Data Structure Practice

- The code prompts the user to enter six integer values, one by one.
- These values are then stored in a list called list1, which is printed at the end.

In [1]:
```
a = int(input("Enter a value:"))
b = int(input("Enter a value:"))
c = int(input("Enter a value:"))
d = int(input("Enter a value:"))
e = int(input("Enter a value:"))
f = int(input("Enter a value:"))

list1 = [a, b, c, d, e, f]

print(list1)
```

```
Enter a value:22
Enter a value:23
Enter a value:25
Enter a value:26
Enter a value:76
Enter a value:50
[22, 23, 25, 26, 76, 50]
```

In [2]:
```
print(list1.index(d))
# The code prints the index (position) of the value d in the list list1.
```

```
3
```

In [3]:
```
print(list1[4])
# The code prints the value at index 4 in the list list1.
```

```
76
```

In [4]:
```
print(list1[1:4])
```

```
[23, 25, 26]
```

The code prints the sublist of list1 containing elements from index 1 to 3 (the slice does not include the element at index 4).

In [5]:
```
print(list1.reverse)
# <built-in method reverse of list object at 0x109055440>
```

```
<built-in method reverse of list object at 0x1132eb700>
```

In [ ]:
```
#you are not passing any argument here that's why it's different in nature to the print(
```

The code `print(list1.reverse)` would not work as intended because reverse is a `method`, not a `property`. To `reverse` the list and `print` it, you should use the `reverse` method followed by a `print statement`.

In [7]:
```
list1.reverse()
print(list1)
```

```
[50, 76, 26, 25, 23, 22]
```

In [9]:
```
print(len(list1))
# This would print the length of the list
```

```
6
```

In [ ]:
```
#you are passing an argument here
```

In [10]:
```
list1.append(99)
print (list1)
# The code adds the value 99 to the end of the list list1 and then prints the updated li
```

```
[50, 76, 26, 25, 23, 22, 99]
```

Difference between append and insert:

- `append(value)` adds value to the end of the list.
- `insert(index, value)` adds value at the specified index, shifting subsequent elements to the right.

Best use situation:

- Use `append` when you need to add an element to the end of the list.
- Use `insert` when you need to add an element at a specific position in the list.

In [11]:
```
list1.insert(1, 'Hello')
print (list1)
```

```
[50, 'Hello', 76, 26, 25, 23, 22, 99]
```

In [12]:
```
list1.extend([10,15,18])
print (list1)
# Adds elements to the end of the list
```

```
[50, 'Hello', 76, 26, 25, 23, 22, 99, 10, 15, 18]
```

In [13]:
```
a = int(input("Enter a value:"))
b = int(input("Enter a value:"))
c = int(input("Enter a value:"))
d = int(input("Enter a value:"))
e = int(input("Enter a value:"))
f = int(input("Enter a value:"))

list2 = [a, b, c, d, e, f]

print(list2)
```

```
Enter a value:100
Enter a value:354
Enter a value:786
Enter a value:435
Enter a value:901
Enter a value:223
[100, 354, 786, 435, 901, 223]
```

In [14]:
```
list1+list2
#Concatenation method
```

Out[14]:
```
[50, 'Hello', 76, 26, 25, 23, 22, 99, 10, 15, 18, 100, 354, 786, 435, 901, 223]
```

In [18]:
```
list2.remove(100)
print (list2)
# to remove - Error
```

```
[354, 786, 435, 901, 223]
```

In [21]:
```
list1.remove('Hello')
```

```
print(list1)
```

```
[50, 76, 26, 25, 23, 22, 99, 10, 15, 18]
```

In [22]:
```
list2.pop(1)
print (list2)
```

```
[354, 435, 901, 223]
```

- The code removes the element at `index 1` from `list2` using the `pop( )` method, and then prints the updated list.

In [23]:
```
list4 = [1, 6, 8, 7, 2, 9, 4, 3, 5]
print (list4)
```

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

In [24]:
```
list4.sort()
print (list4)
# Sorts the list in order
```

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

In [25]:
```
list4.sort(reverse=True)
print (list4)
```

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

The code sorts the list `list4` in `descending order` (from highest to lowest) and stores the sorted result back into list4

In [26]:
```
list4.sort(reverse=False)
print (list4)
#Sorts the list in ascending order.
```

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

In [27]:
```
list5 = ['3.5m', '2', 'axb']
print (list5)
```

```
['3.5m', '2', 'axb']
```

In [28]:
```
list5.sort()
print (list5)
```

```
['2', '3.5m', 'axb']
```

In [29]:
```
object16 = [2, 4, 6, 8, 10]
square_object16 = []

for x in object16:
    square_object16.append(x**2)
square_object16
```

Out[29]:
```
[4, 16, 36, 64, 100]
```

The code creates a new list `square_object16` that contains the squares of each element from the list `object16`.

- `object16 = [2, 4, 6, 8, 10]`: Defines a list of integers.
- `square_object16 = []`: Initializes an empty list to store the squared values.
- `for x in object16:`: Iterates through each element `x` in `object16`.

- `square_object16.append(x**2)` : Computes the square of `x` and appends it to `square_object16` .
- `square_object16` : Outputs the list `square_object16` containing `[4, 16, 36, 64, 100]` , which are the squares of `[2, 4, 6, 8, 10]` respectively.

In [46]:
```
object17 = [12, 14, 16, 18, 20]
square_object17 = [x**2 for x in object17]
square_object17
```

Out[46]:
```
[144, 196, 256, 324, 400]
```

The code creates a list `square_object17` that contains the squares of each element from the list `object17` , using a list comprehension.

Here's a shorter explanation:

- `object17 = [12, 14, 16, 18, 20]` : Defines a list of integers.
- `square_object17 = [x**2 for x in object17]` : Creates a new list `square_object17` where each element is the square of the corresponding element from `object17` .
- `square_object17` : Outputs the list `square_object17` , which contains `[144, 196, 256, 324, 400]` , the squares of `[12, 14, 16, 18, 20]` respectively.