Python Pretest

1. Python is a Interpreted, Dynamic and Objectect Oriented Programming Language Interpreter converts the high level language code into low level language code line by line. Dynamic Typed language is a language in which user need not to mention the data type of variables

2. Identifier is an name of any function, variable or method while variable is an entity which is used to store some kind of data

3. Indexing is used to access the data according to their position from advanced data types like list, tuple. Slicing is used to access the specific range of elements

4. list and tuple are the advanced data types containing multiple values of different data types lists are mutable and tuples are immutable immutables cannot change their values once defined mutables can be changed

5. operators are used to perform any kind of operations on the operands their are relational, arithmetic, assignment, logical, comparison, bitwise,membership and identity operators.

6. mean is the average value calculated by performing sum of values divide by number of values median is the middle value after performing sorting mode is the frequently occuring value in the given data

7. key features of python object oriented, interpreted, dynamic, easy to use

8. data types int, float, char, string, boolean, list, tuple, set and dictionary

9. local variables are accessible within the function or within the local object global variables are accessible throughout the code

10. '#' is used to write an comment is python for single line '/* */' is used for multiline comment

11. '/* */' is used for multiline comment

12. literals are a notation for representing a fixed value in source code.

13. we can assign values to variables using assignment operator as a=10, b=15

14. list is advanced data type in python which is used to store multiple data values

In [19]:
```
list1=[1,2,3,4,5]
```

In [16]:
```
print(list1[2])
```

```
3
```

In [17]:
```
list1[3]=6
```

In [20]:
```
print(list1[3])
```

```
4
```

Day 1

Type Casting

In [1]:
```
a=10
```

In [2]:
```
b=bool(a)
```

In [3]:
```
print(b)
```

```
True
```

In [4]:
```
c=0
```

In [5]:
```
print(bool(c))
```

```
False
```

In [16]:
```
d=10.5+0j
```

In [17]:
```
print(float(d))
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[17], line 1
----> 1 print(float(d))

TypeError: float() argument must be a string or a real number, not 'comple
x'
```

In [21]:
```
a=45.75
```

In [23]:
```
print(int(a))
```

```
45
```

In [25]:
```
type(a)
```

Out[25]:
```
float
```

In [26]:
```
a="Hello"
```

In [31]:
```
print(int(a))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[31], line 1
----> 1 print(int(a))

ValueError: invalid literal for int() with base 10: 'Hello'
```

ASCII Value

In [40]:
```
a='A'
print(ord(a))
```

```
65
```

In [41]:
```
print(type(a))
```

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

In [42]:
```
import numpy as np
```

In [50]:
```
array=np.array([1,2,3,4,1.5,1.7,"string"]) #array is homogenous datatype
# depends on the output data type
```

In [51]:
```
array
```

Out[51]:
```
array(['1', '2', '3', '4', '1.5', '1.7', 'string'], dtype='<U32')
```

1 List

In [52]:
```
list1=[1,2,3,4,"hii",4.5] #List is heterogenous
```

In [53]:
```
print(list1)
```

```
[1, 2, 3, 4, 'hii', 4.5]
```

In [54]:
```
#indexing
print(list1[3])
```

```
4
```

In [55]:
```
print(list1[-2])
```

```
hii
```

In [56]:
```
# Slicing used for accessing multiple elements
```

In [57]:
```
l=[1,2,3,4,5,6,7,8,9]
```

In [58]:
```
print(l[0:5])
```

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

In [62]:
```
list1[0:]
```

Out[62]:
```
[1, 2, 3, 4, 'hii', 4.5]
```

In [68]:
```
list1[-5:]
```

Out[68]:
```
[2, 3, 4, 'hii', 4.5]
```

In [69]:
```
l[-6:-5]
```

Out[69]:
```
[4]
```

In [70]:
```
l[::2]
```

Out[70]:
```
[1, 3, 5, 7, 9]
```

In [71]:
```
l[::1]
```

Out[71]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [72]:
```
l[::-1]
#reverse list using slicing without for loop or function
```

Out[72]:
```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [73]:
```
l[-1::]
```

Out[73]:
```
[9]
```

In [74]:
```
l[-9::1]
```

Out[74]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [75]:
```
l[-1::-1]
```

Out[75]:
```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```