Q1. Create one variable containing following type of data:
(i) string
(ii) list
(iii) float
(iv) tuple

In [2]:

```
x = 'st'
print(x)
```

```
st
```

In [3]:

```
lis = [1,2,3,4,5]
print(lis)
```

```
[1, 2, 3, 4, 5]
```

In [4]:

```
x = 2.5
print(x)
```

```
2.5
```

In [5]:

```
tup = (1,2,3,4)
print(tup)
```

```
(1, 2, 3, 4)
```

Q2. Given are some following variables containing data:
(i) var1 = ‘ ‘
(ii) var2 = ‘[DS , ML , Python]’
(iii) var3 = [‘DS’ , ’ML’ , ‘Python’]
(iv) var4 = 1.

In [6]:

```
var1 =''
print(type(var1))
```

```
<class 'str'>
```

In [7]:

```
var2 = '[Ds,ML,python]'
print(type(var2))
```

```
<class 'str'>
```

In [8]:

```
var3 = ['DS','ML','python']
print(type(var3))
```

```
<class 'list'>
```

In [9]:

```
var4 = 1
print(type(var4))
```

```
<class 'int'>
```

```
Q3. Explain the use of the following operators using an example:
(i) /
(ii) %
(iii) //
(iv) **
```

In [10]:

```
# / is Division operators
x = 4/2
print(x)
```

```
2.0
```

In [12]:

```
# % is Modulo operators to find reminder
x = 5%2
print(x)
```

```
1
```

In [13]:

```
# // is floor division operators
x = 5//2
print(x)
```

```
2
```

In [14]:

```
# ** power value of given a and b
x = 3**2
print(x)
```

```
9
```

Q4. Create a list of length 10 of your choice containing multiple types of data. Using for loop print the
element and its data type.

In [15]:

```
my_list = [1, 3.14, "Hello, world!", [1, 2, 3], {'name': 'John', 'age': 30}, (5, 6), Tru

for element in my_list:
    print("Element:", element)
    print("Data type:", type(element))
    print()
```

```
Element: 1
Data type: <class 'int'>

Element: 3.14
Data type: <class 'float'>

Element: Hello, world!
Data type: <class 'str'>

Element: [1, 2, 3]
Data type: <class 'list'>

Element: {'name': 'John', 'age': 30}
Data type: <class 'dict'>

Element: (5, 6)
Data type: <class 'tuple'>

Element: True
Data type: <class 'bool'>

Element: None
Data type: <class 'NoneType'>

Element: 2023
Data type: <class 'int'>

Element: (1+2j)
Data type: <class 'complex'>
```

Q5. Using a while loop, verify if the number A is purely divisible by number B and if so then how many
times it can be divisible.

In [17]:

```
A = int(input("Enter the number A: "))
B = int(input("Enter the number B: "))
count = 0
if B != 0:
    while A >= B:
        A = A - B
        count += 1
    if A == 0:
        print(f"{A} is purely divisible by {B} and can be divided {count} times.")
    else:
        print(f"{A} is not purely divisible by {B}.")
else:
    print("B cannot be zero for division.")
```

```
Enter the number A: 50
Enter the number B: 10
0 is purely divisible by 10 and can be divided 5 times.
```

```
Q6. Create a list containing 25 int type data. Using for loop and if-else condition
print if the element is
divisible by 3 or not.
```

In [18]:

```
my_list = [12, 7, 9, 27, 15, 36, 41, 20, 63, 18, 5, 10, 30, 42, 55, 8, 21, 16, 33, 6, 25
for element in my_list:
    if element % 3 == 0:
        print(f"{element} is divisible by 3.")
    else:
        print(f"{element} is not divisible by 3.")
```

```
12 is divisible by 3.
7 is not divisible by 3.
9 is divisible by 3.
27 is divisible by 3.
15 is divisible by 3.
36 is divisible by 3.
41 is not divisible by 3.
20 is not divisible by 3.
63 is divisible by 3.
18 is divisible by 3.
5 is not divisible by 3.
10 is not divisible by 3.
30 is divisible by 3.
42 is divisible by 3.
55 is not divisible by 3.
8 is not divisible by 3.
21 is divisible by 3.
16 is not divisible by 3.
33 is divisible by 3.
6 is divisible by 3.
25 is not divisible by 3.
48 is divisible by 3.
99 is divisible by 3.
22 is not divisible by 3.
45 is divisible by 3.
```

Q7. What do you understand about mutable and immutable data types? Give examples for both showing
this property.

immutable data type =
An immutable data type is one that cannot be changed after it is created. This means that any operation on an immutable object will always create a new object.

In [21]:

```
# example of immutable data type
original_tuple = (1, 2, 3)
original_tuple.append(4)
print("Original tuple:", original_tuple)
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
Cell In[21], line 3
      1 # example of immutable data type
      2 original_tuple = (1, 2, 3)
----> 3 original_tuple.append(4)
      4 print("Original tuple:", original_tuple)

AttributeError: 'tuple' object has no attribute 'append'
```

A mutable data type, on the other hand, can be modified after it is created. This means that you can change the contents of the object without creating a new object.

In [20]:

```
# example of mutable data type
original_list = [1, 2, 3]
original_list.append(4)
print("Original list:", original_list)
```

```
Original list: [1, 2, 3, 4]
```

In [ ]: