Python basics

Type	Data science masterclass

Basics of Python - Notion Notes

1. Introduction to Python

What is Python?

Python is a high-level, interpreted programming language known for its readability and simplicity. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Features of Python

- Easy to learn and read
- Interpreted language (no compilation required)
- Dynamically typed
- Large standard library
- Cross-platform compatibility
- Open-source and community-driven
- Supports GUI programming
- Extensive third-party libraries

Applications of Python

- Web Development (Django, Flask)
- Data Science and Machine Learning (NumPy, Pandas, TensorFlow)
- Automation and Scripting
- Cybersecurity

- Game Development
- Embedded Systems
- Internet of Things (IoT)

Installing Python and Setting up the Environment

1. Download Python from python.org.
2. Install it and ensure `Add to PATH` is selected.
3. Verify installation:

```
python --version
```

4. Install an IDE (VS Code, PyCharm, Jupyter Notebook).

Running Python Scripts

- Interactive mode:

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

- Running a script:

```
python script.py
```

2. Python Syntax and Basics

Writing and Executing Python Code

- Python uses `.py` files.
- Code is executed line by line (interpreted language).

Indentation and Code Structure

Python relies on indentation instead of curly braces.

```
if True:
    print("Indented block")
```

Comments in Python

- Single-line comment:

  ```
  # This is a comment
  ```

- Multi-line comment:

  ```
  """
  This is a multi-line comment
  spanning multiple lines.
  """
  ```

3. Variables and Data Types

Variable Declaration and Naming Rules

```
name = "Alice"  # String
age = 25        # Integer
height = 5.6    # Float
is_student = False  # Boolean
```

Data Types

```
x = 10         # Integer
y = 3.14       # Float
z = "Hello"    # String
b = True       # Boolean
c = 3 + 4j     # Complex Number
```

Type Conversion

```
# Implicit Conversion
x = 10
y = 2.5
z = x + y  # 12.5 (float)

# Explicit Conversion
a = "100"
b = int(a)  # 100 (integer)
```

Checking Data Types

```
print(type(42))        # <class 'int'>
print(type(3.14))      # <class 'float'>
print(type("hello"))   # <class 'str'>
```

4. Operators in Python

Arithmetic Operators

```
x, y = 10, 3
print(x + y)  # Addition
print(x - y)  # Subtraction
print(x * y)  # Multiplication
print(x / y)  # Division
print(x % y)  # Modulus
print(x ** y) # Exponentiation
print(x // y) # Floor Division
```

Comparison Operators

```
print(10 > 5)   # True
print(10 == 5)  # False
```

```
print(10 != 5)  # True
```

Logical Operators

```
print(True and False) # False
print(True or False)  # True
print(not True)       # False
```

Assignment Operators

```
x = 10
x += 5  # x = x + 5
```

Identity Operators

```
x = [1, 2, 3]
y = x
print(x is y)      # True (same object)
print(x is not y)  # False
```

Membership Operators

```
print(1 in [1, 2, 3])   # True
print(4 not in [1, 2, 3]) # True
```

Bitwise Operators

```
x, y = 5, 3
print(x & y)  # AND
print(x | y)  # OR
print(x ^ y)  # XOR
```

5. Input and Output

Taking User Input

```
name = input("Enter your name: ")
print("Hello, " + name)
```

Displaying Output

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

String Formatting

```
# f-strings
name = "Alice"
print(f"Hello, {name}!")

# .format()
print("Hello, {}!".format(name))

# % formatting
print("Hello, %s!" % name)
```

6. Conditional Statements

if, elif, else Statements

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

```
else:
    print("x is less than 5")
```

Nested Conditions

```
x = 10
y = 20
if x > 5:
    if y > 15:
        print("x is greater than 5 and y is greater than 15")
```

Short-hand if Statements

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

---

7. Loops in Python

for Loop

The `for` loop is used to iterate over a sequence (like a list, tuple, or string).

```
# Example: Looping through a list
for item in [1, 2, 3, 4]:
    print(item)

# Example: Looping through a string
for char in "Python":
    print(char)
```

while Loop

The `while` loop runs as long as the condition is true.

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

break, continue, pass Statements

- `break` exits the loop.
- `continue` skips to the next iteration.
- `pass` is a placeholder for code to be written later.

```
# break example
for i in range(5):
    if i == 3:
        break
    print(i)

# continue example
for i in range(5):
    if i == 2:
        continue
    print(i)

# pass example
for i in range(5):
    if i == 2:
        pass  # do nothing
    print(i)
```

Looping through Strings, Lists, and Dictionaries

```
# Looping through a list
my_list = [1, 2, 3]
for num in my_list:
    print(num)

# Looping through a string
word = "Python"
for letter in word:
    print(letter)

# Looping through a dictionary
my_dict = {"a": 1, "b": 2}
for key, value in my_dict.items():
    print(key, value)
```

---

8. Functions in Python

Defining Functions

Functions are defined using the `def` keyword.

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

greet("Alice")
```

Function Parameters and Arguments

You can define parameters to pass values to functions.

```
def add(a, b):
    return a + b
```

```
result = add(5, 3)
print(result)  # 8
```

Default and Keyword Arguments

```
def greet(name="User"):
    print(f"Hello, {name}!")

greet()          # Default value
greet("Alice")   # Keyword argument
```

return Statement

A function can return a value using `return`.

```
def multiply(a, b):
    return a * b

result = multiply(2, 4)
print(result)  # 8
```

Lambda (Anonymous) Functions

A lambda function is a small anonymous function.

```
square = lambda x: x * x
print(square(5))  # 25
```

---

9. Lists and Tuples

List Basics and Operations

Lists are mutable (can be modified).

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)  # [1, 2, 3, 4]
my_list.remove(2)
print(my_list)  # [1, 3, 4]
```

List Methods

```
my_list = [3, 1, 2]
my_list.sort()  # Sorts in ascending order
print(my_list)  # [1, 2, 3]
```

Tuples: Immutability, Packing and Unpacking

Tuples are immutable.

```
my_tuple = (1, 2, 3)
print(my_tuple[0])  # 1

# Packing and Unpacking
x, y, z = (1, 2, 3)
print(x, y, z)  # 1 2 3
```

Difference Between List and Tuple

- Lists are mutable, tuples are immutable.
- Tuples are faster than lists.

---

10. Dictionaries and Sets

Dictionary Basics and Operations

Dictionaries store key-value pairs.

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

Accessing, Adding, and Removing Elements

```
# Adding an element
my_dict["address"] = "123 Main St"

# Removing an element
del my_dict["age"]
```

Dictionary Methods

```
print(my_dict.keys())   # dict_keys(['name', 'address'])
print(my_dict.values()) # dict_values(['Alice', '123 Main S
t'])
print(my_dict.items())  # dict_items([('name', 'Alice'), ('ad
dress', '123 Main St')])
```

Set Basics and Operations

Sets are unordered collections of unique items.

```
my_set = {1, 2, 3}
my_set.add(4)
my_set.remove(2)
```

Set Methods

```
# Union
set1 = {1, 2}
set2 = {2, 3}
print(set1.union(set2))  # {1, 2, 3}
```

```
# Intersection
print(set1.intersection(set2))  # {2}
```

11. Strings in Python

String Basics and Indexing

Strings are sequences of characters.

```
my_string = "Python"
print(my_string[0])  # P
```

String Methods

```
my_string = "hello"
print(my_string.upper())    # HELLO
print(my_string.replace("e", "a"))  # hallo
print(my_string.split())    # ['hello']
```

String Concatenation and Repetition

```
greeting = "Hello"
name = "Alice"
print(greeting + " " + name)  # Hello Alice

print("Hello " * 3)  # Hello Hello Hello
```

String Slicing

```
my_string = "Python"
print(my_string[1:4])  # yth
```

Escape Sequences and Raw Strings

```
# Escape Sequence
print("Hello\nWorld")  # prints Hello on one line, World on t
he next

# Raw String (ignores escape characters)
print(r"Hello\nWorld")  # prints Hello\nWorld
```

---

12. File Handling

Reading a File

```
# Open file in read mode
with open("file.txt", "r") as file:
    content = file.read()
    print(content)
```

Writing to a File

```
# Open file in write mode
with open("file.txt", "w") as file:
    file.write("Hello, World!")
```

Appending to a File

```
# Open file in append mode
with open("file.txt", "a") as file:
    file.write("\nAppended text")
```

Working with Different File Modes

- `r`: Read mode
- `w`: Write mode

- `a`: Append mode
- `r+`: Read/Write mode

Using `with` Statement for File Handling

Using the `with` statement automatically closes the file when done.

```
with open("file.txt", "r") as file:
    data = file.read()
```

---

13. Exception Handling

Understanding Errors

Python has different types of errors, such as:

- `SyntaxError`: Incorrect syntax.
- `NameError`: Referencing a variable that hasn't been defined.
- `TypeError`: Invalid operation on a data type.

```
# Example of SyntaxError
# if True print("Hello")  # Incorrect syntax

# Example of NameError
# print(x)  # x is not defined

# Example of TypeError
# print(10 + "hello")  # Cannot add integer and string
```

try, except, else, finally Blocks

- `try`: Defines the block of code to check for errors.
- `except`: Handles the error if one occurs.

- `else`: Runs if no errors occur.
- `finally`: Always runs, regardless of whether an error occurred.

```
try:
    x = 10 / 0  # Dividing by zero
except ZeroDivisionError as e:
    print(f"Error: {e}")
else:
    print("No errors!")
finally:
    print("This will always execute.")
```

Raising Exceptions

You can raise your own exceptions using the `raise` keyword.

```
def check_age(age):
    if age < 18:
        raise ValueError("Age must be 18 or older")
    return age

try:
    check_age(15)
except ValueError as e:
    print(f"Error: {e}")
```

---

14. Modules and Packages

Importing Modules

Modules are files containing Python code. You can import them using the `import` keyword.

```
import math
```

```
print(math.sqrt(16))  # 4.0
```

from ... import

You can import specific functions or variables from a module.

```
from math import pi
print(pi)  # 3.14159
```

Built-in Modules

Python has a large standard library. Here are some examples:

- `math`: Provides mathematical functions.
- `random`: For generating random numbers.
- `datetime`: For working with dates and times.

```
import random
print(random.randint(1, 100))  # Random integer between 1 and
100

import datetime
print(datetime.datetime.now())  # Current date and time
```

Creating and Using Custom Modules

You can create your own module by saving Python code in a `.py` file. Here's how to use it:

```
# mymodule.py
def greet(name):
    print(f"Hello, {name}!")

# main.py
```

```
import mymodule
mymodule.greet("Alice")
```

Python Standard Library Overview

Some common modules in the standard library include:

- `os`: Interaction with the operating system.
- `sys`: Access to system-specific parameters.
- `json`: Handling JSON data.
- `collections`: Specialized container datatypes.