Lab Work — List and Tuple

Objective

This lab focuses on mastering Python lists and tuples for data storage and manipulation. You will learn to create, access, update, slice, and modify lists, work with nested and heterogeneous data, apply mutator methods, use operators, comprehensions, and convert between tuples and lists. For tuples, the lab demonstrates immutability, indexing, slicing, packing/unpacking, and built-in functions. By applying these concepts to student management, inventory, and configuration scenarios, you gain practical skills in organizing and processing structured data efficiently.

Task 1 — Inventory System

Create list: ["pencil", "notebook", "eraser", "marker"].

Add 2 new products at the end.

Insert a product at position 2.

Explain: list is appropriate due to mutability.

In [1]:
```
product_names =["pencil", "notebook", "eraser", "marker"]
product_names.extend(["pen","paints"])
product_names.insert( 1,"highlighters")
print(product_names)
```

```
['pencil', 'highlighters', 'notebook', 'eraser', 'marker', 'pen', 'paints']
```

Explanation: A list is used because it is ordered, changeable, and supports easy insertion and extension. Perfect for managing an inventory that changes over time.

Task 2 — Data Slicing & Indexing

Access first, last, and second-to-last items from temps = [23,25,19,30,21,22].

Slice to get middle 4 values.

Use negative indexing to get item “21”

In [2]:
```
temps = [23, 25, 19, 30, 21, 22]
first= temps[0]
last= temps[5]
second_last= temps[4]
mid_four= temps[1:5]
val_21= temps[-2]
print(first,last,second_last)
print(mid_four)
```

```
print(val_21)
```

```
23 22 21
[25, 19, 30, 21]
21
```

Task 3 — Nested Lists & Heterogeneous Data

Create 3 students as [name, roll_number, grades_list].

Change one student’s second grade.

Add a fourth grade to one student’s grades_list.

In [3]:
```
student_record=[
                ["Adeel Amjid",456,[21,24,22]],
                ["Sarah Neelum",457,[20,21,22]],
                ["Gulab khan",458,[15,17,20]]
                ]
student_record[1][2][2]=25
student_record[2][2].append(22)
print(student_record)
```

```
[['Adeel Amjid', 456, [21, 24, 22]], ['Sarah Neelum', 457, [20, 21, 25]], ['Gul
ab khan', 458, [15, 17, 20, 22]]]
```

Task 4 — List Mutator Methods

Start with colours = ["red","green","blue"].

Append “yellow”.

Extend with ["purple","orange"].

Insert “black” at index 1.

Remove “green”.

Pop last item and print it.

Print final list.

In [4]:
```
colors = ["red", "green", "blue"]
colors.append('yellow')
colors.extend(["purple", "orange"])
colors.insert(1,'black')
colors.remove('green')
removed_color=colors.pop()
print(removed_color)
print(colors)
```

```
orange
['red', 'black', 'blue', 'yellow', 'purple']
```

Task 5 — Operators & Built-in Functions on Lists

numbers = [5,10,15,20].

Check if 15 is in the list.

Concatenate [25,30] using +.

Repeat list twice using *2.

Apply len(), min(), max(), sum().

In [5]:
```
numbers = [5, 10, 15, 20]
print(15 in numbers)
new_list= numbers+[23,30]
print(new_list)
repeating_list=numbers*2
print(repeating_list)
print(len(numbers))
print(min(numbers))
print(max(numbers))
print(sum(numbers))
```

```
True
[5, 10, 15, 20, 23, 30]
[5, 10, 15, 20, 5, 10, 15, 20]
4
5
20
50
```

Task 6 — List Comprehensions & Conversion

Convert tuple values = (2,4,6,8,10) to list.

Use list comprehension to square each value.

Filter squared values > 30.

Print resulting list.

In [6]:
```
values = (2, 4, 6, 8, 10)
A=list(values)
squaring=[x**2 for x in A ]
filtered=[num for num in squaring if num>30]
print(filtered)
```

```
[36, 64, 100]
```

Task 1 — Fixed Configuration

device = ("device01","192.168.1.10",8080,True).

Access each element by index.

Attempt to change port (8080 → 9090); observe error.

Convert tuple → list → modify port → convert back to tuple.

In [7]:
```
device = ("device01", "192.168.1.10", 8080, True)
print(device[0])
print(device[1])
print(device[2])
print(device[3])
try:
    device[2]=9090
except TypeError as error:
    print(error)
device_list= list(device)
device_list[2]=9090
device= tuple(device_list)
print(device)
```

```
device01
192.168.1.10
8080
True
'tuple' object does not support item assignment
('device01', '192.168.1.10', 9090, True)
```

Task 2 — Creating Tuples & Single Element Nuance

Create single = (5,).

Create not_tuple = (5); show type.

Create t = tuple(range(5)).

Print t.

In [8]:
```
single=(5,)
print(single)
print(type(single))

not_tuple=(5)
print(not_tuple)
print(type(not_tuple))

t=tuple(range(5))
print(t)
```

```
(5,)
<class 'tuple'>
5
<class 'int'>
(0, 1, 2, 3, 4)
```

Task 3 — Indexing, Slicing, Negative Indices

weekdays = ("Mon","Tue","Wed","Thu","Fri","Sat","Sun").

Access first and last day.

Slice to get "Wed","Thu","Fri".

Negative indexing to get “Sat”.

In [9]:
```
weekdays = ("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
print(weekdays[0])
print(weekdays[-1])
print(weekdays[2:5])
print(weekdays[-2])
```

```
Mon
Sun
('Wed', 'Thu', 'Fri')
Sat
```

Task 4 — Nested Tuples & Mixing

data = ("UserA",(2023,9,15),["task1","task2"]).

Access date tuple: print year, month, day.

Access task list: add a task.

Explain: mutable item in tuple can be modified.

In [10]:
```
data = ("UserA", (2023, 9, 15), ["task1", "task2"])
date=data[1]
print(data[0])
print(date[1])
print(date[2])
tasks=(data[2])
tasks.append("task3")
print(data)
```

```
UserA
9
15
('UserA', (2023, 9, 15), ['task1', 'task2', 'task3'])
```

Task 5 — Packing & Unpacking

point = (10,20,30); unpack into x,y,z.

Swap x and z in one line.

Pack x,y,z into point2.

Print both point and point2.

In [11]:
```
point = (10, 20, 30)
x,y,z=point
print(x)
print(y)
print(z)
x,z=z,x
print(x)
print(z)
point2=(x,y,z)
print(point)
print(point2)
```

```
10
20
30
30
10
(10, 20, 30)
(30, 20, 10)
```

Task 6 — Choosing Tuple vs List

Define months as tuple.

Explain: tuple better for fixed data.

Try append → shows error.

Convert to list if modification needed.

In [12]:
```
months = ("Jan", "Feb", "Mar", "Apr", "May", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
print(months)

#EXPLANATION:-
  #The months of year are fixed they dont change so its better to store them i
  #tuples are immutable so can prevent accidental changes. moreover they use l

try:
    months.append('femto')
except AttributeError as e:
    print(e)

months_list=list(months)
```

```
months_list.append("femto")
print(months_list)
```

```
('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec')
'tuple' object has no attribute 'append'
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec', 'femto']
```

Task 7 — Operators & Built-in Functions with Tuples

nums = (3,6,9,12).

Test membership using in and not in.

Concatenate nums + (15,18).

Multiply tuple by 3.

Use len(), min(), max(), sum().

In [13]:
```
nums = (3, 6, 9, 12)
print(6 in nums)
print(5 in nums)
print(5 not in nums)
new_nums= nums+ (15,18)
print(new_nums)
multiplying= nums*3
print(multiplying)
print(len(nums))
print(min(nums))
print(max(nums))
print(sum(nums))
```

```
True
False
True
(3, 6, 9, 12, 15, 18)
(3, 6, 9, 12, 3, 6, 9, 12, 3, 6, 9, 12)
4
3
12
30
```