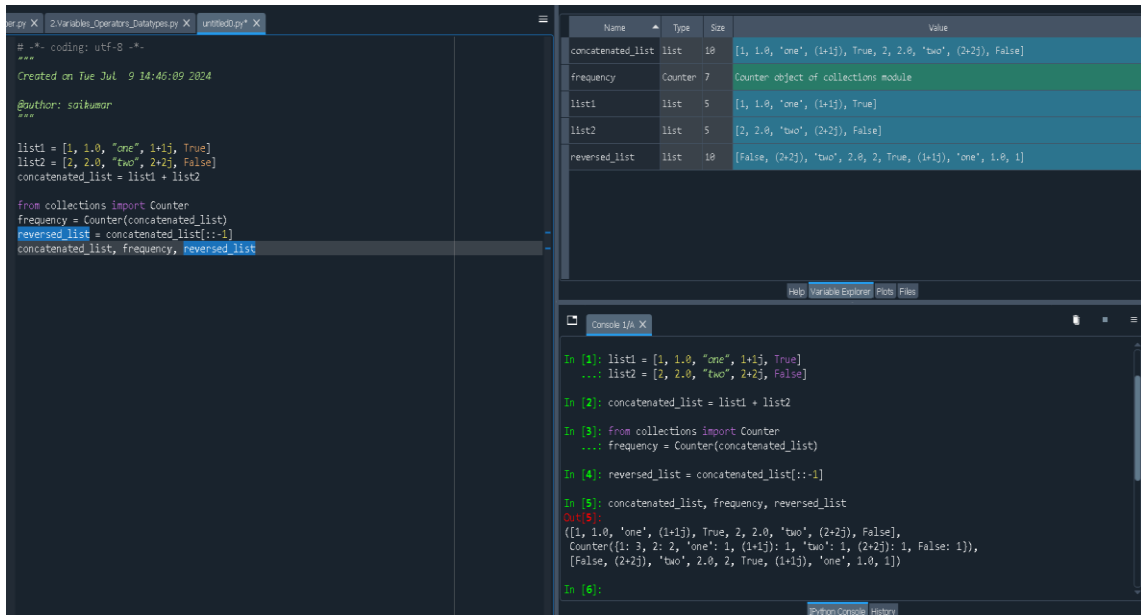


## Module – 2 ASSIGNMENT

### Data Types

Please implement it by using Python.

1. Construct 2 lists containing all the available data types (integer, float, string, complex and Boolean) and do the following..
  - a. Create another list by concatenating above 2 lists
  - b. Find the frequency of each element in the concatenated list.
  - c. Print the list in reverse order.



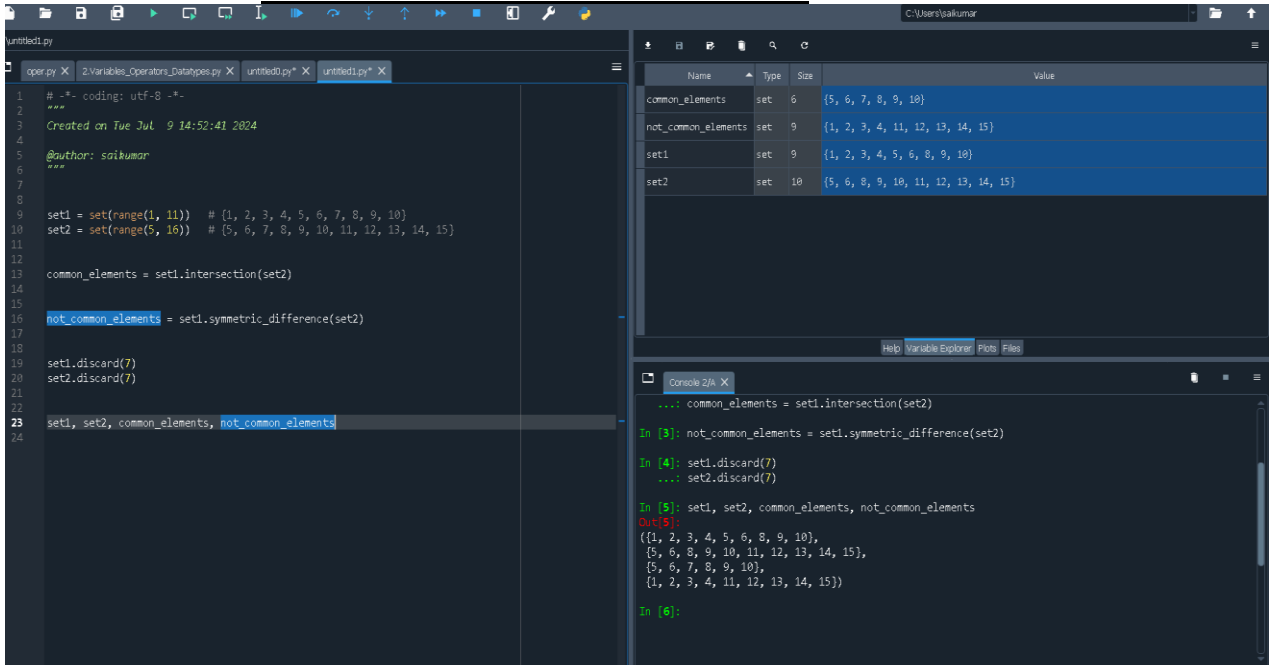
The screenshot shows a Python IDE with a code editor on the left and a variable explorer on the right. The code in the editor creates two lists, concatenates them, finds the frequency of each element using the Counter class, and prints the concatenated list in reverse order. The variable explorer on the right shows the state of the variables: concatenated\_list (list, size 10), frequency (Counter, size 7), list1 (list, size 5), list2 (list, size 5), and reversed\_list (list, size 10).

```
# -*- coding: utf-8 -*-  
"""  
Created on Tue Jul 9 14:46:09 2024  
  
@author: saikumar  
"""  
  
list1 = [1, 1.0, "one", 1+1j, True]  
list2 = [2, 2.0, "two", 2+2j, False]  
concatenated_list = list1 + list2  
  
from collections import Counter  
frequency = Counter(concatenated_list)  
reversed_list = concatenated_list[::-1]  
concatenated_list, frequency, reversed_list
```

Name	Type	Size	Value
concatenated_list	list	10	[1, 1.0, 'one', (1+1j), True, 2, 2.0, 'two', (2+2j), False]
frequency	Counter	7	Counter object of collections module
list1	list	5	[1, 1.0, 'one', (1+1j), True]
list2	list	5	[2, 2.0, 'two', (2+2j), False]
reversed_list	list	10	[False, (2+2j), 'two', 2.0, 2, True, (1+1j), 'one', 1.0, 1]

```
In [1]: list1 = [1, 1.0, "one", 1+1j, True]  
....: list2 = [2, 2.0, "two", 2+2j, False]  
  
In [2]: concatenated_list = list1 + list2  
  
In [3]: from collections import Counter  
....: frequency = Counter(concatenated_list)  
  
In [4]: reversed_list = concatenated_list[::-1]  
  
In [5]: concatenated_list, frequency, reversed_list  
Out[5]:  
([1, 1.0, 'one', (1+1j), True, 2, 2.0, 'two', (2+2j), False],  
 Counter({1: 3, 2: 2, 'one': 1, (1+1j): 1, 'two': 1, (2+2j): 1, False: 1}),  
 [False, (2+2j), 'two', 2.0, 2, True, (1+1j), 'one', 1.0, 1])  
  
In [6]:
```

2. Create 2 Sets containing integers (numbers from 1 to 10 in one set and 5 to 15 in other set)
  - a. Find the common elements in above 2 Sets.
  - b. Find the elements that are not common.
  - c. Remove element 7 from both the Sets.



```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jul 9 14:52:41 2024
4
5  @author: saikumar
6  """
7
8
9  set1 = set(range(1, 11)) # {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
10 set2 = set(range(5, 16)) # {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
11
12 common_elements = set1.intersection(set2)
13
14 not_common_elements = set1.symmetric_difference(set2)
15
16 set1.discard(7)
17 set2.discard(7)
18
19 set1, set2, common_elements, not_common_elements

```

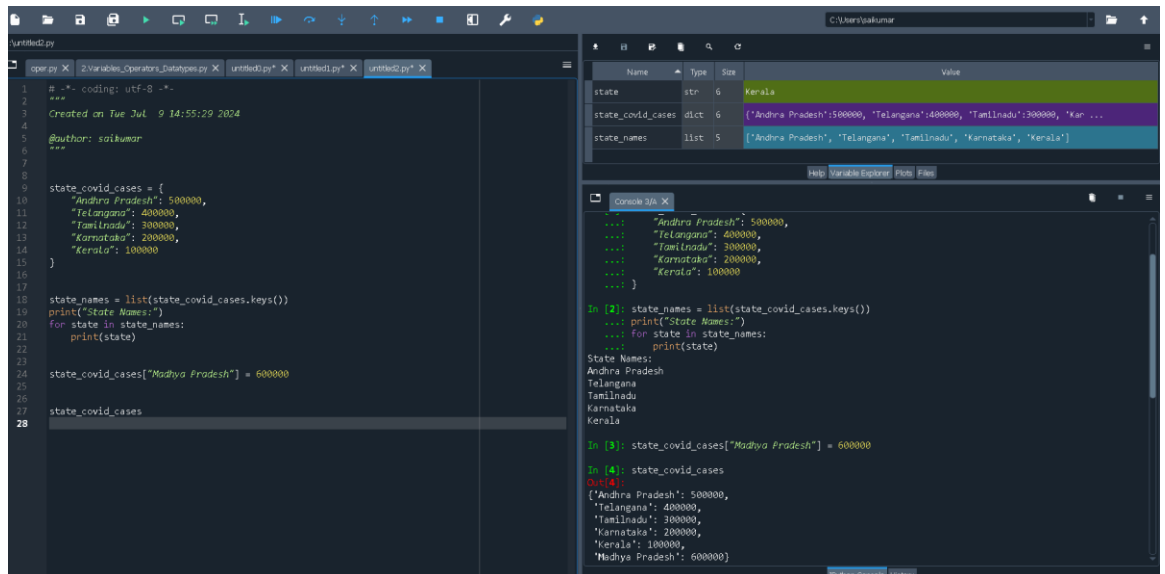
Name	Type	Size	Value
common_elements	set	6	{5, 6, 7, 8, 9, 10}
not_common_elements	set	9	{1, 2, 3, 4, 11, 12, 13, 14, 15}
set1	set	9	{1, 2, 3, 4, 5, 6, 8, 9, 10}
set2	set	10	{5, 6, 8, 9, 10, 11, 12, 13, 14, 15}

```

... common_elements = set1.intersection(set2)
In [3]: not_common_elements = set1.symmetric_difference(set2)
In [4]: set1.discard(7)
... set2.discard(7)
In [5]: set1, set2, common_elements, not_common_elements
Out[5]:
({1, 2, 3, 4, 5, 6, 8, 9, 10},
 {5, 6, 8, 9, 10, 11, 12, 13, 14, 15},
 {5, 6, 7, 8, 9, 10},
 {1, 2, 3, 4, 11, 12, 13, 14, 15})
In [6]:

```

3. Create a data dictionary of 5 states having state name as key and number of covid-19 cases as values.
  - a. Print only state names from the dictionary.
  - b. Update another country and its covid-19 cases in the dictionary.



```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jul 9 14:55:29 2024
4
5  @author: saikumar
6  """
7
8
9  state_covid_cases = {
10     "Andhra Pradesh": 500000,
11     "Telangana": 400000,
12     "Tamilnadu": 300000,
13     "Karnataka": 200000,
14     "Kerala": 100000
15 }
16
17 state_names = list(state_covid_cases.keys())
18 print("State Names:")
19 for state in state_names:
20     print(state)
21
22 state_covid_cases["Madhya Pradesh"] = 600000
23
24 state_covid_cases

```

Name	Type	Size	Value
state	str	6	Kerala
state_covid_cases	dict	6	{'Andhra Pradesh': 500000, 'Telangana': 400000, 'Tamilnadu': 300000, 'Kar ...
state_names	list	5	['Andhra Pradesh', 'Telangana', 'Tamilnadu', 'Karnataka', 'Kerala']

```

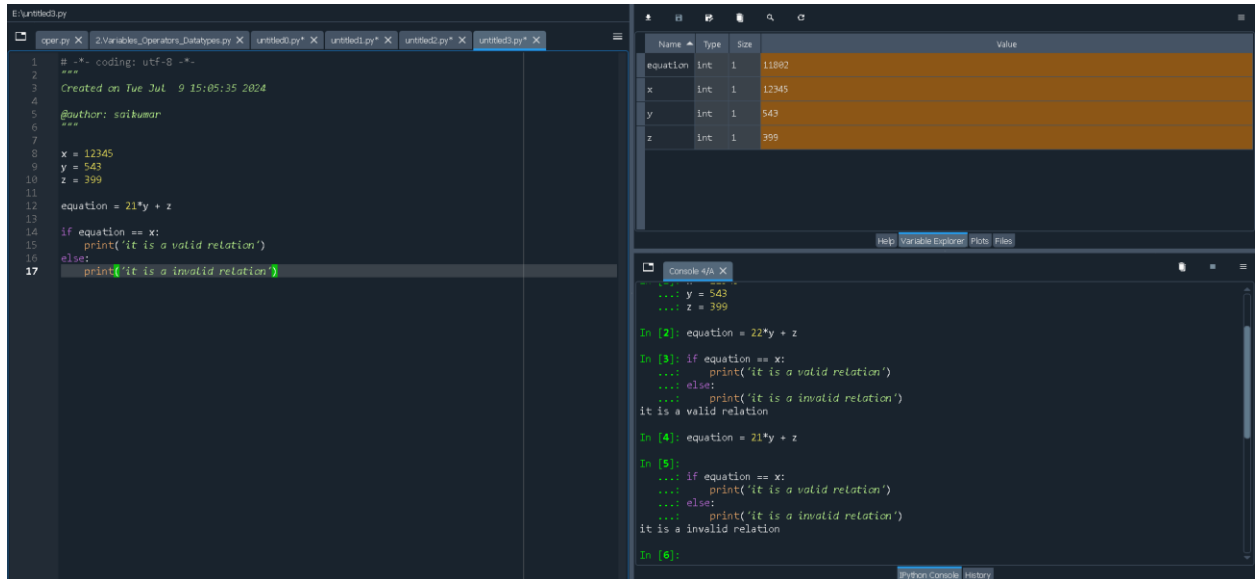
... "Andhra Pradesh": 500000,
... "Telangana": 400000,
... "Tamilnadu": 300000,
... "Karnataka": 200000,
... "Kerala": 100000
In [2]: state_names = list(state_covid_cases.keys())
... print("State Names:")
... for state in state_names:
...     print(state)
State Names:
Andhra Pradesh
Telangana
Tamilnadu
Karnataka
Kerala
In [3]: state_covid_cases["Madhya Pradesh"] = 600000
In [4]: state_covid_cases
Out[4]:
{'Andhra Pradesh': 500000,
 'Telangana': 400000,
 'Tamilnadu': 300000,
 'Karnataka': 200000,
 'Kerala': 100000,
 'Madhya Pradesh': 600000}

```

## Operators

Please implement by using Python

1. A. Write an equation which relates 399, 543 and 12345



```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Jul 9 15:05:35 2024
4
5 @author: saikumar
6 """
7
8 x = 12345
9 y = 543
10 z = 399
11
12 equation = 21*y + z
13
14 if equation == x:
15     print('it is a valid relation')
16 else:
17     print('it is a invalid relation')
```

Name	Type	Size	Value
equation	int	1	11802
x	int	1	12345
y	int	1	543
z	int	1	399

```
In [2]: equation = 21*y + z
In [3]: if equation == x:
...:     print('it is a valid relation')
...: else:
...:     print('it is a invalid relation')
it is a valid relation
In [4]: equation = 21*y + z
In [5]:
...: if equation == x:
...:     print('it is a valid relation')
...: else:
...:     print('it is a invalid relation')
it is a invalid relation
In [6]:
```

B. “When I divide 5 with 3, I get 1. But when I divide -5 with 3, I get -2”—How would you justify it?

In Python (and many other programming languages):

**1.Integer Division ('//'):** This operation returns the floor of the division, which means it rounds down to the nearest integer.

**2.Floor Division ('//'):** This operation performs division and rounds down to the nearest integer.

### 1. Dividing 5 by 3:

When you divide 5 by 3 in Python:

`5 // 3`

- Result: 1

This is because 5 divided by 3 is approximately 1.666..., and // operator rounds down to the nearest integer, which is 1.

## 2. Dividing -5 by 3:

When you divide -5 by 3 in Python:

```
-5 // 3
```

- Result: -2

This might seem counterintuitive at first glance. Here's the reasoning:

- -5 / 3 in decimal form is approximately -1.666....
- However, // operator rounds down towards negative infinity. This means it gives the largest integer less than or equal to the result of the division.

To understand why -5 // 3 gives -2:

- -1.666... rounds down to -2, as -2 is less than -1.666... and the nearest integer towards negative infinity.

2. a=5,b=3,c=10.. What will be the output of the following:

A. a/=b

```
a = a / b
```

```
a = 5 / 3
```

```
a = 1.6666666666666667
```

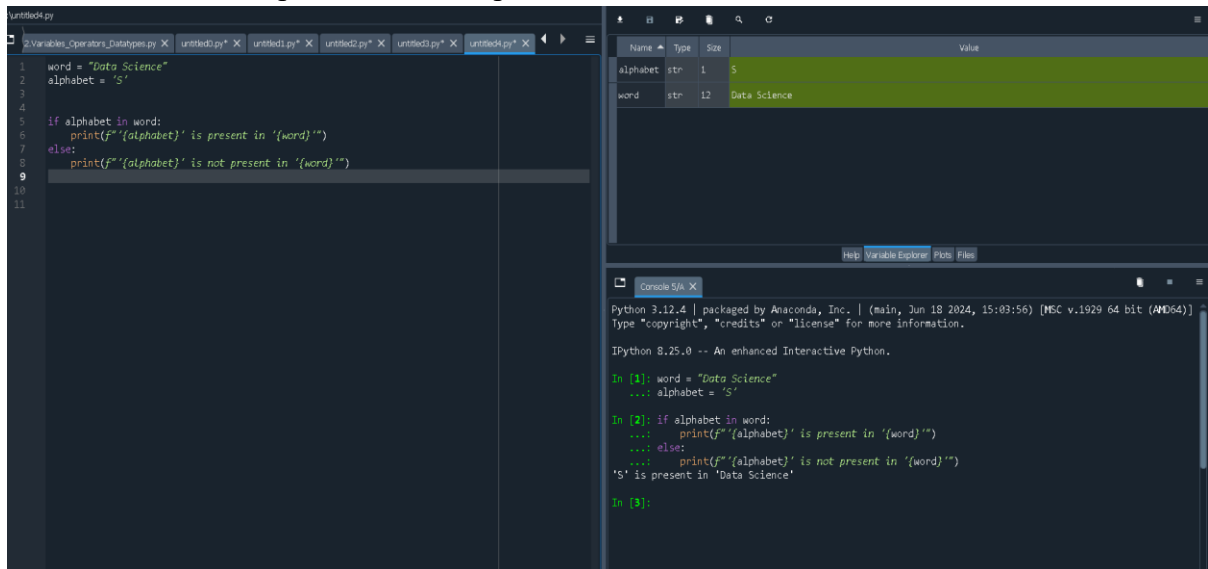
B. c\*=5

```
c = c * 5
```

```
c = 10 * 5
```

c = 50

2. A. How to check the presence of an alphabet 'S' in the word "Data Science" .



```

1 word = "Data Science"
2 alphabet = 'S'
3
4
5 if alphabet in word:
6     print(f'{alphabet} is present in {word}')
7 else:
8     print(f'{alphabet} is not present in {word}')
9
10
11

```

Name	Type	Size	Value
alphabet	str	1	S
word	str	12	Data Science

```

Python 3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 15:03:56) [MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 8.25.0 -- An enhanced Interactive Python.

In [1]: word = "Data Science"
...: alphabet = 'S'

In [2]: if alphabet in word:
...:     print(f'{alphabet} is present in {word}')
...: else:
...:     print(f'{alphabet} is not present in {word}')
'S' is present in 'Data Science'

In [3]:

```

- B. How can you obtain 64 by using numbers 4 and 3 .

```
result = 4 ** 3
```

```
print(result)
```

## Variables

### Please implement by using Python

- What will be the output of the following (can/cannot):
  - Age1=5  
Age1=5  
Valid variable name.  
Starts with a letter(`A`)  
Followed by letters (`g` and `e`) and a digit (`1`).  
Result: valid
  - 5age=55  
Invalid variable name

Starts with a digit(`5`)

Variable names cannot start with a digit according to Python naming rules.

Result: Invalid

2. What will be the output of following (can/cannot):

a. Age\_1=100

Valid variable name.

Starts with a letter(`A`) and includes an underscore(`\_`).

Followed by letters(`g` and `e`) and a digit(`1`).

Result: **Valid**

b. age@1=100

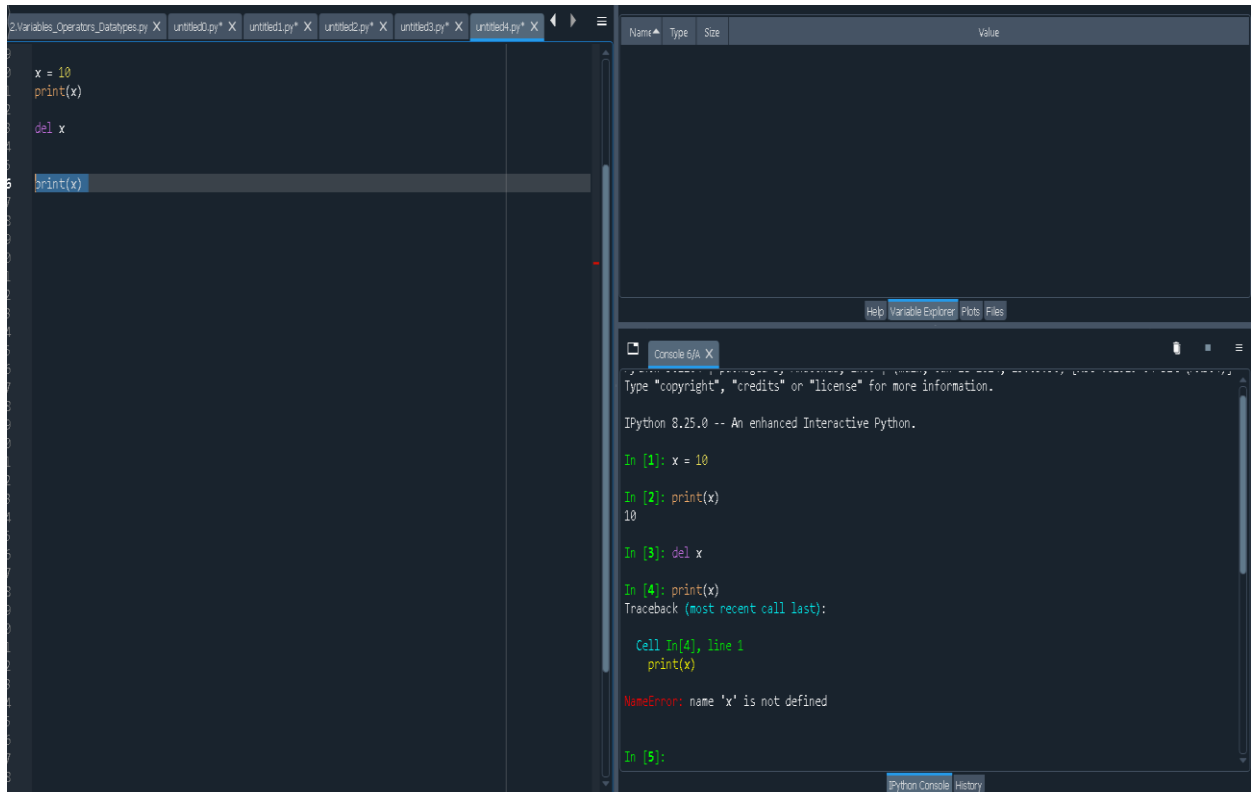
Invalid variable name.

Contains a special character(`@`).

Python variable names cannot contain special characters like `@`.

Result: **Invalid**

### 3. How can you delete variables in Python ?



The screenshot shows a Jupyter Notebook interface with a code editor on the left and an IPython console on the right. The code editor contains the following Python code:

```
x = 10
print(x)
del x
print(x)
```

The IPython console shows the execution history:

```
In [1]: x = 10
In [2]: print(x)
10
In [3]: del x
In [4]: print(x)
Traceback (most recent call last):
  Cell In[4], line 1
    print(x)
NameError: name 'x' is not defined
In [5]:
```

The console output demonstrates that after deleting the variable `x` with `del x`, attempting to print it results in a `NameError: name 'x' is not defined`.