Complete Python Learning Notes

---

1. Python and its Features

What is Python?

Python is an interpreted, high-level, general-purpose programming language known for its simplicity and readability.

Key Features:

- Easy to learn and use
- Interpreted language
- Dynamically typed
- Extensive standard libraries
- Cross-platform
- Supports OOP (Object Oriented Programming)
- Huge community support

---

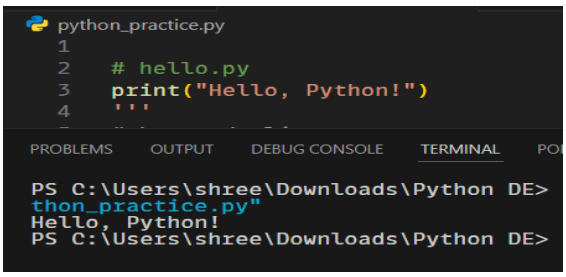
2. History of Python

- Created by **Guido van Rossum**
- Released in **1991**
- Influenced by languages like ABC, C, Modula-3

---

3. Writing and Running Your First Python Program

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



4. Keywords & Identifiers

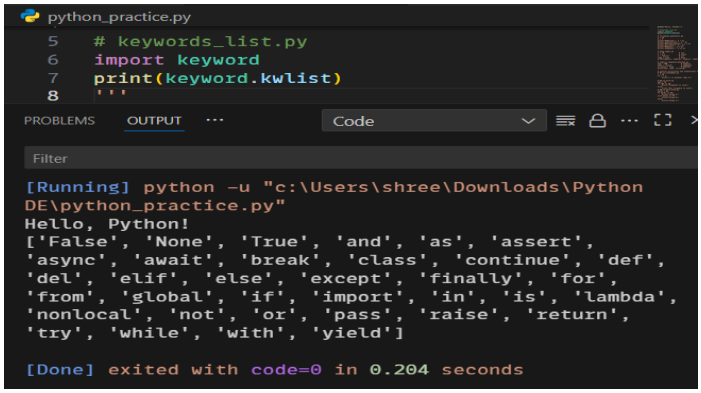
- **Keywords:**

  Reserved words like if, else, class, try, etc.
  You can list them using:

  # keywords_list.py

  import keyword

  print(keyword.kwlist)



- **Identifiers:**

  User-defined names for variables, functions, classes. Must start with a letter or underscore.

  # identifiers_example.py

  name = "Pooja"

  _age = 25

  print(name, _age)

---

5. Variables & Operators

  # variables_operators.py

  a = 10

  b = 5

  print("Addition:", a + b)

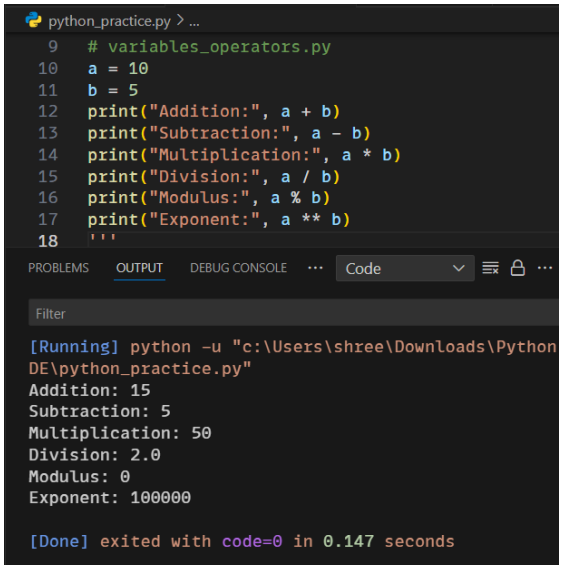
  print("Subtraction:", a - b)

print("Multiplication:", a * b)

print("Division:", a / b)

print("Modulus:", a % b)

print("Exponent:", a ** b)



---

6. Data Types

data_types.py

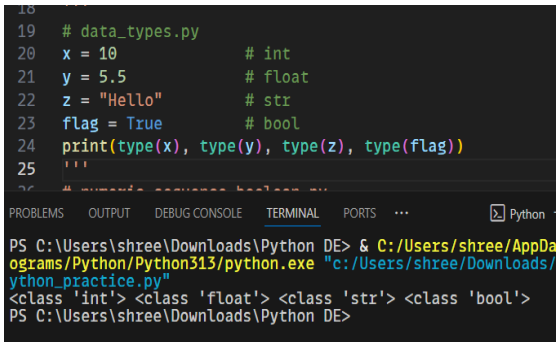
x = 10 # int

y = 5.5 # float

z = "Hello" # str

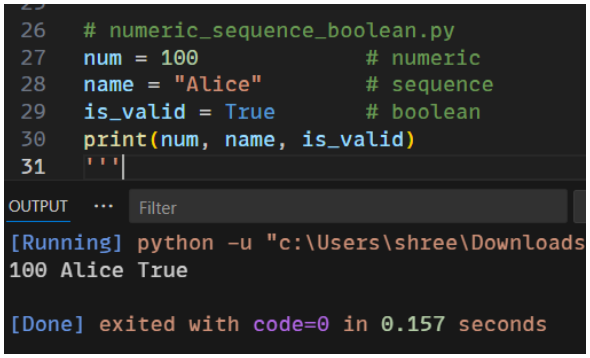
flag = True # bool

print(type(x), type(y), type(z), type(flag))



7. Numeric, Sequence, Boolean Types

```
# numeric_sequence_boolean.py
num = 100            # numeric
name = "Alice"       # sequence
is_valid = True      # boolean
print(num, name, is_valid)
```



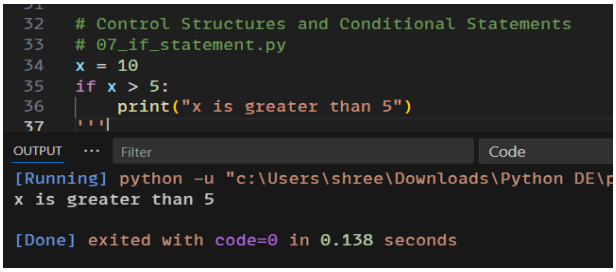
Control Structures and Conditional Statements

1. if Statement

python

CopyEdit

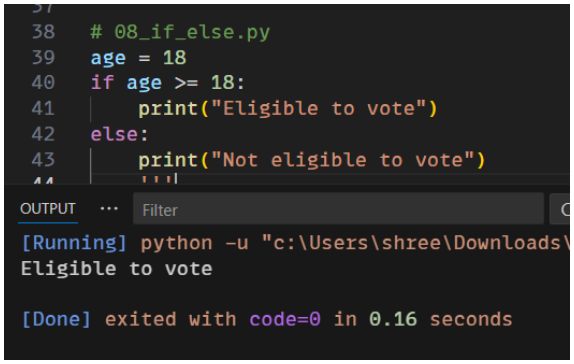
```
# 07_if_statement.py
x = 10
if x > 5:
    print("x is greater than 5")
```



---

2. if-else Statement

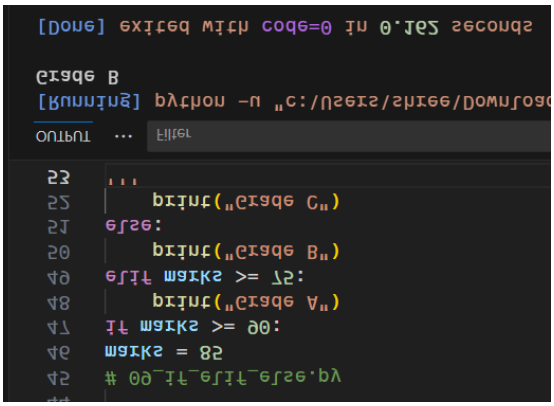
```
# 08_if_else.py

age = 18

if age >= 18:

    print("Eligible to vote")

else:

    print("Not eligible to vote")
```



---

3. if-elif-else Statement

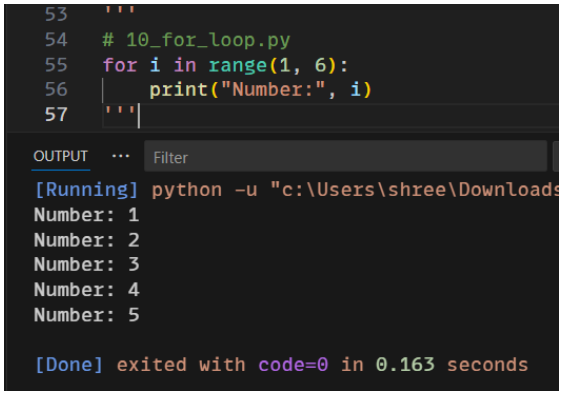
```
# 09_if_elif_else.py

marks = 85

if marks >= 90:

    print("Grade A")

elif marks >= 75:

    print("Grade B")

else:

    print("Grade C")
```



---

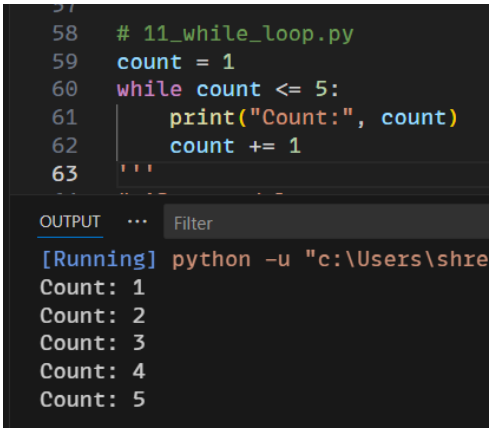
4. for Loop

```
# 10_for_loop.py

for i in range(1, 6):
    print("Number:", i)
```



5. while Loop

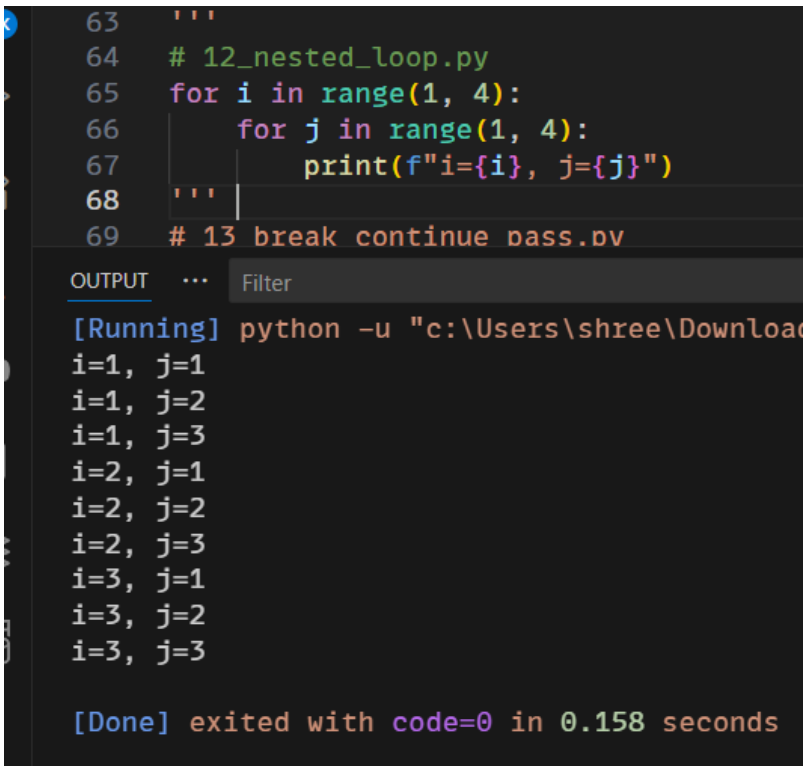
```
# 11_while_loop.py

count = 1

while count <= 5:
    print("Count:", count)
    count += 1
```



---

6. Nested Loops

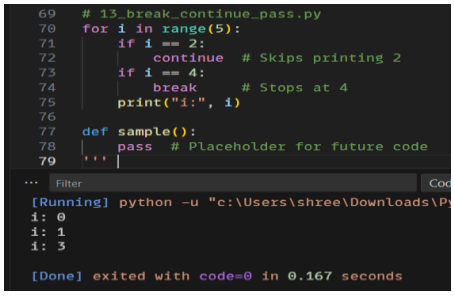
```
# 12_nested_loop.py

for i in range(1, 4):

    for j in range(1, 4):

        print(f"i={i}, j={j}")
```



---

7. Break, Continue, and Pass

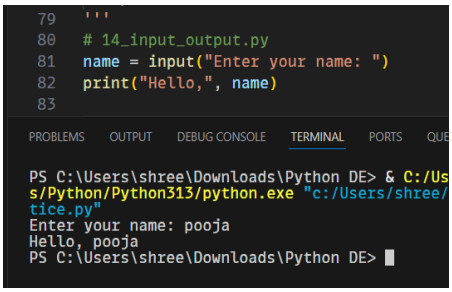
```
# 13_break_continue_pass.py

for i in range(5):

    if i == 2:

        continue  # Skips printing 2

    if i == 4:

        break     # Stops at 4

    print("i:", i)


def sample():

    pass  # Placeholder for future code
```



---

8. Input and Output

```
# 14_input_output.py

name = input("Enter your name: ")

print("Hello,", name)
```



Lists, Dictionaries, and Sets

1. Introduction to Lists

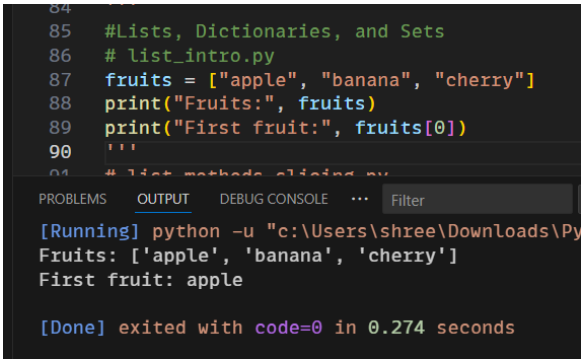
A list is an ordered collection of items that is mutable.

list_intro.py

fruits = ["apple", "banana", "cherry"]

print("Fruits:", fruits)

print("First fruit:", fruits[0])



---

2. List Methods and Slicing

list_methods_slicing.py

numbers = [10, 20, 30, 40, 50]

Methods

numbers.append(60)

numbers.remove(20)

numbers.insert(2, 25)

Slicing

print("List:", numbers)

print("First 3 elements:", numbers[:3])

print("Last 2 elements:", numbers[-2:])

```
# list_methods_slicing.py
numbers = [10, 20, 30, 40, 50]

# Methods
numbers.append(60)
numbers.remove(20)
numbers.insert(2, 25)

# Slicing
print("List:", numbers)
print("First 3 elements:", numbers[:3])
print("Last 2 elements:", numbers[-2:])
'''
```

PROBLEMS OUTPUT DEBUG CONSOLE ... Filter

```
[Running] python -u "c:\Users\shree\Downloads\Py
List: [10, 30, 25, 40, 50, 60]
First 3 elements: [10, 30, 25]
Last 2 elements: [50, 60]

[Done] exited with code=0 in 0.173 seconds
```

---

3. Introduction to Dictionaries

Dictionaries store key-value pairs.

dict_intro.py

student = {

"name": "Alice",

"age": 21,

"course": "Python"

}

print("Name:", student["name"])

print("Age:", student.get("age"))

```
# dict_intro.py
student = {
    "name": "Alice",
    "age": 21,
    "course": "Python"
}
print("Name:", student["name"])
print("Age:", student.get("age"))
'''
```

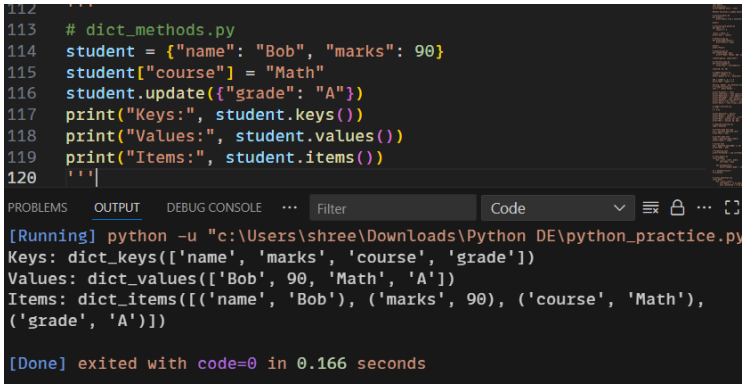
PROBLEMS OUTPUT DEBUG CONSOLE ... Filter

```
[Running] python -u "c:\Users\shree\Downloa
Name: Alice
Age: 21

[Done] exited with code=0 in 0.163 seconds
```

---

4. Dictionary Methods

```
# dict_methods.py

student = {"name": "Bob", "marks": 90}

student["course"] = "Math"

student.update({"grade": "A"})

print("Keys:", student.keys())

print("Values:", student.values())

print("Items:", student.items())
```



---

5. Introduction to Sets

A set is an unordered collection of unique items.

```
# set_intro.py

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

print("Unique Colors:", colors)
```

6. Set Methods

```
# set_methods.py

set1 = {1, 2, 3}

set2 = {3, 4, 5}

print("Union:", set1.union(set2))

print("Intersection:", set1.intersection(set2))
```

print("Difference:", set1.difference(set2))

set1.add(6)

set1.discard(2)

print("Updated Set1:", set1)

```
# set_intro.py
colors = {"red", "green", "blue", "red"}
print("Unique Colors:", colors)

# set_methods.py
set1 = {1, 2, 3}
set2 = {3, 4, 5}

print("Union:", set1.union(set2))
print("Intersection:", set1.intersection(set2))
print("Difference:", set1.difference(set2))

set1.add(6)
set1.discard(2)
print("Updated Set1:", set1)
'''
```

```
Filter                                  Code
[Running] python -u "c:\Users\shree\Downloads\Python DE
Unique Colors: {'green', 'red', 'blue'}
Union: {1, 2, 3, 4, 5}
Intersection: {3}
Difference: {1, 2}
Updated Set1: {1, 3, 6}

[Done] exited with code=0 in 0.168 seconds
```

Python Functions & Lambda Expressions

1. Defining and Calling a Function

function_basic.py

def greet():

    print("Hello from a function!")

greet()

2. Function with Parameters

function_with_params.py

def add(a, b):

    return a + b

result = add(5, 3)

print("Sum:", result)

3. Default Argument Values

```
# default_args.py
def greet(name="Guest"):
    print("Hello", name)
greet()
greet("Pooja")
```

4. Keyword Arguments

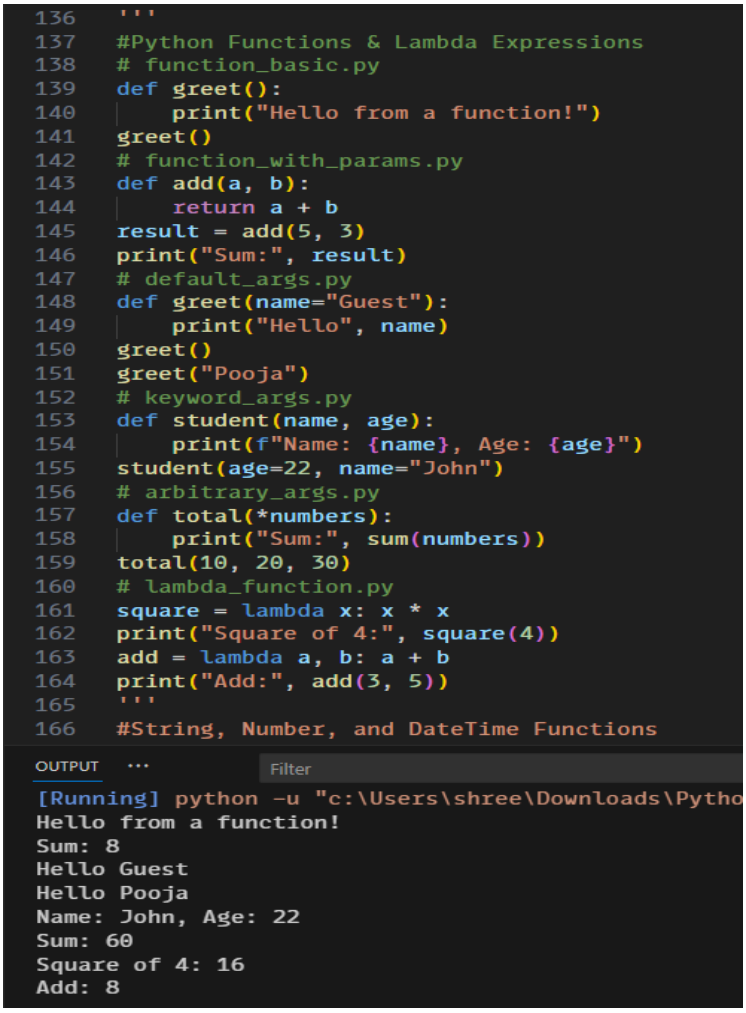
```
# keyword_args.py
def student(name, age):
    print(f"Name: {name}, Age: {age}")
student(age=22, name="John")
```

5. Arbitrary Argument Lists

```
# arbitrary_args.py
def total(*numbers):
    print("Sum:", sum(numbers))
total(10, 20, 30)
```

6. Lambda Expressions (Anonymous Functions)

```
# lambda_function.py
square = lambda x: x * x
print("Square of 4:", square(4))
add = lambda a, b: a + b
print("Add:", add(3, 5))
```



Python Built-in Functions:

String, Number, and DateTime Functions

1. String Functions

string_functions.py

text = " Hello Python "

print("Original:", text)

print("Lowercase:", text.lower())

print("Uppercase:", text.upper())

print("Stripped:", text.strip())

print("Replace:", text.replace("Python", "World"))

```
print("Starts with 'Hello':", text.strip().startswith("Hello"))
print("Split:", text.strip().split(" "))
```

---

2. Number Functions

```
# number_functions.py
x = -5
y = 4.75
print("Absolute:", abs(x))
print("Round:", round(y))
print("Power:", pow(2, 3))  # 2^3 = 8
print("Max:", max(10, 20, 30))
print("Min:", min(10, 20, 30))
```

3. Date and Time Functions

```
# datetime_functions.py
import datetime
# Current date and time
now = datetime.datetime.now()
print("Now:", now)
# Current date only
today = datetime.date.today()
print("Today:", today)
# Custom date
dob = datetime.date(2000, 5, 25)
print("DOB:", dob)
# Formatting date
```

print("Formatted:", now.strftime("%d-%m-%Y %H:%M:%S"))

Object-Oriented Programming in Python

1. Class and Object

class_object.py

class Student:

    def __init__(self, name):

```
        self.name = name
    def display(self):
        print("Student Name:", self.name)
s1 = Student("Alice")
s1.display()
```

2. Access Specifiers (Public, Protected, Private)

```
# access_specifiers.py
class Demo:
    def __init__(self):
        self.public = "I am public"
        self._protected = "I am protected"
        self.__private = "I am private"
    def show(self):
        print(self.public)
        print(self._protected)
        print(self.__private)
obj = Demo()
obj.show()
print(obj.public)       # Accessible
print(obj._protected)   # Accessible (but use with caution)
print(obj.__private)   # Error: Name mangling
print(obj._Demo__private)  # Accessing private via name mangling
```

```python
# class_object.py
class Student:
    def __init__(self, name):
        self.name = name

    def display(self):
        print("Student Name:", self.name)

s1 = Student("Alice")
s1.display()


# access_specifiers.py
class Demo:
    def __init__(self):
        self.public = "I am public"
        self._protected = "I am protected"
        self.__private = "I am private"

    # def show(self):
    #     print(self.public)
    #     print(self._protected)
    #     print(self.__private)

obj = Demo()
# obj.show()

print(obj.public)         # Accessible
print(obj._protected)     # Accessible (but use with caution)
# print(obj.__private)    # Error: Name mangling
# print(obj._Demo__private)  # Accessing private via name mangling
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  ...   Filter
[Running] python -u "c:\Users\shree\Downloads\Python DE\python_practice.py"
Student Name: Alice
I am public
I am protected

[Done] exited with code=0 in 0.168 seconds
```

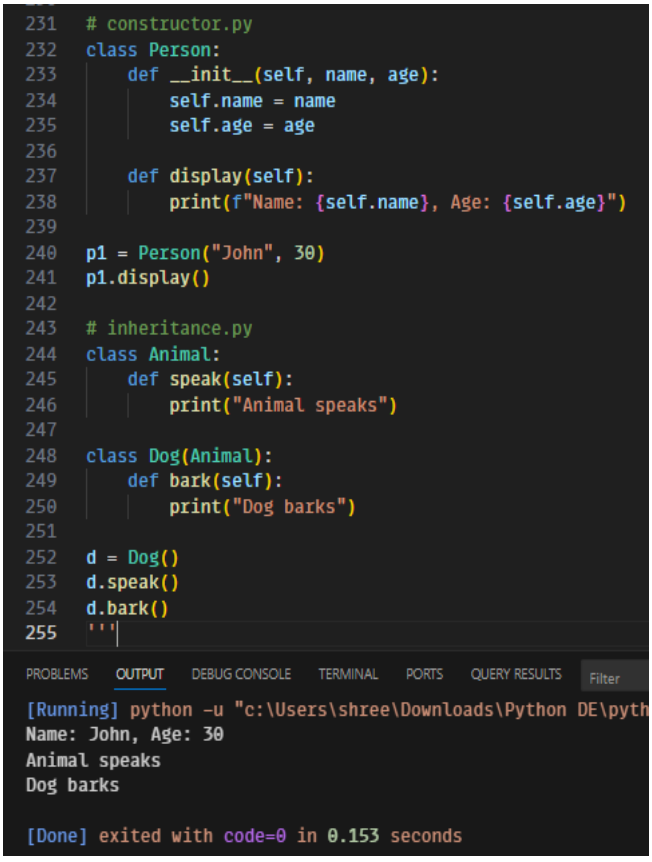
---

3. Constructor (__init__)

```
# constructor.py

class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age


    def display(self):

        print(f"Name: {self.name}, Age: {self.age}")

p1 = Person("John", 30)

p1.display()
```

4. Inheritance

```
# inheritance.py

class Animal:

    def speak(self):

        print("Animal speaks")

class Dog(Animal):

    def bark(self):

        print("Dog barks")

d = Dog()

d.speak()

d.bark()
```



---

5. Polymorphism

```
# polymorphism.py
class Bird:
    def intro(self):
        print("There are many types of birds.")
    def flight(self):
        print("Most birds can fly.")


class Parrot(Bird):
    def flight(self):
        print("Parrots can fly.")
class Ostrich(Bird):
    def flight(self):
        print("Ostriches cannot fly.")
b = Bird()
p = Parrot()
o = Ostrich()
b.flight()
p.flight()
o.flight()
```

6. Method Overriding

```
# method_overriding.py
class Parent:
    def show(self):
        print("Parent class")
```

```
class Child(Parent):
    def show(self):
        print("Child class")
c = Child()
c.show()
```

```
# polymorphism.py
class Bird:
    def intro(self):
        print("There are many types of birds.")

    def flight(self):
        print("Most birds can fly.")

class Parrot(Bird):
    def flight(self):
        print("Parrots can fly.")

class Ostrich(Bird):
    def flight(self):
        print("Ostriches cannot fly.")

b = Bird()
p = Parrot()
o = Ostrich()

b.flight()
p.flight()
o.flight()

# method_overriding.py
class Parent:
    def show(self):
        print("Parent class")

class Child(Parent):
    def show(self):
        print("Child class")

c = Child()
c.show()

'''
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  ...  Filter
[Running] python -u "c:\Users\shree\Downloads\Python DE\python_
Most birds can fly.
Parrots can fly.
Ostriches cannot fly.
Child class

[Done] exited with code=0 in 0.164 seconds
```

File Handling in Python (Text & Binary Files)

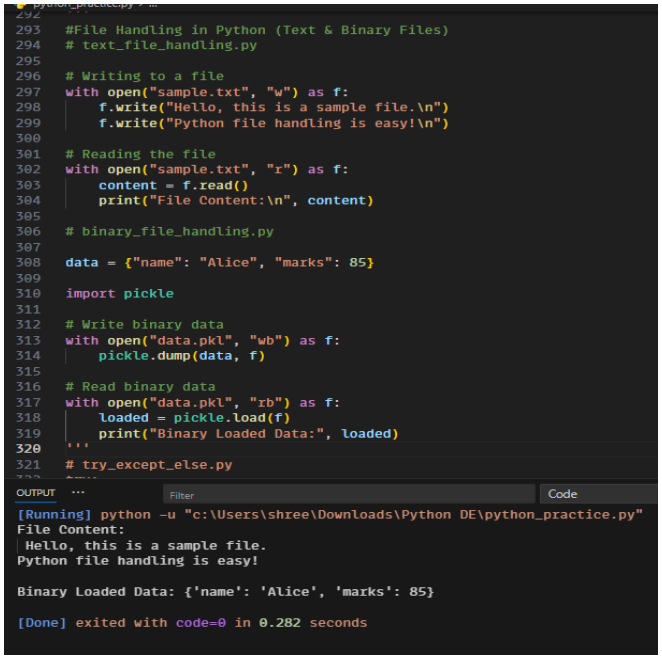
1. **Reading and Writing Text Files**

```
# text_file_handling.py

# Writing to a file

with open("sample.txt", "w") as f:

    f.write("Hello, this is a sample file.\n")
```

```
    f.write("Python file handling is easy!\n")

# Reading the file

with open("sample.txt", "r") as f:

    content = f.read()

    print("File Content:\n", content)
```

2. Binary File Handling

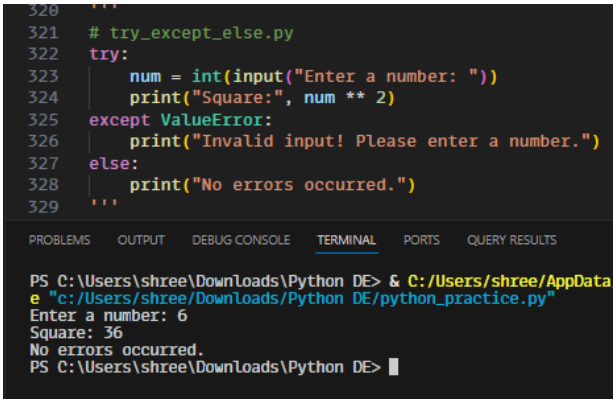
```
# binary_file_handling.py

data = {"name": "Alice", "marks": 85}

import pickle

# Write binary data

with open("data.pkl", "wb") as f:

    pickle.dump(data, f)

# Read binary data

with open("data.pkl", "rb") as f:

    loaded = pickle.load(f)

    print("Binary Loaded Data:", loaded)
```



Exception Handling in Python

1. try…except…else

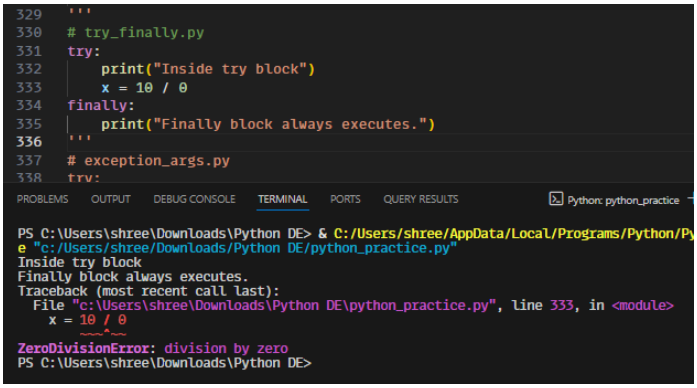
```
# try_except_else.py
try:
    num = int(input("Enter a number: "))
    print("Square:", num ** 2)
except ValueError:
    print("Invalid input! Please enter a number.")
else:
    print("No errors occurred.")
```



---

2. try…finally

```
# try_finally.py
try:
    print("Inside try block")
    x = 10 / 0
finally:
    print("Finally block always executes.")
```



---

3. Arguments of Exception

```
# exception_args.py

try:
    x = int("abc")
except ValueError as e:
    print("Error:", e)
```

4. Raising Exceptions

```
# raise_exception.py

def divide(x, y):
    if y == 0:
        raise ZeroDivisionError("You cannot divide by zero!")
    return x / y

print(divide(10, 2))

# print(divide(10, 0))  # Uncomment to see the exception
```

```
337  # exception_args.py
338  try:
339      x = int("abc")
340  except ValueError as e:
341      print("Error:", e)
342
343  # raise_exception.py
344  def divide(x, y):
345      if y == 0:
346          raise ZeroDivisionError("You cannot divide by zero!")
347      return x / y
348
349  print(divide(10, 2))
350  print(divide(10, 0))  # Uncomment to see the exception
351  '''
352  # user_defined_exception.py
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  QUERY RESULTS  Python: python_practice

PS C:\Users\shree\Downloads\Python DE> & C:/Users/shree/AppData/Local/Programs/Python/P
e "c:/Users/shree/Downloads/Python DE/python_practice.py"
Error: invalid literal for int() with base 10: 'abc'
5.0
Traceback (most recent call last):
  File "c:\Users\shree\Downloads\Python DE\python_practice.py", line 350, in <module>
    print(divide(10, 0))  # Uncomment to see the exception
          ~~~~~~^^^^^^^
  File "c:\Users\shree\Downloads\Python DE\python_practice.py", line 346, in divide
    raise ZeroDivisionError("You cannot divide by zero!")
ZeroDivisionError: You cannot divide by zero!
PS C:\Users\shree\Downloads\Python DE>
```

---

5. User-Defined Exceptions

user_defined_exception.py

class AgeTooSmallError(Exception):

    pass

age = 15

if age < 18:

    raise AgeTooSmallError("Age is too small to vote")

```
351  '''
352  # user_defined_exception.py
353  class AgeTooSmallError(Exception):
354      pass
355  age = 15
356  if age < 18:
357      raise AgeTooSmallError("Age is too small to vote")
358
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  QUERY RESULTS  Python: python_practice

PS C:\Users\shree\Downloads\Python DE> & C:/Users/shree/AppData/Local/Programs/Python/Pyt
e "c:/Users/shree/Downloads/Python DE/python_practice.py"
Traceback (most recent call last):
  File "c:\Users\shree\Downloads\Python DE\python_practice.py", line 357, in <module>
    raise AgeTooSmallError("Age is too small to vote")
AgeTooSmallError: Age is too small to vote
PS C:\Users\shree\Downloads\Python DE>
```

Python Modules & Packages

1. Creating and Using a User-Defined Module

Step 1: Create a module file:

```
# mymodule.py
def greet(name):
    return f"Hello, {name}!"
PI = 3.14
```

Step 2: Import and use it:

```
# use_module.py
import mymodule
print(mymodule.greet("Pooja"))
print("PI =", mymodule.PI)
```

---

2. Executing Module as a Script

```
# script_module.py
def welcome():
    print("Welcome to Python modules!")
if __name__ == "__main__":
    welcome()
```

---

3. Standard Modules (like math, random)

```
# std_module_math.py
import math
print("Square root:", math.sqrt(16))
print("Power:", math.pow(2, 3))
```

---

4. Creating a Package

Folder Structure:

mypackage/

├── __init__.py

├── calc.py

└── greetings.py

calc.py:

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

greetings.py:

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

Using the package:

```
# use_package.py
from mypackage import calc, greetings
print(calc.add(3, 4))
print(greetings.hello("Pooja"))
```

---

5. Importing * from a Package

```
# __init__.py in mypackage
__all__ = ["calc", "greetings"]
```

Then in your main:

```
from mypackage import *
```

```
print(calc.add(5, 7))
```

---

6. Intra-package References

Use relative imports inside packages like:

```
# Inside greetings.py
from .calc import add
```

Python + MySQL Integration (CRUD)

To use MySQL with Python, install the connector:

```
pip install mysql-connector-python
```

1. Create DB Connection

```
# db_connect.py
import mysql.connector
con = mysql.connector.connect(
    host="localhost",
    user="root",
    password="your_password",
    database="your_database"
)
print("Connected!" if con.is_connected() else "Failed to connect.")
```

---

2. CREATE Operation

```
python
CopyEdit
# db_insert.py
```

```
import mysql.connector
con = mysql.connector.connect(
   host="localhost", user="root", password="your_password", database="your_database"
)
cur = con.cursor()
cur.execute("INSERT INTO students (name, age) VALUES (%s, %s)", ("Alice", 21))
con.commit()
print("Data inserted.")
con.close()
```

3. READ Operation

```
# db_read.py
con = mysql.connector.connect(
   host="localhost", user="root", password="your_password", database="your_database"
)
cur = con.cursor()
cur.execute("SELECT * FROM students")
for row in cur.fetchall():
   print(row)
con.close()
```

---

4. UPDATE Operation

```
# db_update.py
con = mysql.connector.connect(
   host="localhost", user="root", password="your_password",
database="your_database")
cur = con.cursor()
cur.execute("UPDATE students SET age = 22 WHERE name = %s", ("Alice",))
```

```
con.commit()
print("Data updated.")
con.close()
```

5. DELETE Operation

```
# db_delete.py
con = mysql.connector.connect(
    host="localhost", user="root", password="your_password", database="your_database"
)
cur = con.cursor()
cur.execute("DELETE FROM students WHERE name = %s", ("Alice",))
con.commit()
print("Data deleted.")
con.close()
```