

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!")
```

```
python_practice.py
1
2  # hello.py
3  print("Hello, Python!")
4 '''

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PO

PS C:\Users\shree\Downloads\Python DE>
thon_practice.py
Hello, Python!
PS C:\Users\shree\Downloads\Python DE>
```

4. Keywords & Identifiers

◆ Keywords:

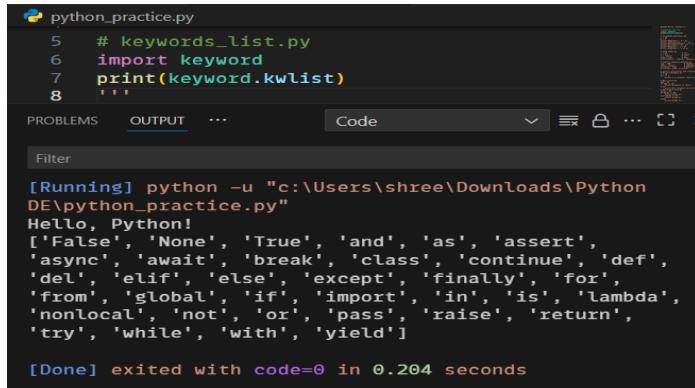
Reserved words like if, else, class, try, etc.

You can list them using:

```
# keywords_list.py
```

```
import keyword
```

```
print(keyword.kwlist)
```



The screenshot shows a code editor window with a Python file named 'python_practice.py'. The code in the file is:

```
5 # keywords_list.py
6 import keyword
7 print(keyword.kwlist)
8 ''''
```

Below the code, the 'OUTPUT' tab is selected, displaying the results of running the script:

```
[Running] python -u "c:\Users\shree\Downloads\Python DE\python_practice.py"
Hello, Python!
['False', 'None', 'True', 'and', 'as', 'assert',
 'async', 'await', 'break', 'class', 'continue', 'def',
 'del', 'elif', 'else', 'except', 'finally', 'for',
 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
 'try', 'while', 'with', 'yield']
```

The output ends with:

```
[Done] exited with code=0 in 0.204 seconds
```

◆ 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)

```

```

python_practice.py > ...
9  # variables_operators.py
10 a = 10
11 b = 5
12 print("Addition:", a + b)
13 print("Subtraction:", a - b)
14 print("Multiplication:", a * b)
15 print("Division:", a / b)
16 print("Modulus:", a % b)
17 print("Exponent:", a ** b)
18 ...

PROBLEMS  OUTPUT  DEBUG CONSOLE  ...  Code  Filter  ...

[Running] python -u "c:\Users\shree\Downloads\Python DE\python_practice.py"
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Modulus: 0
Exponent: 100000

[Done] exited with code=0 in 0.147 seconds

```

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))

```

```

18
19  # data_types.py
20  x = 10          # int
21  y = 5.5         # float
22  z = "Hello"     # str
23  flag = True     # bool
24  print(type(x), type(y), type(z), type(flag))
25
26

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  ...  Python

PS C:\Users\shree\Downloads\Python DE> & C:/Users/shree/AppData/Programs/Python/Python313/python.exe "c:/Users/shree/Downloads/python_practice.py"
<class 'int'> <class 'float'> <class 'str'> <class 'bool'>
PS C:\Users\shree\Downloads\Python DE>

```

7. Numeric, Sequence, Boolean Types

```
# numeric_sequence_boolean.py

num = 100      # numeric

name = "Alice"    # sequence

is_valid = True    # boolean

print(num, name, is_valid)
```

```
25
26  # numeric_sequence_boolean.py
27  num = 100          # numeric
28  name = "Alice"      # sequence
29  is_valid = True       # boolean
30  print(num, name, is_valid)
31  ''

OUTPUT ... Filter
[Running] python -u "c:\Users\shree\Downloads\Python DE\7\7. Numeric, Sequence, Boolean Types\06_nu...
100 Alice True

[Done] exited with code=0 in 0.157 seconds
```

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")
```

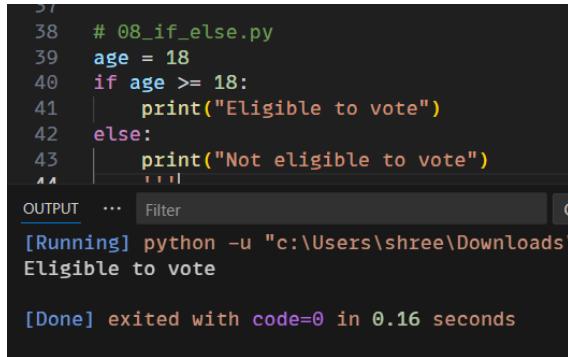
```
31
32  # Control Structures and Conditional Statements
33  # 07_if_statement.py
34  x = 10
35  if x > 5:
36  |   print("x is greater than 5")
37  ...

OUTPUT ... Filter
[Running] python -u "c:\Users\shree\Downloads\Python DE\7\7. Control Structures and Conditional Statements\07_if...
x is greater than 5

[Done] exited with code=0 in 0.138 seconds
```

2. if-else Statement

```
# 08_if_else.py  
age = 18  
  
if age >= 18:  
  
    print("Eligible to vote")  
  
else:  
  
    print("Not eligible to vote")
```



```
37  
38     # 08_if_else.py  
39     age = 18  
40     if age >= 18:  
41         print("Eligible to vote")  
42     else:  
43         print("Not eligible to vote")  
44  
[Running] python -u "c:\Users\shree\Downloads\08_if_else.py"  
Eligible to vote  
  
[Done] exited with code=0 in 0.16 seconds
```

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")
```

```
[Done] exited with code=0 in 0.0 seconds
```

```
exitcode B
[Running] python -u "c:/Users/shree/Downloads/09_if_elif.py"
OUTPUT ... Filter
23 | ...
22 |     ...
21 |     if marks >= 12:
20 |         print("Grade C")
19 |     elif marks >= 18:
18 |         print("Grade B")
17 |     else:
16 |         print("Grade A")
15 |     marks = 82
14 | # Output
```

4. for Loop

```
# 10_for_loop.py
for i in range(1, 6):
    print("Number:", i)
```

```
53 | ...
54 | # 10_for_loop.py
55 | for i in range(1, 6):
56 |     print("Number:", i)
57 | ...

OUTPUT ... Filter
[Running] python -u "c:/Users/shree/Downloads/10_for_loop.py"
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5

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

5. while Loop

```
# 11_while_loop.py
count = 1
while count <= 5:
    print("Count:", count)
    count += 1
```

```
57
58     # 11_while_loop.py
59     count = 1
60     while count <= 5:
61         print("Count:", count)
62         count += 1
63     ''''
```

OUTPUT ... Filter

```
[Running] python -u "c:\Users\shree\Downloads\11_while_loop.py"
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

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}')
```

```
63     '''''
64     # 12_nested_loop.py
65     for i in range(1, 4):
66         for j in range(1, 4):
67             print(f'i={i}, j={j}')
68     ''''
```

OUTPUT ... Filter

```
[Running] python -u "c:\Users\shree\Downloads\12_nested_loop.py"
i=1, j=1
i=1, j=2
i=1, j=3
i=2, j=1
i=2, j=2
i=2, j=3
i=3, j=1
i=3, j=2
i=3, j=3
```

[Done] exited with code=0 in 0.158 seconds

7. Break, Continue, and Pass

```
# 13_break_continue_pass.py

for i in range(5):

    if i == 2:

        continue # Skips printing 2

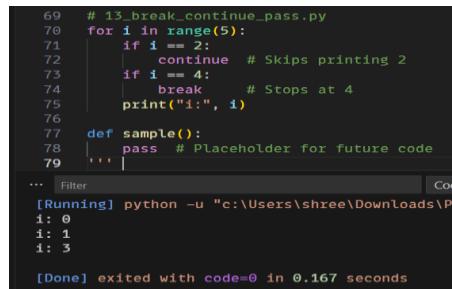
    if i == 4:

        break # Stops at 4

    print("i:", i)
```

```
def sample():

    pass # Placeholder for future code
```



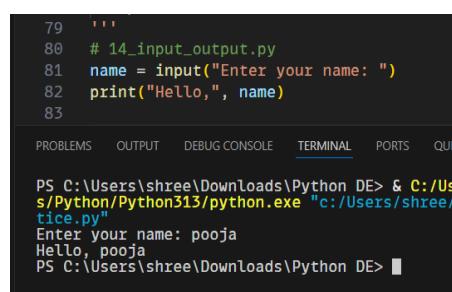
```
69  # 13_break_continue_pass.py
70  for i in range(5):
71      if i == 2:
72          continue # Skips printing 2
73      if i == 4:
74          break # Stops at 4
75      print("i:", i)
76
77 def sample():
78     pass # Placeholder for future code
79 ...
...
[Running] python -u "c:/Users/shree/Downloads/13_break_continue_pass.py"
i: 0
i: 1
i: 3
[Done] exited with code=0 in 0.167 seconds
```

8. Input and Output

```
# 14_input_output.py

name = input("Enter your name: ")

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



```
79  ...
80  # 14_input_output.py
81  name = input("Enter your name: ")
82  print("Hello, " + name)
83
...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUE

PS C:\Users\shree\Downloads\Python DE> & C:/Users/shree/Downloads/Python313/python.exe "c:/Users/shree/Downloads/14_input_output.py"
Enter your name: pooja
Hello, pooja
PS C:\Users\shree\Downloads\Python DE>
```

Lists, Dictionaries, and Sets

1. Introduction to Lists

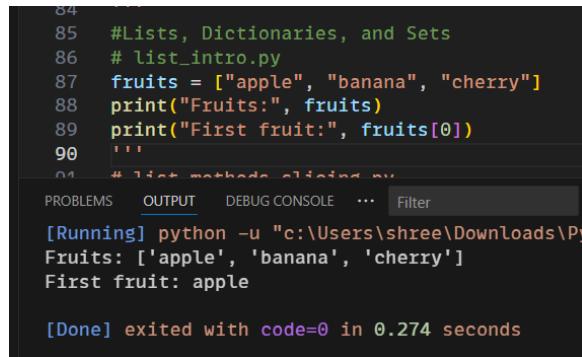
A list is an ordered collection of items that is mutable.

```
# list_intro.py

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

print("Fruits:", fruits)

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



```
84 # Lists, Dictionaries, and Sets
85 # list_intro.py
86 fruits = ["apple", "banana", "cherry"]
87 print("Fruits:", fruits)
88 print("First fruit:", fruits[0])
89 ...
90 ...

[Running] python -u "c:\Users\shree\Downloads\Py
Fruits: ['apple', 'banana', 'cherry']
First fruit: apple

[Done] exited with code=0 in 0.274 seconds
```

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:])
```

```
91 # list_methods_slicing.py
92 numbers = [10, 20, 30, 40, 50]
93
94 # Methods
95 numbers.append(60)
96 numbers.remove(20)
97 numbers.insert(2, 25)
98
99 # Slicing
100 print("List:", numbers)
101 print("First 3 elements:", numbers[:3])
102 print("Last 2 elements:", numbers[-2:])
103 '''

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"))
```

```
104 # dict_intro.py
105 student = {
106     "name": "Alice",
107     "age": 21,
108     "course": "Python"
109 }
110 print("Name:", student["name"])
111 print("Age:", student.get("age"))
112 ''

PROBLEMS OUTPUT DEBUG CONSOLE ... Filter
[Running] python -u "c:\Users\shree\Downloads\Py
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())

112 """
113 # dict_methods.py
114 student = {"name": "Bob", "marks": 90}
115 student["course"] = "Math"
116 student.update({"grade": "A"})
117 print("Keys:", student.keys())
118 print("Values:", student.values())
119 print("Items:", student.items())
120 """

PROBLEMS OUTPUT DEBUG CONSOLE ... Filter Code ⚙️ ⌂ ⌂ ⌂ [3]
[Running] python -u "c:\Users\shree\Downloads\Python DE\python_practice.py"
Keys: dict_keys(['name', 'marks', 'course', 'grade'])
Values: dict_values(['Bob', 90, 'Math', 'A'])
Items: dict_items([('name', 'Bob'), ('marks', 90), ('course', 'Math'), ('grade', 'A')])

[Done] exited with code=0 in 0.166 seconds
```

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)

120
121 # set_intro.py
122 colors = {"red", "green", "blue", "red"}
123 print("Unique Colors:", colors)
124
125 # set_methods.py
126 set1 = {1, 2, 3}
127 set2 = {3, 4, 5}
128
129 print("Union:", set1.union(set2))
130 print("Intersection:", set1.intersection(set2))
131 print("Difference:", set1.difference(set2))
132
133 set1.add(6)
134 set1.discard(2)
135 print("Updated Set1:", set1)
136 '''

... 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))
```

```
136  """
137  #Python Functions & Lambda Expressions
138  # function_basic.py
139  def greet():
140      print("Hello from a function!")
141  greet()
142  # function_with_params.py
143  def add(a, b):
144      return a + b
145  result = add(5, 3)
146  print("Sum:", result)
147  # default_args.py
148  def greet(name="Guest"):
149      print("Hello", name)
150  greet()
151  greet("Pooja")
152  # keyword_args.py
153  def student(name, age):
154      print(f"Name: {name}, Age: {age}")
155  student(age=22, name="John")
156  # arbitrary_args.py
157  def total(*numbers):
158      print("Sum:", sum(numbers))
159  total(10, 20, 30)
160  # lambda_function.py
161  square = lambda x: x * x
162  print("Square of 4:", square(4))
163  add = lambda a, b: a + b
164  print("Add:", add(3, 5))
165  """
166  #String, Number, and DateTime Functions
```

OUTPUT ... Filter

```
[Running] python -u "c:\Users\shree\Downloads\Python Functions & Lambda Expressions\function_basic.py"
Hello from a function!
Sum: 8
Hello Guest
Hello Pooja
Name: John, Age: 22
Sum: 60
Square of 4: 16
Add: 8
```

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"))
```

```
python_practice.py > ...
166  #String, Number, and DateTime Functions
167  # string_functions.py
168  text = "Hello Python"
169  print("Original:", text)
170  print("Lowercase:", text.lower())
171  print("Uppercase:", text.upper())
172  print("Stripped:", text.strip())
173  print("Replace:", text.replace("Python", "World"))
174  print("Starts with 'Hello':", text.startswith("Hello"))
175  print("Split:", text.split(" "))
176  # number_functions.py
177  x = -5
178  y = 4.75
179  print("Absolute:", abs(x))
180  print("Round:", round(y))
181  print("Power:", pow(2, 3)) # 2^3 = 8
182  print("Max:", max(10, 20, 30))
183  print(["Min:", min(10, 20, 30)])
184  # datetime_functions.py
185  import datetime
186  # Current date and time
187  now = datetime.datetime.now()
188  print("Now:", now)
189  # Current date only
190  today = datetime.date.today()
191  print("Today:", today)
192  # Custom date
193  dob = datetime.date(2000, 5, 25)
194  print("DOB:", dob)
195  # Formatting date
196  print("Formatted:", now.strftime("%d-%m-%Y %H:%M:%S"))
197  ...

PROBLEMS    OUTPUT    DEBUG CONSOLE ... Filter
[Running] python -u "c:\Users\shree\Downloads\Python DE\python_practice.py"
Original: Hello Python
Lowercase: hello python
Uppercase: HELLO PYTHON
Stripped: Hello Python
Replace: Hello World
Starts with 'Hello': True
Split: ['Hello', 'Python']
Absolute: 5
Round: 5
Power: 8
Max: 30
Min: 10
Now: 2025-07-26 15:28:18.066398
Today: 2025-07-26
DOB: 2000-05-25
Formatted: 26-07-2025 15:28:18

[DONE] exited with code=0 in 0.182 seconds
```

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
```

```

198 # class_object.py
199 class Student:
200     def __init__(self, name):
201         self.name = name
202
203     def display(self):
204         print("Student Name:", self.name)
205
206 s1 = Student("Alice")
207 s1.display()
208
209
210 # access_specifiers.py
211 class Demo:
212     def __init__(self):
213         self.public = "I am public"
214         self._protected = "I am protected"
215         self.__private = "I am private"
216
217     # def show(self):
218     #     print(self.public)
219     #     print(self._protected)
220     #     print(self.__private)
221
222 obj = Demo()
223 # obj.show()
224
225 print(obj.public)      # Accessible
226 print(obj._protected) # Accessible (but use with caution)
227 # print(obj.__private) # Error: Name mangling
228 # 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()
```

```
231 # constructor.py
232 class Person:
233     def __init__(self, name, age):
234         self.name = name
235         self.age = age
236
237     def display(self):
238         print(f"Name: {self.name}, Age: {self.age}")
239
240 p1 = Person("John", 30)
241 p1.display()
242
243 # inheritance.py
244 class Animal:
245     def speak(self):
246         print("Animal speaks")
247
248 class Dog(Animal):
249     def bark(self):
250         print("Dog barks")
251
252 d = Dog()
253 d.speak()
254 d.bark()
255 '''

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    QUERY RESULTS    Filter
[Running] python -u "c:\Users\shree\Downloads\Python DE\pyth
Name: John, Age: 30
Animal speaks
Dog barks

[Done] exited with code=0 in 0.153 seconds
```

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()

```

The screenshot shows a code editor with two files open. The left file contains code demonstrating polymorphism and method overriding. The right file shows the output of running the code in a terminal.

```

256 # polymorphism.py
257 class Bird:
258     def intro(self):
259         print("There are many types of birds.")
260
261     def flight(self):
262         print("Most birds can fly.")
263
264 class Parrot(Bird):
265     def flight(self):
266         print("Parrots can fly.")
267
268 class Ostrich(Bird):
269     def flight(self):
270         print("Ostriches cannot fly.")
271
272 b = Bird()
273 p = Parrot()
274 o = Ostrich()
275
276 b.flight()
277 p.flight()
278 o.flight()
279
280 # method_overriding.py
281 class Parent:
282     def show(self):
283         print("Parent class")
284
285 class Child(Parent):
286     def show(self):
287         print("Child class")
288
289 c = Child()
290 c.show()
291
292 '''

```

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)

```

```

292 #File Handling in Python (Text & Binary Files)
293 # text_file_handling.py
294
295
296 # Writing to a file
297 with open("sample.txt", "w") as f:
298     f.write("Hello, this is a sample file.\n")
299     f.write("Python file handling is easy!\n")
300
301 # Reading the file
302 with open("sample.txt", "r") as f:
303     content = f.read()
304     print("File Content:\n", content)
305
306 # binary_file_handling.py
307
308 data = {"name": "Alice", "marks": 85}
309
310 import pickle
311
312 # Write binary data
313 with open("data.pkl", "wb") as f:
314     pickle.dump(data, f)
315
316 # Read binary data
317 with open("data.pkl", "rb") as f:
318     loaded = pickle.load(f)
319     print("Binary Loaded Data:", loaded)
320 ...
321 # try_except_else.py
322 ...
323
324 OUTPUT ... Filter Code
[Running] python -u "c:/Users/shree/Downloads/Python DE/python_practice.py"
File Content:
Hello, this is a sample file.
Python file handling is easy!

Binary Loaded Data: {'name': 'Alice', 'marks': 85}
[Done] exited with code=0 in 0.282 seconds

```

Exception Handling in Python

1. try...except...else

```
# try_except_else.py

try:

    num = int(input("Enter a number: "))

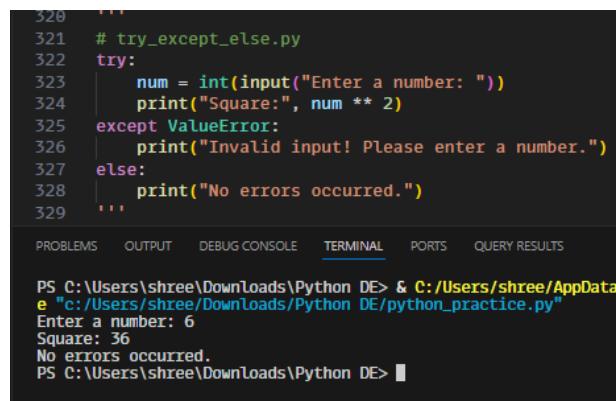
    print("Square:", num ** 2)

except ValueError:

    print("Invalid input! Please enter a number.")

else:

    print("No errors occurred.")
```



```
320  """
321  # try_except_else.py
322  try:
323      num = int(input("Enter a number: "))
324      print("Square:", num ** 2)
325  except ValueError:
326      print("Invalid input! Please enter a number.")
327  else:
328      print("No errors occurred.")
329  """

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    QUERY RESULTS

PS C:\Users\shree\Downloads\Python DE> & C:/Users/shree/AppData
e "c:/Users/shree/Downloads/Python DE/python_practice.py"
Enter a number: 6
Square: 36
No errors occurred.
PS C:\Users\shree\Downloads\Python DE>
```

2. try...finally

```
# try_finally.py

try:

    print("Inside try block")

    x = 10 / 0

finally:

    print("Finally block always executes.")
```

A screenshot of a Python terminal window. The code in the terminal is:

```
329  """
330  # try_finally.py
331  try:
332      print("Inside try block")
333      x = 10 / 0
334  finally:
335      print("Finally block always executes.")
336  ...
337  # exception_args.py
338  trv:
```

The terminal output shows:

```
PS C:\Users\shree\Downloads\Python DE> & C:/Users/shree/AppData/Local/Programs/Python/Py
e "c:/Users/shree/Downloads/Python DE/python_practice.py"
Inside try block
Finally block always executes.
Traceback (most recent call last):
File "c:/Users/shree/Downloads/Python DE/python_practice.py", line 333, in <module>
x = 10 / 0
ZeroDivisionError: division by zero
PS C:\Users\shree\Downloads\Python DE>
```

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 nv
```

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"
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()
```