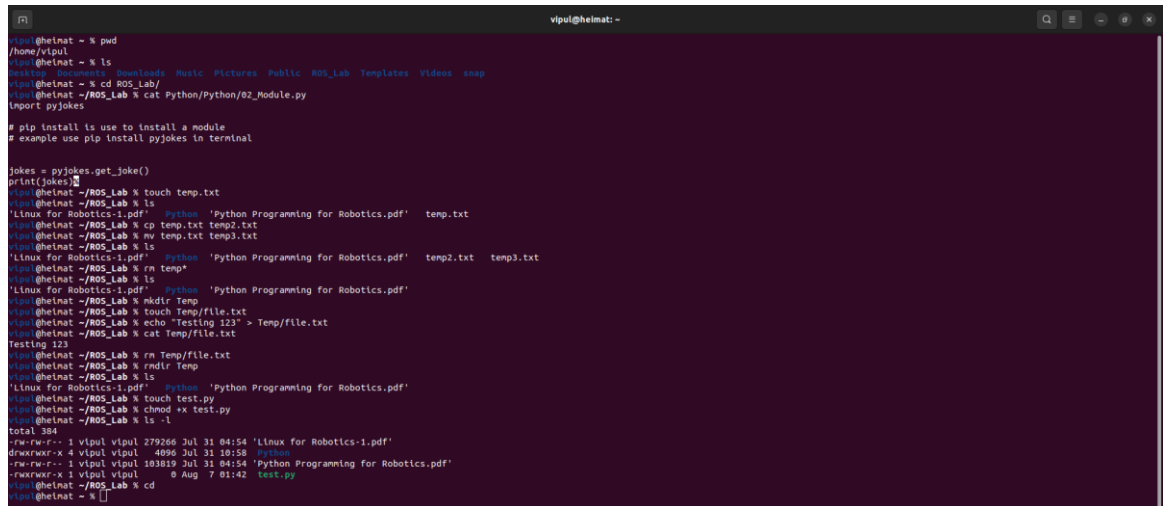


ROS Lab - Week 1

1. Common Linux Commands

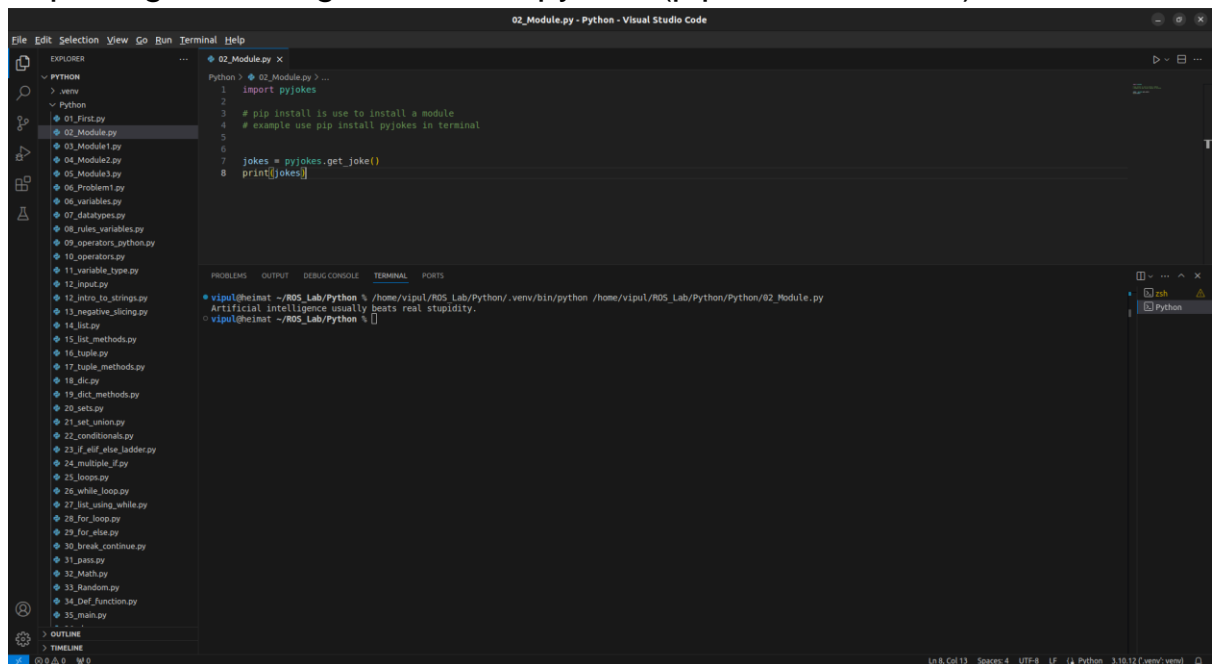


```
vipul@heimat: ~$ pwd
/home/vipul
vipul@heimat: ~$ ls
Desktop  Downloads  Music  Pictures  Public  ROS_Lab  Templates  Videos  snap
vipul@heimat: ~$ cd ROS_Lab/
vipul@heimat: ~/ROS_Lab$ cat Python/Python/02_Module.py
import pyjokes

# pip install is use to install a module
# example use pip install pyjokes in terminal

jokes = pyjokes.get_joke()
print(jokes)
vipul@heimat: ~/ROS_Lab$ touch temp.txt
vipul@heimat: ~/ROS_Lab$ ls
'Linux for Robotics-1.pdf'  Python  'Python Programming for Robotics.pdf'  temp.txt
vipul@heimat: ~/ROS_Lab$ mv temp.txt temp3.txt
vipul@heimat: ~/ROS_Lab$ ls
'Linux for Robotics-1.pdf'  Python  'Python Programming for Robotics.pdf'  temp2.txt  temp3.txt
vipul@heimat: ~/ROS_Lab$ rm temp*
vipul@heimat: ~/ROS_Lab$ ls
'Linux for Robotics-1.pdf'  Python  'Python Programming for Robotics.pdf'
vipul@heimat: ~/ROS_Lab$ mkdir Temp
vipul@heimat: ~/ROS_Lab$ touch Temp/file.txt
vipul@heimat: ~/ROS_Lab$ echo "Testing 123" > Temp/file.txt
vipul@heimat: ~/ROS_Lab$ cat Temp/file.txt
Testing 123
vipul@heimat: ~/ROS_Lab$ rm Temp/file.txt
vipul@heimat: ~/ROS_Lab$ rmdir Temp
vipul@heimat: ~/ROS_Lab$ ls
'Linux for Robotics-1.pdf'  Python  'Python Programming for Robotics.pdf'
vipul@heimat: ~/ROS_Lab$ touch test.py
vipul@heimat: ~/ROS_Lab$ chmod +x test.py
vipul@heimat: ~/ROS_Lab$ ls -l
total 384
-rw-rw-r-- 1 vipul vipul 279266 Jul 31 04:54 'Linux for Robotics-1.pdf'
drwxrwxr-x 4 vipul vipul 4896 Jul 31 10:58 Python
-rwxrwxr-x 1 vipul vipul 103819 Jul 31 04:54 'Python Programming for Robotics.pdf'
-rwxrwxr-x 1 vipul vipul    0 Aug  7 01:42 test.py
vipul@heimat: ~/ROS_Lab$ cd
vipul@heimat: ~$
```

2. Importing and using modules in python (pip for installation)



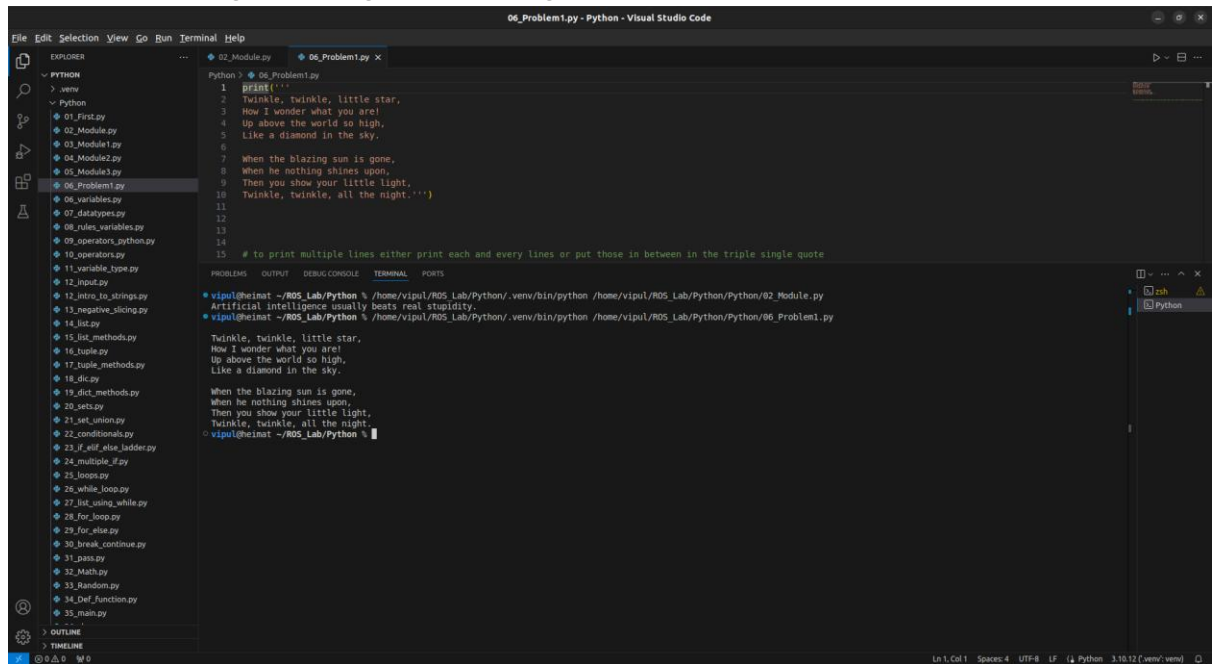
The screenshot shows the Visual Studio Code interface with a file named `02_Module.py` open. The file contains the following code:

```
1 import pyjokes
2
3 # pip install is use to install a module
4 # example use pip install pyjokes in terminal
5
6
7 jokes = pyjokes.get_joke()
8 print(jokes)
```

The output of the script is displayed in the terminal window at the bottom:

```
• vipul@heimat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/.venv/bin/python /home/vipul/ROS_Lab/Python/Python/02_Module.py
Artificial Intelligence usually beats real stupidity.
• vipul@heimat ~/ROS_Lab/Python %
```

3. Multiline string printing (Docstrings)



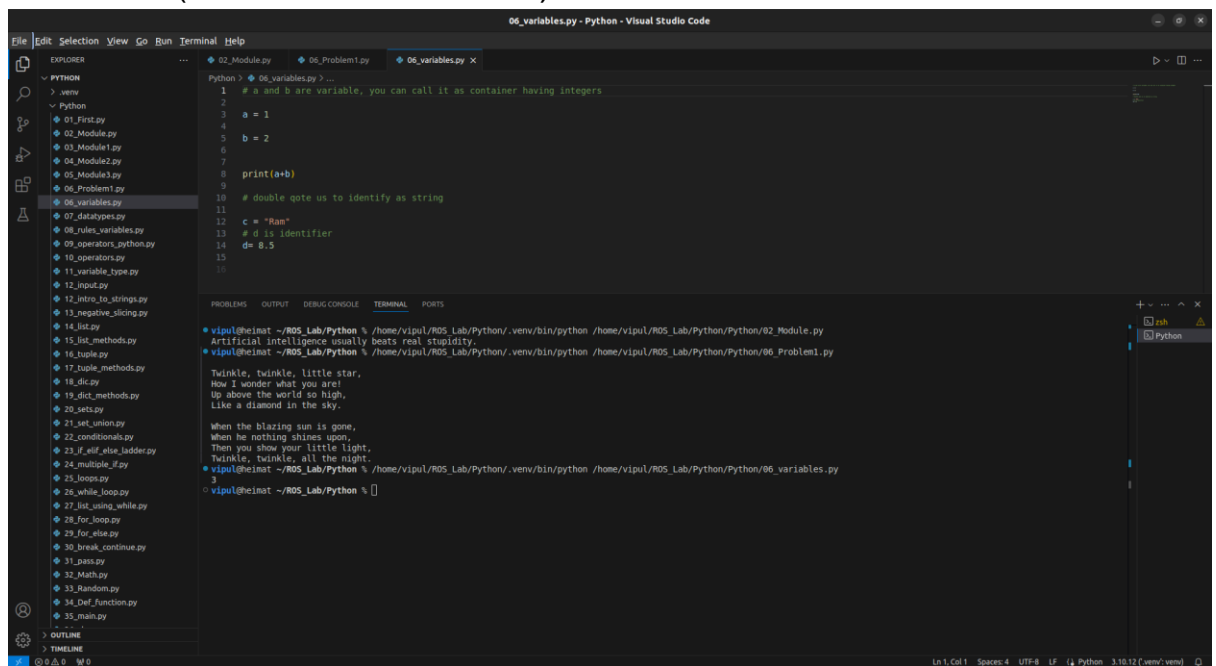
The screenshot shows the Visual Studio Code editor with a file named `06_Problem1.py`. The code defines a docstring for a function that prints a poem. The docstring is a multiline string enclosed in triple single quotes. The terminal output shows the function being called, which prints the poem's text.

```
1 """
2 Twinkle, twinkle, little star,
3 How I wonder what you are!
4 Up above the world so high,
5 Like a diamond in the sky.
6
7 When the blazing sun is gone,
8 When he nothing shines upon,
9 Then you show your little light,
10 Twinkle, twinkle, all the night."""
11
12
13 # to print multiple lines either print each and every lines or put those in between in the triple single quote
14
15
```

Terminal Output:

```
• vipul@heinst ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/.venv/bin/python /home/vipul/ROS_Lab/Python/Python/06_Problem1.py
Artificial intelligence usually beats real stupidity.
Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.
When the blazing sun is gone,
When he nothing shines upon,
Then you show your little light,
Twinkle, twinkle, all the night.
• vipul@heinst ~/ROS_Lab/Python %
```

4. Variables (Identifiers and values)



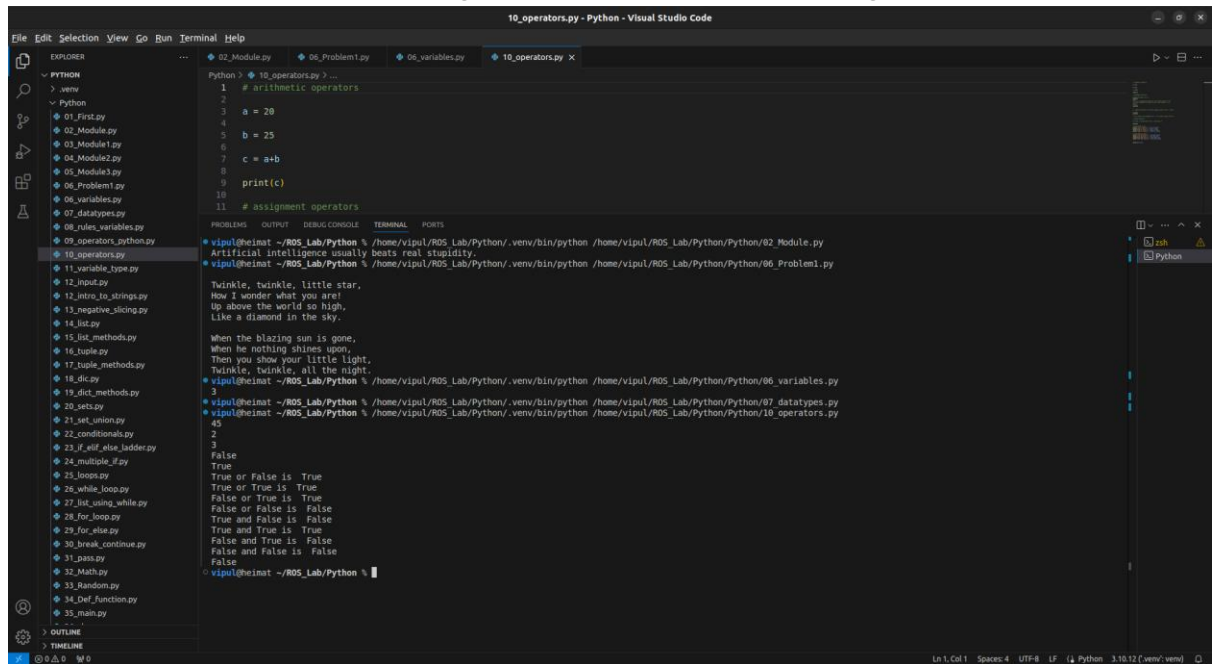
The screenshot shows the Visual Studio Code editor with a file named `06_variables.py`. The code demonstrates variable assignment and printing. It shows how to assign integer values to variables, how to assign a string value to a variable, and how to print the values of the variables.

```
1 # a and b are variable, you can call it as container having integers
2
3 a = 1
4
5 b = 2
6
7 print(a+b)
8
9
10 # double quote us to identify as string
11
12 c = "Ram"
13 # d is identifier
14 d = 8.5
15
16
```

Terminal Output:

```
• vipul@heinst ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/.venv/bin/python /home/vipul/ROS_Lab/Python/Python/06_variables.py
Artificial intelligence usually beats real stupidity.
Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.
When the blazing sun is gone,
When he nothing shines upon,
Then you show your little light,
Twinkle, twinkle, all the night.
• vipul@heinst ~/ROS_Lab/Python %
```

5. Operators (Arithmetic, Assignment, Comparison, Logical)

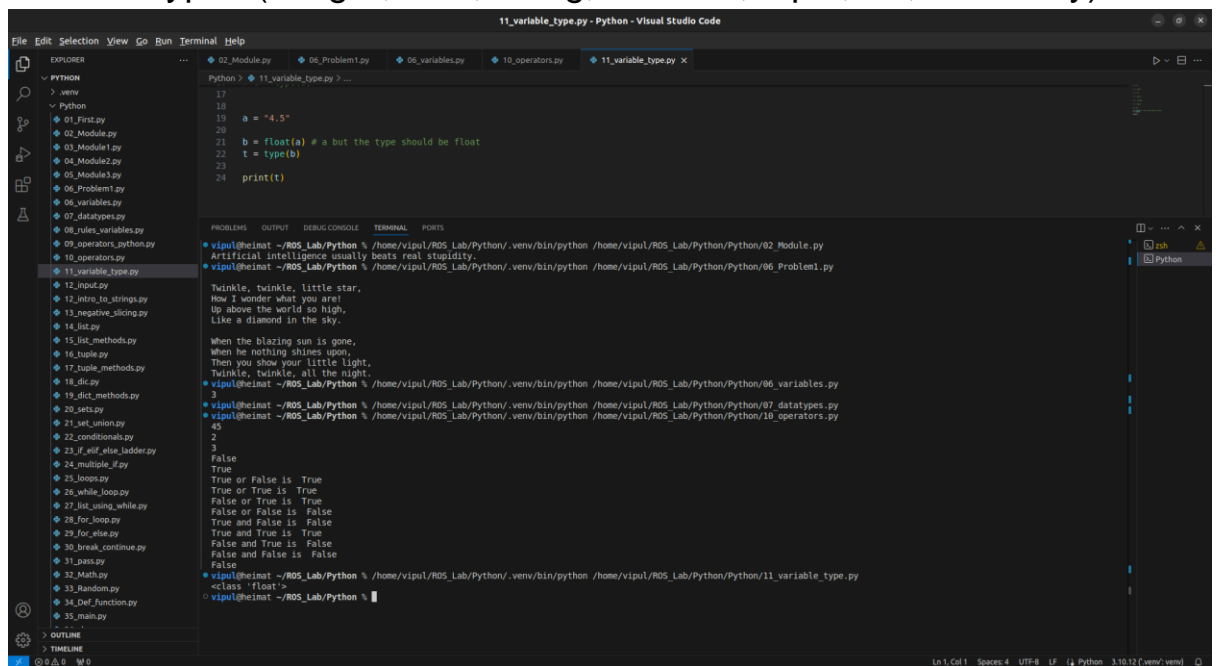


```
10_operators.py - Python - Visual Studio Code

Python 3.10.12 [tags: Python 3.10.12]

1 # arithmetic operators
2
3 a = 20
4
5 b = 25
6
7 c = a+b
8
9 print(c)
10
11 # assignment operators
12
13 Artificial intelligence usually beats real stupidity.
14
15 Twinkle, twinkle, little star,
16 How I wonder what you are!
17 Up above the world so high,
18 Like a diamond in the sky.
19
20 When the blazing sun is gone,
21 When he nothing shines upon,
22 Then you show your little light,
23 Twinkle, twinkle, all the night.
24
25 True or False is True
26 True or True is True
27 False or True is True
28 False or False is False
29 True and False is False
30 True and True is True
31 False and True is False
32 False and False is False
33
34 <class 'float'>
```

6. Variable types (integer, float, string, boolean, tuple, list, dictionary)

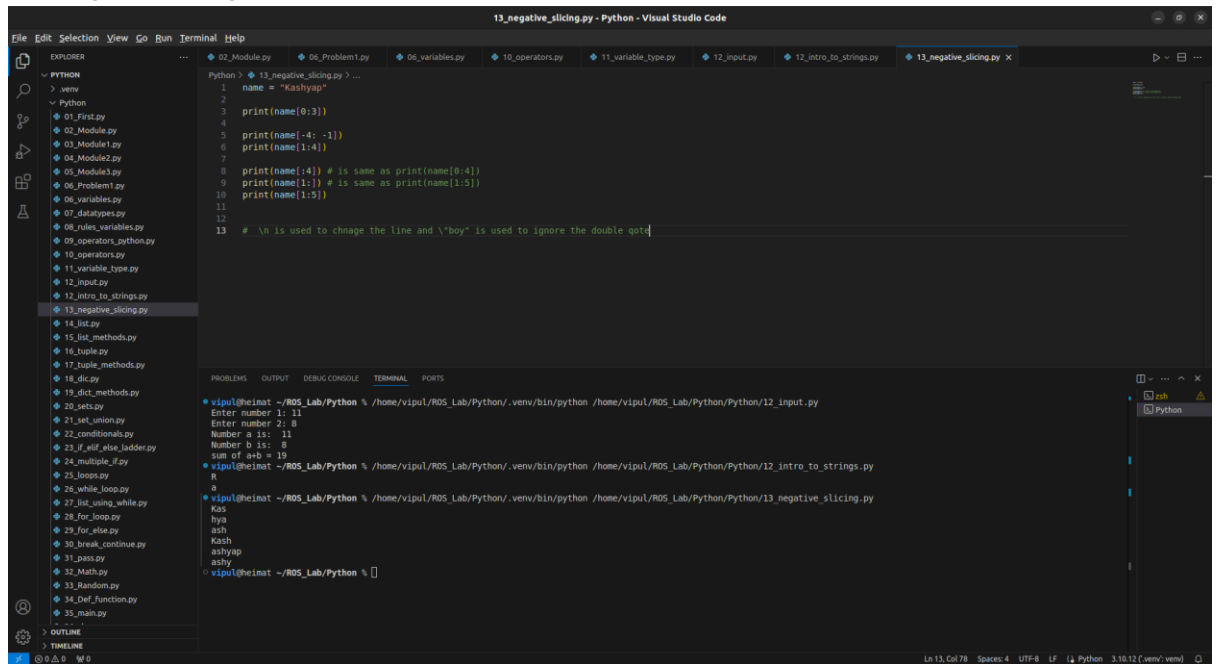


```
11_variable_type.py - Python - Visual Studio Code

Python 3.10.12 [tags: Python 3.10.12]

17
18
19 a = "4.5"
20
21 b = float(a) # a but the type should be float
22 t = type(b)
23
24 print(t)
25
26 Twinkle, twinkle, little star,
27 How I wonder what you are!
28 Up above the world so high,
29 Like a diamond in the sky.
30
31 When the blazing sun is gone,
32 When he nothing shines upon,
33 Then you show your little light,
34 Twinkle, twinkle, all the night.
35
36 True or False is True
37 True or True is True
38 False or True is True
39 False or False is False
40 True and False is False
41 True and True is True
42 False and True is False
43 False and False is False
44
45 <class 'float'>
```

7. String Parsing



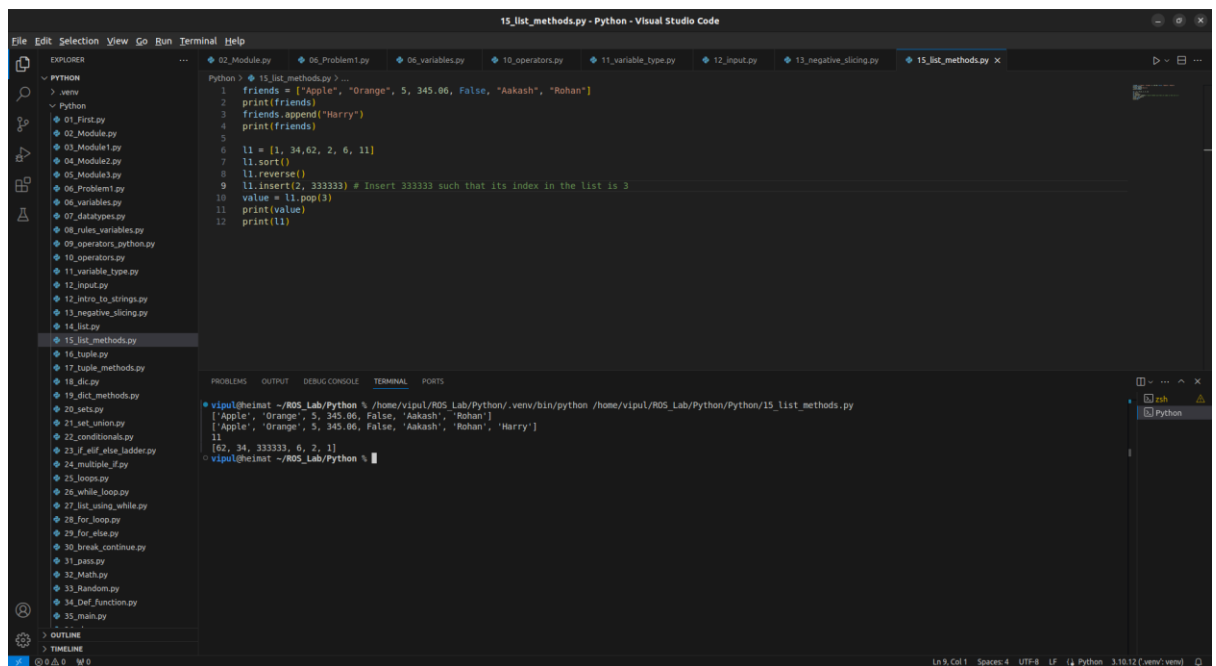
The screenshot shows a Visual Studio Code editor with a Python file named `13_negative_slicing.py`. The code demonstrates various string slicing techniques on the string `"Kashyap"`. The terminal output shows the execution results, including the original string, slices from index 0 to 3, from -4 to -1, and from 1 to 4. It also shows that slicing from 4 to the end is equivalent to slicing from 0 to 4, and that slicing from 1 to 5 is equivalent to slicing from the start to 5. A comment at the end explains that `\n` is used to change the line and `"boy"` is used to ignore the double quote.

```
1 name = "Kashyap"
2
3 print(name[0:3])
4
5 print(name[-4: -1])
6 print(name[1:4])
7
8 print(name[4:]) # is same as print(name[0:4])
9 print(name[1:]) # is same as print(name[1:5])
10 print(name[1:5])
11
12
13 # \n is used to chnage the line and \"boy\" is used to ignore the double quot
```

Terminal Output:

```
Python > 13_negative_slicing.py
Enter number 1: 11
Enter number 2: 8
Number a is: 11
Number b is: 8
sum of a+b = 19
vipu@heimat ~/R05_Lab/Python % /home/vipu/R05_Lab/Python/venv/bin/python /home/vipu/R05_Lab/Python/Python/12_intro_to_strings.py
R
Kas
hya
ash
Kash
ashyp
ashy
vipu@heimat ~/R05_Lab/Python %
```

8. List Methods



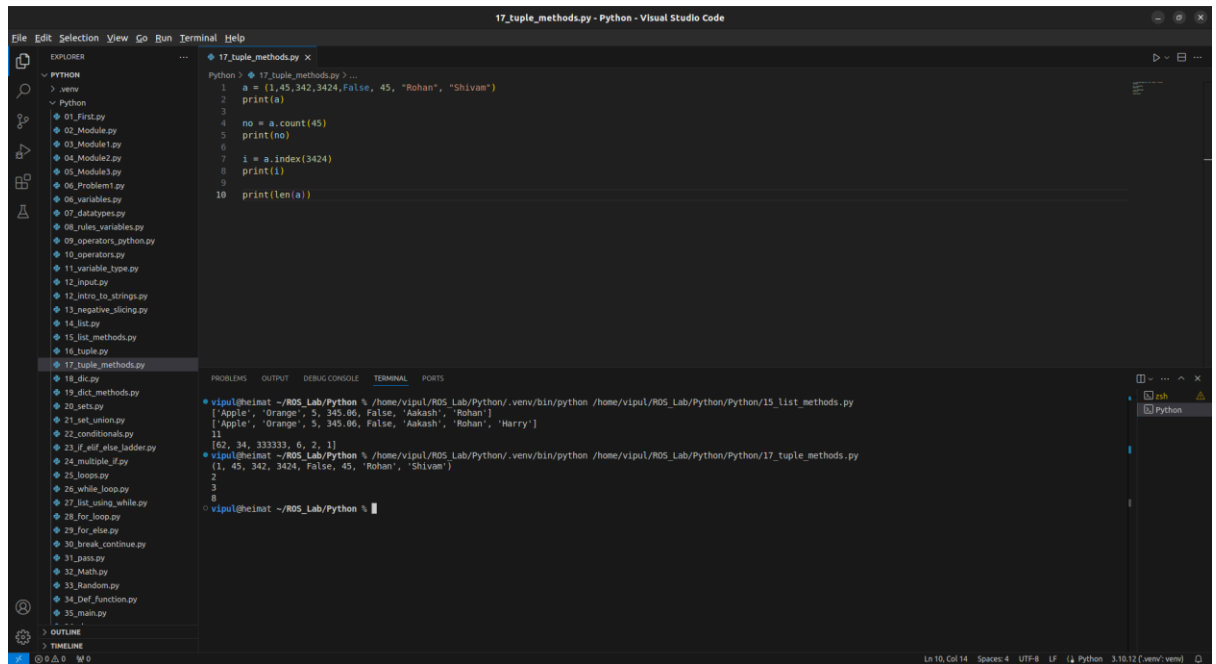
The screenshot shows a Visual Studio Code editor with a Python file named `15_list_methods.py`. The code demonstrates various list methods on a list of friends and a list of numbers. The terminal output shows the original list, the list after adding 'Harry', the sorted list, the list after reversing, and the list after inserting '333333' at index 2. It also shows the value of the element at index 3 after popping it.

```
1 friends = ['Apple', 'Orange', 5, 345.06, False, 'Aakash', 'Rohan']
2 print(friends)
3 friends.append('Harry')
4 print(friends)
5
6 ll = [1, 34, 62, 2, 6, 11]
7 ll.sort()
8 ll.reverse()
9 ll.insert(2, 333333) # Insert 333333 such that its index in the list is 3
10 value = ll.pop(3)
11 print(value)
12 print(ll)
```

Terminal Output:

```
Python > 15_list_methods.py
['Apple', 'Orange', 5, 345.06, False, 'Aakash', 'Rohan']
['Apple', 'Orange', 5, 345.06, False, 'Aakash', 'Rohan', 'Harry']
ll
[62, 34, 333333, 6, 2, 1]
vipu@heimat ~/R05_Lab/Python %
```

9. Tuple Methods



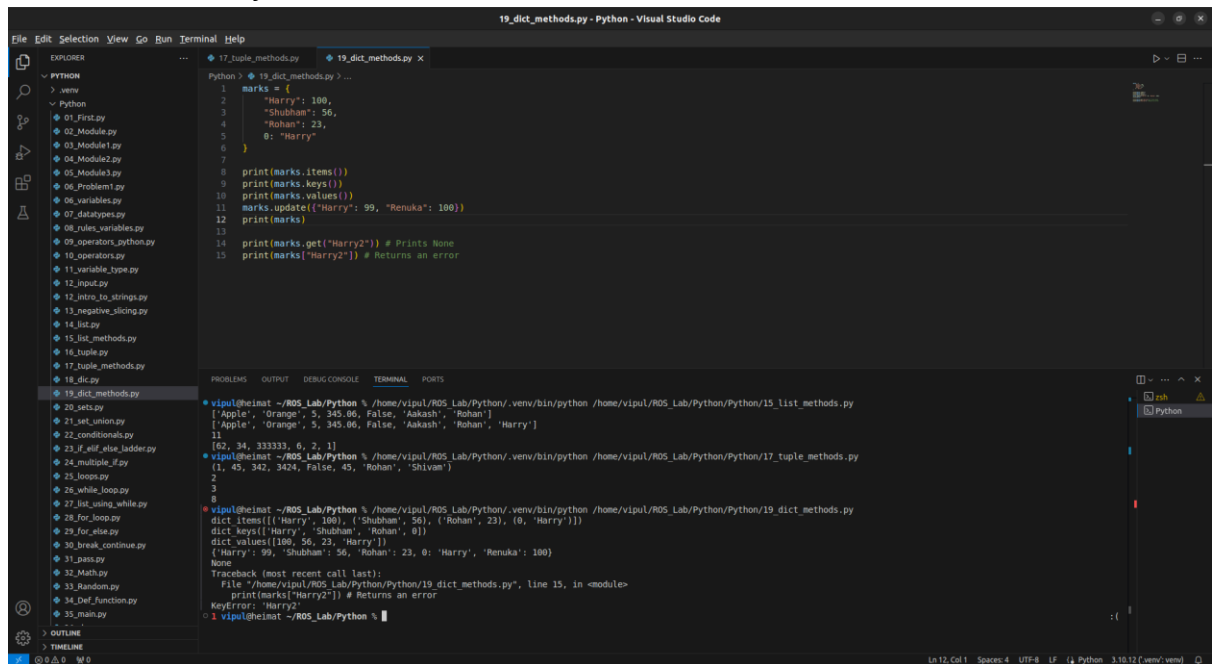
The screenshot shows a Visual Studio Code editor with a file named `17_tuple_methods.py`. The code defines a tuple `a` and performs several operations on it:

```
1 a = (1, 45, 342, 3424, False, 45, "Rohan", "Shivam")
2 print(a)
3
4 no = a.count(45)
5 print(no)
6
7 i = a.index(3424)
8 print(i)
9
10 print(len(a))
```

The terminal output shows the execution results:

```
vipul@heimat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/venv/bin/python /home/vipul/ROS_Lab/Python/Python/15_list_methods.py
['Apple', 'Orange', 5, 345.06, False, 'Aakash', 'Rohan']
['Apple', 'Orange', 5, 345.06, False, 'Aakash', 'Rohan', 'Harry']
11
[62, 34, 33333, 6, 2, 1]
vipul@heimat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/venv/bin/python /home/vipul/ROS_Lab/Python/Python/17_tuple_methods.py
(1, 45, 342, 3424, False, 45, 'Rohan', 'Shivam')
2
3
8
vipul@heimat ~/ROS_Lab/Python %
```

10. Dictionary Methods



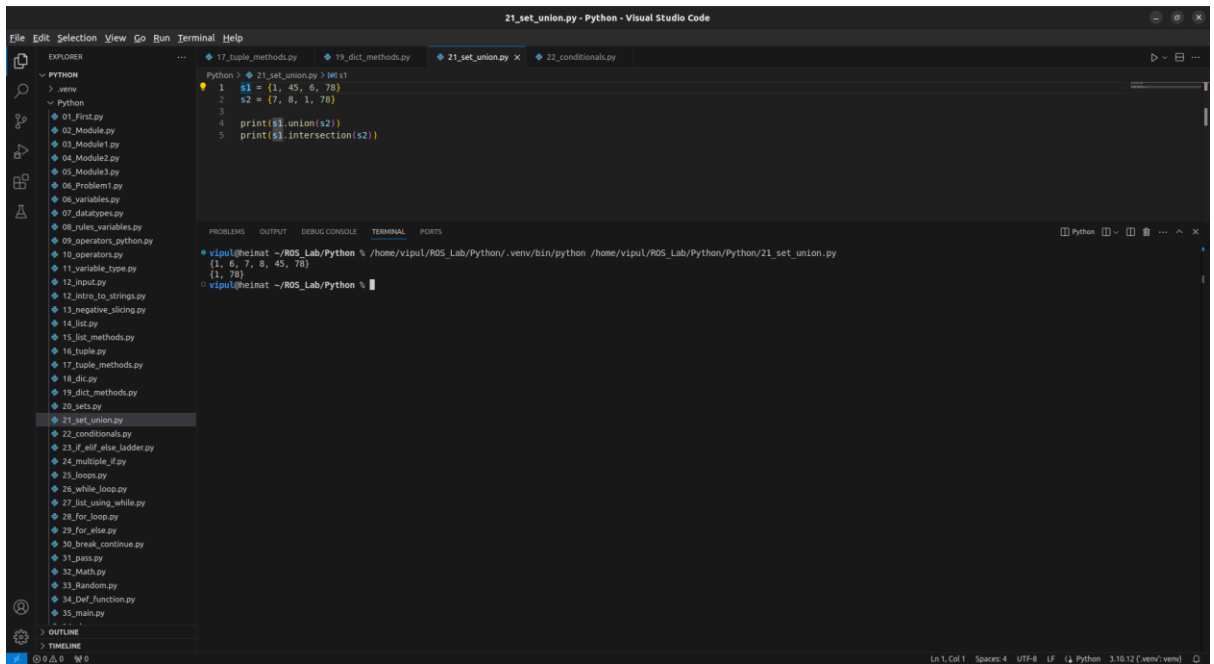
The screenshot shows a Visual Studio Code editor with a file named `19_dict_methods.py`. The code defines a dictionary `marks` and performs several operations on it:

```
1 marks = {
2     "Harry": 100,
3     "Shubham": 56,
4     "Rohan": 23,
5     0: "Harry"
6 }
7
8 print(marks.items())
9 print(marks.keys())
10 print(marks.values())
11 marks.update({"Harry": 99, "Renuka": 100})
12 print(marks)
13
14 print(marks.get("Harry2")) # Prints None
15 print(marks["Harry2"]) # Returns an error
```

The terminal output shows the execution results:

```
vipul@heimat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/venv/bin/python /home/vipul/ROS_Lab/Python/Python/15_list_methods.py
['Apple', 'Orange', 5, 345.06, False, 'Aakash', 'Rohan']
['Apple', 'Orange', 5, 345.06, False, 'Aakash', 'Rohan', 'Harry']
11
[62, 34, 33333, 6, 2, 1]
vipul@heimat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/venv/bin/python /home/vipul/ROS_Lab/Python/Python/17_tuple_methods.py
(1, 45, 342, 3424, False, 45, 'Rohan', 'Shivam')
2
3
8
vipul@heimat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/venv/bin/python /home/vipul/ROS_Lab/Python/Python/19_dict_methods.py
dict_items([('Harry', 100), ('Shubham', 56), ('Rohan', 23), (0, 'Harry')])
dict_keys(['Harry', 'Shubham', 'Rohan', 0])
dict_values([100, 56, 23, 'Harry'])
{'Harry': 99, 'Shubham': 56, 'Rohan': 23, 0: 'Harry', 'Renuka': 100}
None
Traceback (most recent call last):
  File "/home/vipul/ROS_Lab/Python/Python/19_dict_methods.py", line 15, in <module>
    print(marks["Harry2"]) # Returns an error
KeyError: 'Harry2'
vipul@heimat ~/ROS_Lab/Python %
```

11. Set Methods (Union, Intersection)

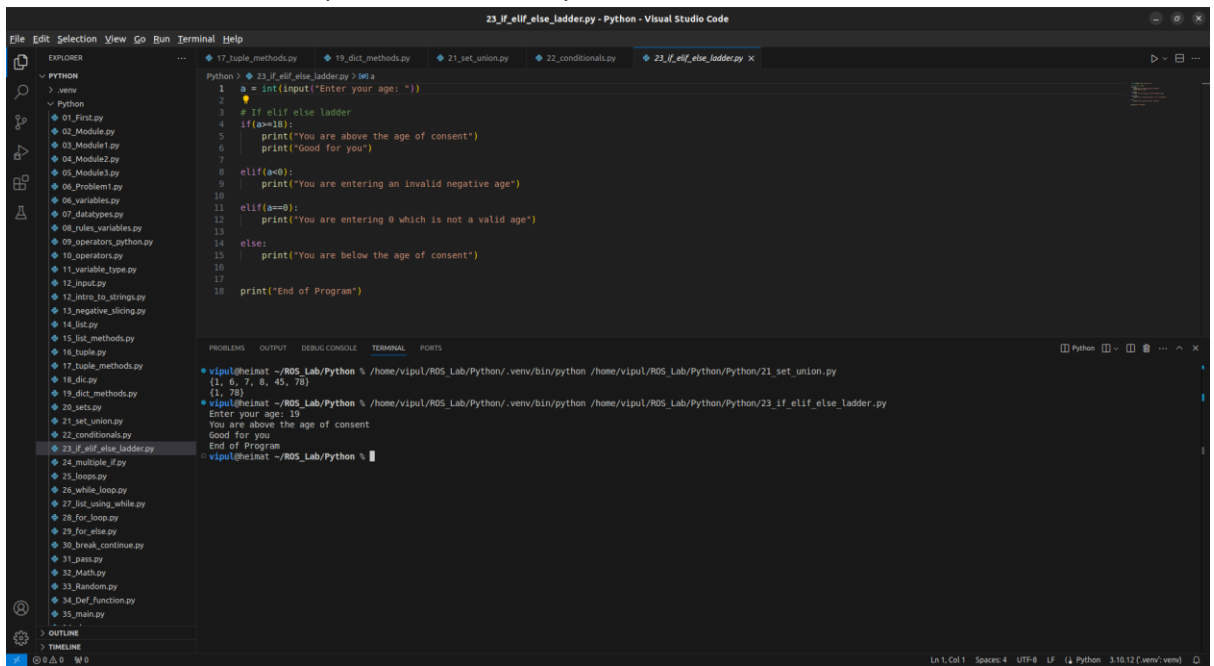


The screenshot shows the Visual Studio Code editor with a file named `21_set_union.py` open. The code defines two sets, `s1` and `s2`, and prints their union and intersection. The terminal output shows the results of these operations.

```
Python > 21_set_union.py > set1
1 s1 = {1, 45, 6, 78}
2 s2 = {7, 8, 1, 78}
3
4 print(s1.union(s2))
5 print(s1.intersection(s2))
```

```
Python > 21_set_union.py % /home/vipul/RDS_Lab/Python/.venv/bin/python /home/vipul/RDS_Lab/Python/21_set_union.py
{1, 6, 7, 8, 45, 78}
{1, 78}
```

12. Conditionals (Else-If Ladder)



The screenshot shows the Visual Studio Code editor with a file named `23_if_elif_else_ladder.py` open. The code uses an else-if ladder to check the age of a user and print a message. The terminal output shows the user's input and the resulting message.

```
Python > 23_if_elif_else_ladder.py > let a
1 a = int(input("Enter your age: "))
2
3 # if elif else ladder
4 if(a>18):
5     print("You are above the age of consent")
6     print("Good for you")
7
8 elif(a<0):
9     print("You are entering an invalid negative age")
10
11 elif(a==0):
12     print("You are entering 0 which is not a valid age")
13
14 else:
15     print("You are below the age of consent")
16
17 print("End of Program")
```

```
Python > 23_if_elif_else_ladder.py % /home/vipul/RDS_Lab/Python/.venv/bin/python /home/vipul/RDS_Lab/Python/23_if_elif_else_ladder.py
Enter your age: 19
You are above the age of consent
Good for you
End of Program
```

13. While Loop (Iterating over a list)

The screenshot shows the Visual Studio Code interface with a file explorer on the left containing various Python files. The main editor displays a file named `27_list_using_while.py` with the following code:

```
1 t = ["Harry", False, "This", "Rohan", "Shubham", "Shubhi"]
2
3 i = 0
4
5 while(i < len(t)):
6     print(t[i])
7     i += 1
8
```

The bottom panel shows the terminal output:

```
vipul@heinat ~/RDS_Lab/Python % /home/vipul/RDS_Lab/Python/venv/bin/python /home/vipul/RDS_Lab/Python/Python/27_list_using_while.py
1
Harry
False
This
Rohan
Shubham
Shubhi
vipul@heinat ~/RDS_Lab/Python %
```

14. For else Loop

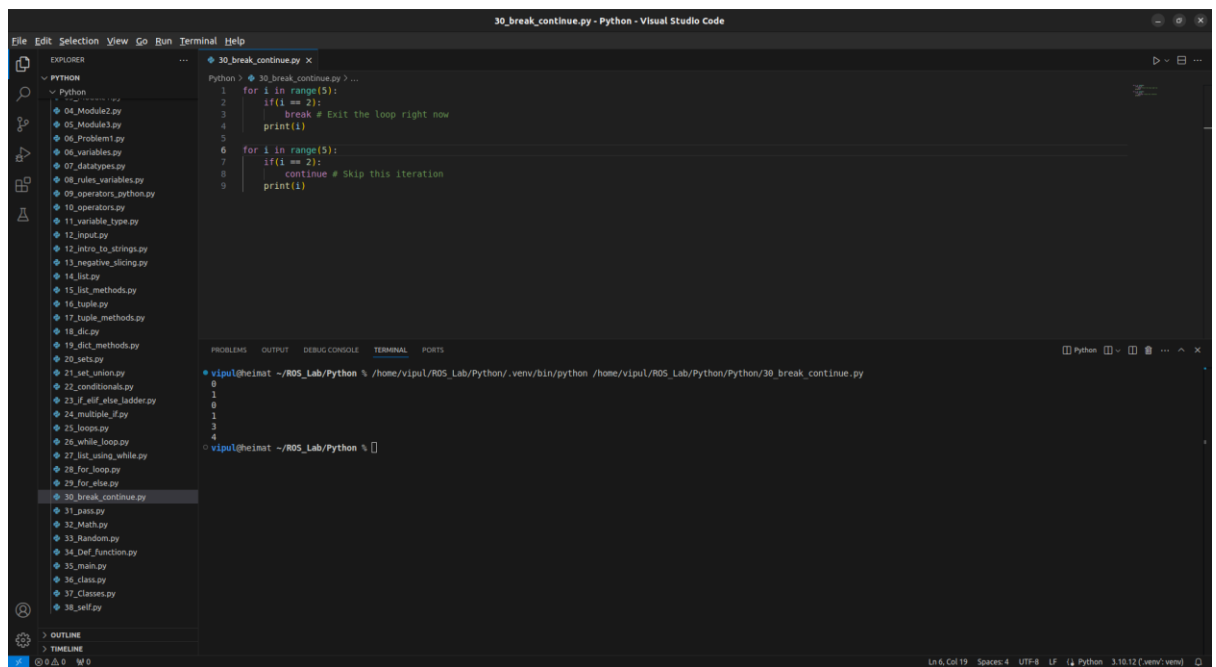
The screenshot shows the Visual Studio Code interface with a file explorer on the left. The main editor displays a file named `29_for_else.py` with the following code:

```
1 l = [1,7,8]
2
3 for item in l:
4     print(item)
5
6 else:
7     print("done") # this is printed when the loop exhausts!
```

The bottom panel shows the terminal output:

```
vipul@heinat ~/RDS_Lab/Python % /home/vipul/RDS_Lab/Python/venv/bin/python /home/vipul/RDS_Lab/Python/Python/29_for_else.py
1
7
8
done
vipul@heinat ~/RDS_Lab/Python %
```

15. Break/Continue



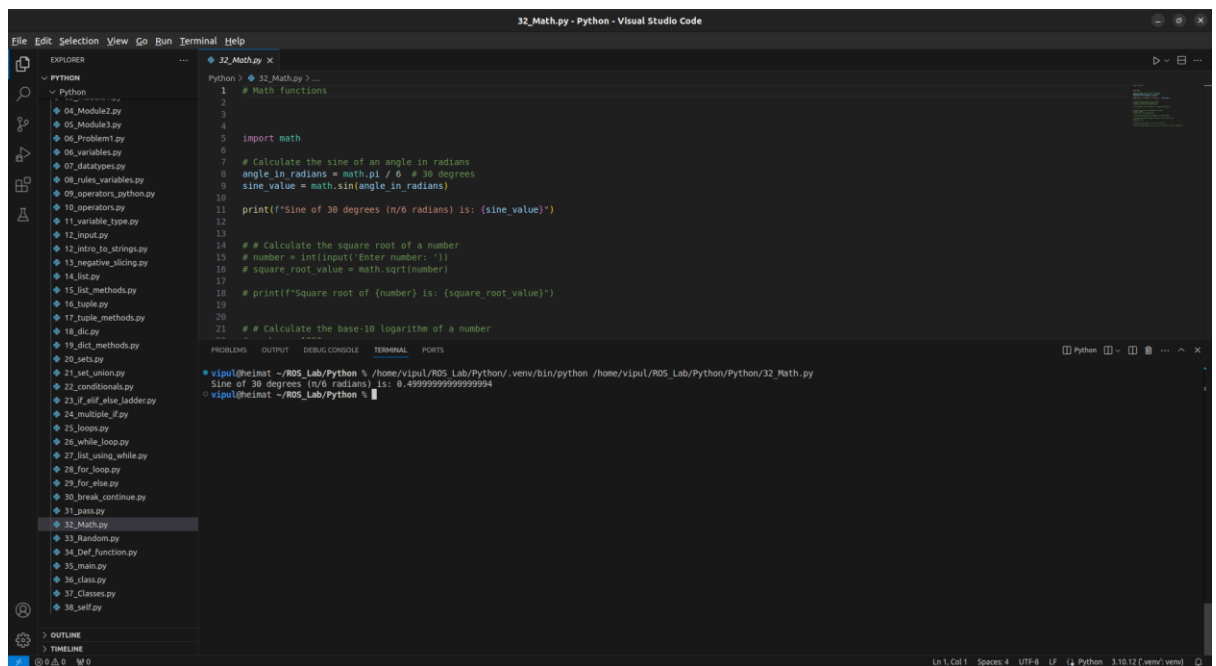
The screenshot shows the Visual Studio Code interface with a Python file named `30_break_continue.py`. The code is as follows:

```
1 for i in range(5):
2     if i == 2:
3         break # Exit the loop right now
4     print(i)
5
6 for i in range(5):
7     if i == 2:
8         continue # Skip this iteration
9     print(i)
```

The left sidebar shows a file explorer with a list of Python files, including `30_break_continue.py`. The bottom panel shows the terminal output:

```
• vipul@heimat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/venv/bin/python /home/vipul/ROS_Lab/Python/Python/30_break_continue.py
0
1
2
3
4
```

16. Math Module



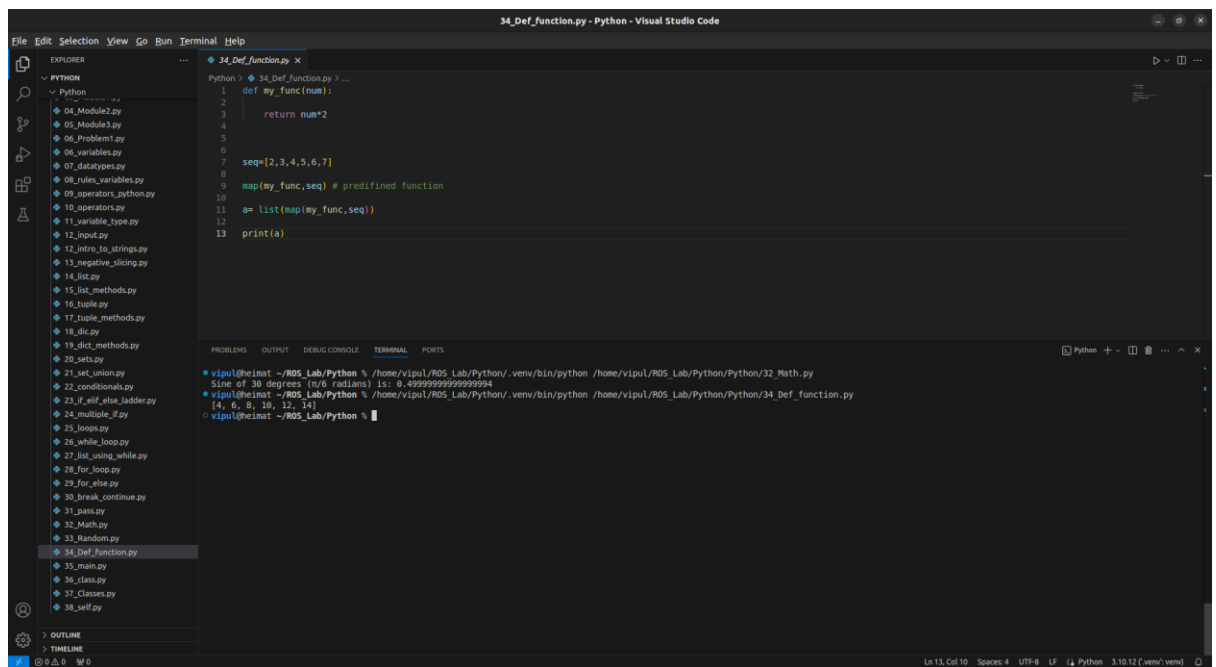
The screenshot shows the Visual Studio Code interface with a Python file named `32_Math.py`. The code is as follows:

```
1 # Math functions
2
3
4
5 import math
6
7 # Calculate the sine of an angle in radians
8 angle_in_radians = math.pi / 6 # 30 degrees
9 sine_value = math.sin(angle_in_radians)
10
11 print(f"Sine of 30 degrees (pi/6 radians) is: {sine_value}")
12
13
14 ## Calculate the square root of a number
15 # number = int(input("Enter number: "))
16 # square_root_value = math.sqrt(number)
17
18 # print(f"Square root of (number) is: {square_root_value}")
19
20
21 ## Calculate the base-10 logarithm of a number
22
```

The left sidebar shows a file explorer with a list of Python files, including `32_Math.py`. The bottom panel shows the terminal output:

```
• vipul@heimat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/venv/bin/python /home/vipul/ROS_Lab/Python/Python/32_Math.py
Sine of 30 degrees (pi/6 radians) is: 0.49999999999999994
• vipul@heimat ~/ROS_Lab/Python %
```


17. Functions

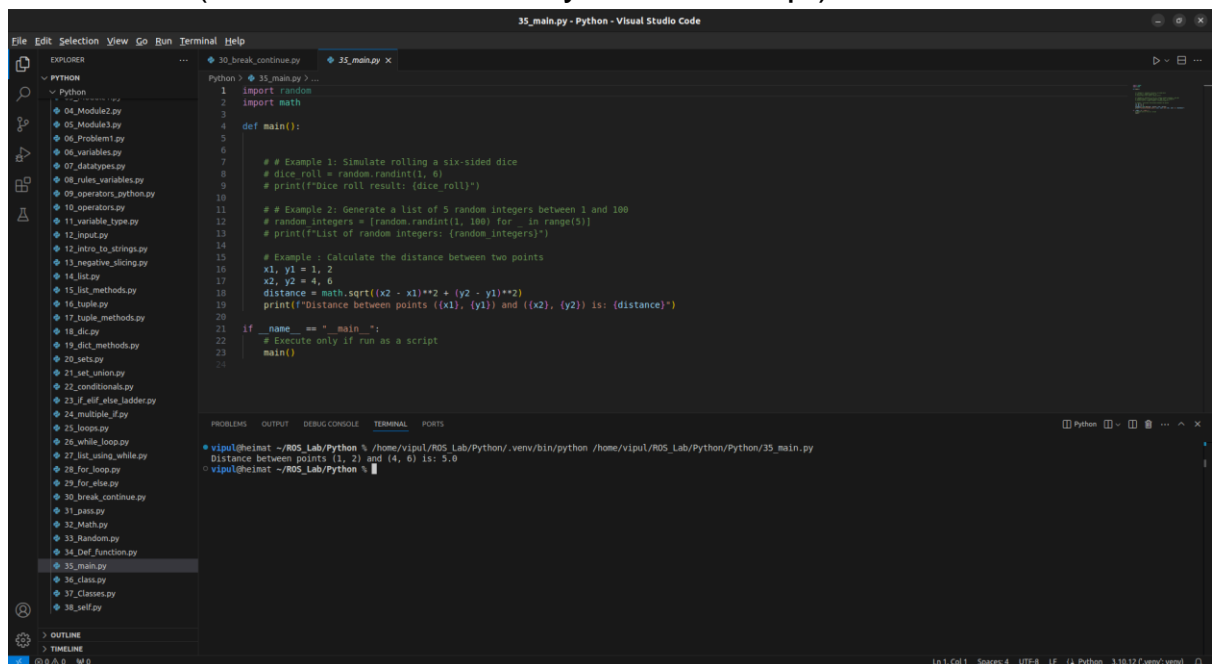


```
def my_func(num):  
    return num*2  
  
seq=[2,3,4,5,6,7]  
map(my_func,seq) # predefined function  
a= list(map(my_func,seq))  
print(a)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

• vipul@heinat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/.venv/bin/python /home/vipul/ROS_Lab/Python/Python/32_Math.py
Sine of 30 degrees (π/6 radians) is: 0.49999999999999994
• vipul@heinat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/.venv/bin/python /home/vipul/ROS_Lab/Python/34_Def_function.py
[4, 6, 8, 10, 12, 14]
• vipul@heinat ~/ROS_Lab/Python %

18. Main (Will not run if called by another script)

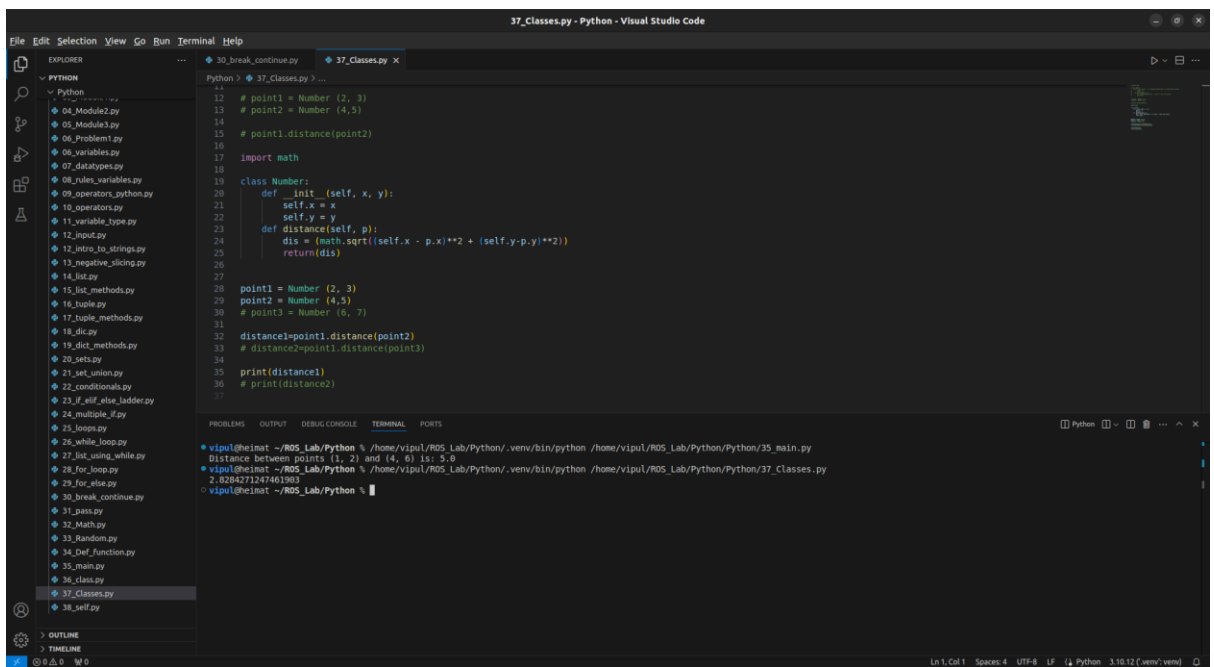


```
import random  
import math  
  
def main():  
  
    # Example 1: Simulates rolling a six-sided dice  
    # dice roll = random.randint(1, 6)  
    # print("Dice roll result: (dice roll)")  
  
    # Example 2: Generate a List of 5 random integers between 1 and 100  
    # random integers = [random.randint(1, 100) for _ in range(5)]  
    # print("List of random integers: (random integers)")  
  
    # Example 3: Calculate the distance between two points  
    x1, y1 = 1, 2  
    x2, y2 = 4, 6  
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)  
    print("Distance between points ((x1), (y1)) and ((x2), (y2)) is: (distance)")  
  
if __name__ == "__main__":  
    # Execute only if run as a script  
    main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

• vipul@heinat ~/ROS_Lab/Python % /home/vipul/ROS_Lab/Python/.venv/bin/python /home/vipul/ROS_Lab/Python/Python/35_main.py
Distance between points (1, 2) and (4, 6) is: 5.0
• vipul@heinat ~/ROS_Lab/Python %

19. Classes

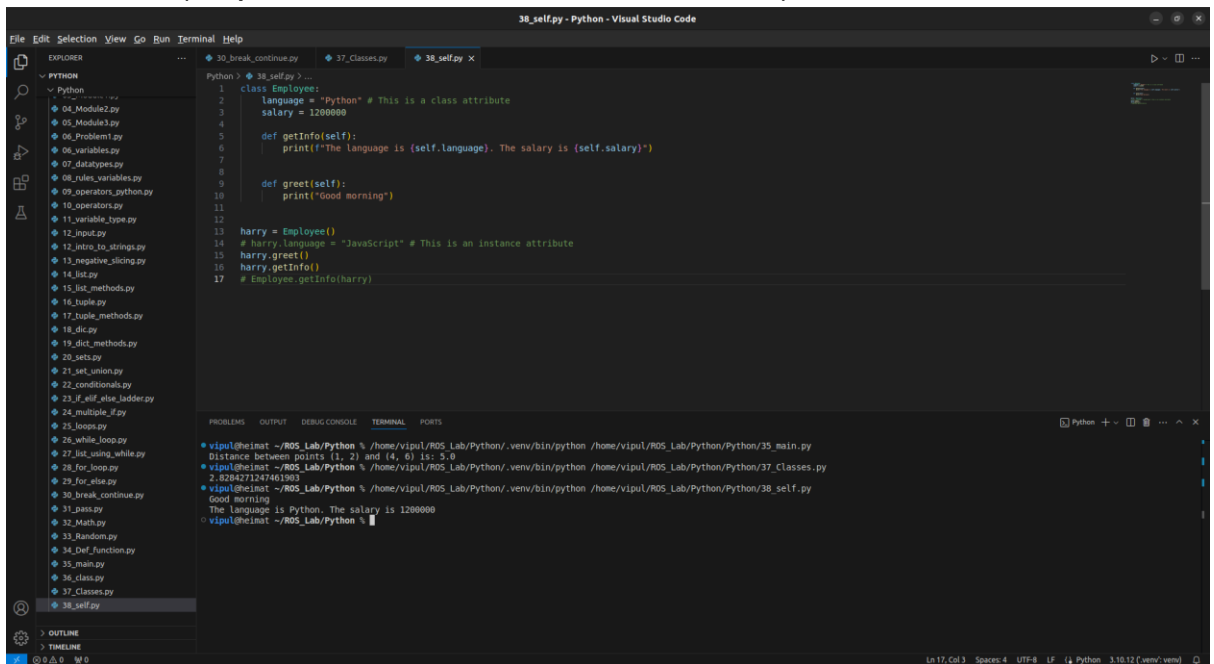


The screenshot shows a Visual Studio Code editor with a Python file named `37_Classes.py`. The code defines a `Number` class with an `__init__` method and a `distance` method. It then creates three instances of the class: `point1`, `point2`, and `point3`. The `distance` method is called on `point1` and `point2` to calculate the distance between them.

```
11
12 # point1 = Number (2, 3)
13 # point2 = Number (4,5)
14
15 # point1.distance(point2)
16
17 import math
18
19 class Number:
20     def __init__(self, x, y):
21         self.x = x
22         self.y = y
23     def distance(self, p):
24         dis = (math.sqrt((self.x - p.x)**2 + (self.y-p.y)**2))
25         return(dis)
26
27
28 point1 = Number (2, 3)
29 point2 = Number (4,5)
30 # point3 = Number (6, 7)
31
32 distance=point1.distance(point2)
33 # distance2=point1.distance(point3)
34
35 print(distance)
36 # print(distance2)
37
```

The terminal output shows the execution of the script, displaying the distance between points (1, 2) and (4, 6) as 5.0.

20. Self (Represents an instance of the class)



The screenshot shows a Visual Studio Code editor with a Python file named `38_self.py`. The code defines an `Employee` class with class attributes `language` and `salary`, and methods `getInfo` and `greet`. It then creates an instance of the class named `harry` and calls the `greet` and `getInfo` methods on it.

```
1 class Employee:
2     language = "Python" # This is a class attribute
3     salary = 1200000
4
5     def getInfo(self):
6         print(f"The language is {self.language}. The salary is {self.salary}")
7
8     def greet(self):
9         print("Good morning")
10
11
12 harry = Employee()
13 # harry.language = "JavaScript" # This is an instance attribute
14 harry.greet()
15 harry.getInfo()
16 # Employee.getInfo(harry)
17
```

The terminal output shows the execution of the script, displaying the language and salary of the `harry` instance.