Program 1: NumPy Transactions

Analyzes transaction volumes for multiple branches using NumPy arrays.

```
import numpy as np

# 1. Create a 2D array representing transaction volumes for 4 branches over 6 months
transactions = np.array([
    [120, 135, 150, 160, 145, 155],  # Branch 1
    [200, 195, 210, 220, 215, 225],  # Branch 2
    [90,  100, 95,  110, 105, 115],  # Branch 3
    [160, 170, 165, 175, 180, 190]   # Branch 4
])

print("Transaction volumes (branches x months):")
print(transactions)

# 2. Calculate the total transactions per branch
branch_totals = transactions.sum(axis=1)
print("\nTotal transactions per branch:", branch_totals)

# 3. Identify the branch with the highest total transactions
max_branch_index = np.argmax(branch_totals)  # returns index (0-based)
print("Branch with highest total transactions:", max_branch_index + 1)

# 4. Compute the average monthly transaction volume across all branches
avg_monthly = transactions.mean(axis=0)
print("\nAverage monthly transaction volume across all branches:", avg_monthly)

# 5. Reshape the array to a 3x8 format
reshaped = transactions.reshape(3, 8)
print("\nReshaped array (3x8):")
print(reshaped)
```

Program 2: Delivery Routes (Lists)

Demonstrates operations on a list of delivery routes.

```
# 1. Create a list of 10 delivery routes (strings)
routes = [
    "North Line", "South Express", "East Cargo", "West Freight", "Central Hub",
    "Northern Star", "New Town", "Night Express", "River Route", "Green Valley"
]

print("Initial routes:")
print(routes)

# 2. Append a new route and remove a discontinued one
routes.append("Harbor Line")   # adding new route
routes.remove("Night Express") # remove discontinued /stopped route

print("\nAfter adding and removing routes:")
print(routes)

# 3. Sort the list alphabetically and reverse it
routes.sort()
```

```
print("\nSorted routes (A–Z):")
print(routes)

routes.reverse()
print("\nRoutes in reverse order (Z–A):")
print(routes)

# 4. Count how many routes start with the letter 'N'
count_N = sum(route.startswith("N") for route in routes)
print("\nNumber of routes starting with 'N':", count_N)

# 5. Use list comprehension to create a new list of routes longer than 10 characters
long_routes = [route for route in routes if len(route) > 10]
print("\nRoutes longer than 10 characters:")
print(long_routes)
```

Program 3: Patient Records (Tuples)

Uses tuples to store patient records and demonstrates immutability.

```
# 1. Create a tuple for a patient (Name, Age, Blood Pressure, Heart Rate)
patient = ("Opige", 21, "120/80", 72)

print("Patient record:", patient)

# 2. Access and print the patient’s age and heart rate
print("\nAge:", patient[1])
print("Heart Rate:", patient[3])

# 3. Explain why tuples are suitable
# Tuples are immutable, meaning they cannot be changed once created.
# This is good for storing patient vitals (static records) to ensure data integrity.

# 4. Convert the tuple to a list, update the heart rate, and convert it back
patient_list = list(patient)
patient_list[3] = 76   # update heart rate
patient_updated = tuple(patient_list)

print("\nUpdated patient record:", patient_updated)

# 5. Create a tuple of 5 patients and extract all names
patients = (
    ("Opige", 45, "120/80", 72),
    ("Glen", 52, "130/85", 80),
    ("Kali", 39, "118/75", 70),
    ("Tegemeo", 60, "140/90", 85),
    ("Good Job", 29, "110/70", 68)
)

names = [p[0] for p in patients]
print("\nAll patient names:", names)
```

Program 4: Inventory Management (Dictionaries)

Manages product inventory using dictionaries and functions.

```
# 1. Create a dictionary with 5 products and stock quantities
inventory = {
    "Laptop": 15,
    "Headphones": 8,
    "Smartphone": 25,
    "Keyboard": 5,
    "Monitor": 12
}

print("Initial Inventory:")
print(inventory)

# 2. Add a new product and update the quantity of an existing one
inventory["Mouse"] = 18          # added new product
inventory["Laptop"] = 20         # updated Laptop quantity

print("\nAfter adding Mouse and updating Laptop:")
print(inventory)

# 3. Write a function to return products with stock less than 10
def low_stock(inv):
    return {product: qty for product, qty in inv.items() if qty < 10}

print("\nProducts with stock less than 10:")
print(low_stock(inventory))

# 4. Delete a discontinued product and display the updated dictionary
del inventory["Keyboard"]

print("\nAfter removing Keyboard:")
print(inventory)

# 5. Use .items() to loop through and print each product with its quantity
print("\nFinal Inventory List:")
for product, qty in inventory.items():
    print(f"{product}: {qty}")
```