```
x = 10
if x > 5:
   print("x is greater than 5")
else:
   print("x is less than or equal to 5")
```

---

7. Loops:

Loops allow you to repeat a block of code multiple times.

- **For Loop**: You use a for loop when you know how many times you want to repeat something.

```
for i in range(5):
   print(i)  # Output: 0, 1, 2, 3, 4
```

- **While Loop**: You use a while loop when you want to repeat something until a certain condition is met.

```
count = 0
while count < 5:
   print(count)
   count += 1  # Output: 0, 1, 2, 3, 4
```

---

8. Functions:

A function is a block of code that does something. You can define your own functions to organize your code and reuse it. Functions help make your code cleaner and more efficient.

```
def greet(name):
   return "Hello, " + name

print(greet("Alice"))  # Output: Hello, Alice
```

```
def square(x):

        return x * x

print(square(4)) # Output: 1
```

---

9. Classes and Objects:

In Python, you can create your own types using **classes**. A class is like a blueprint for creating objects. An **object** is an instance of a class. Classes allow you to group related data and functions together.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        return f"Hello, my name is {self.name} and I am {self.age} years old."

person1 = Person("Alice", 25)
print(person1.greet())  # Output: Hello, my name is Alice and I am 25 years old.
```

---

10. Importing Libraries:

Python comes with a lot of built-in libraries (also called **modules**) that help you do common tasks. You can import these libraries into your code to use their functionality.

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

---

11. Fibonacci Series in Python

```
def fib(n):

    a, b = 0, 1

    while a < n:

        print(a, end=' ')

        a, b = b, a + b

    print()
```

```
fib(1000)
```

Output:

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Fibonacci in Stock Market

```
def fib(n):
    a, b = 0, 1
    sequence = []
    while a < n:
        sequence.append(a)
        a, b = b, a + b
    return sequence

max_price = 1000
retracement_levels = fib(max_price)
print("Fibonacci retracement levels:", retracement_levels)
```

Output:

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
```