Python for Cheminformatics & Bioinformatics: Lab Solutions

AI-Driven Drug Development Training

February 2026

Contents

1 Lab 1: Variables & Data Types **2**
1.1 Scenario 1: Personal Info . . . 2
1.2 Scenario 2: Financial Data . . . 2

2 Lab 2: Operators **2**
2.1 Scenario 1: Student Grades . . . 2
2.2 Scenario 2: Shopping Cart . . . 3

3 Lab 3: Strings **3**
3.1 Scenario 1: Text Processing . . . 3
3.2 Scenario 2: File Names . . . 3

4 Lab 4: Conditionals **4**
4.1 Scenario 1: Bank Balance . . . 4
4.2 Scenario 2: Ticket Pricing . . . 4
4.3 Scenario 3: Login System . . . 5

5 Lab 5: Loops **5**
5.1 Scenario 1: Multiples of 3 . . . 5
5.2 Scenario 2: Nested Loop Matrix . . . 5

6 Lab 6: Functions **6**
6.1 Scenario 1: Calculator . . . 6
6.2 Scenario 2: Temperature Conversion . . . 6

7 Lab 6B: Error Handling **7**
7.1 Scenario 1: Safe Calculator . . . 7
7.2 Scenario 2: File Reader . . . 7

8 Lab 7: Lists **8**
8.1 Scenario 1: Grocery List . . . 8
8.2 Scenario 2: Numbers . . . 8

9 Lab 7B: Tuples & Sets **8**
9.1 Scenario 1: Coordinate System . . . 8
9.2 Scenario 2: Unique Items . . . 9

10 Lab 8: Advanced Lists (Comprehensions & Map/Filter) **9**
10.1 Scenario 1: Student Scores . . . 9
10.2 Scenario 2: Product Prices . . . 10

11 Lab 8B: Lambda & Scope **10**
11.1 Scenario 1: Sorting 10
11.2 Scenario 2: Counter 10

12 Lab 9: Dictionaries **11**
12.1 Scenario 1: Employee Records 11
12.2 Scenario 2: Grades 11

13 Lab 10: File Handling **12**
13.1 Scenario 1: Log File 12
13.2 Scenario 2: Student Scores 12

14 Lab 11: NumPy Arrays **13**
14.1 Scenario 1: Matrix Operations 13
14.2 Scenario 2: Sensor Data 13

15 Lab 11B: Pandas Data Structures **14**
15.1 Scenario 1: Employee Database 14
15.2 Scenario 2: Sales Analysis 14

16 Lab 12: JSON & Regex **14**
16.1 Scenario 1: Student JSON 14
16.2 Scenario 2: Email Validation 15

17 Lab 13: Modules & Packages **16**
17.1 Scenario 1: Utility Module (mymath.py) 16
17.2 Scenario 2: Standard Library 16

1 Lab 1: Variables & Data Types

1.1 Scenario 1: Personal Info

```python
# Create variables
name = "John Doe"
age = 25
city = "New York"
is_student = True

# Print info with f-string
print(f"Name: {name}, Age: {age}, City: {city}, Student: {is_student}")

# Convert age to string and concatenate
age_str = str(age)
message = name + " is " + age_str + " years old"
print(message)

# Bonus: Check adult or minor
if age > 18:
    status = "Adult"
else:
    status = "Minor"
print(f"{name} is an {status}")
```

1.2 Scenario 2: Financial Data

```python
# Variables
price = 49.99
quantity = 3
balance = 200.00

# Compute total cost and remaining balance
total_cost = price * quantity
remaining_balance = balance - total_cost

print(f"Total Cost: ${total_cost:.2f}")
print(f"Remaining Balance: ${remaining_balance:.2f}")

# Bonus: Check if total cost exceeds balance
if total_cost > balance:
    print("Insufficient funds!")
else:
    print("Purchase successful!")
```

2 Lab 2: Operators

2.1 Scenario 1: Student Grades

```python
# Scores
math = 78
science = 85
english = 72

# Compute total and average
```

```
total = math + science + english
average = total / 3

print(f"Total: {total}, Average: {average:.2f}")

# Check pass/fail
if average > 50:
    print("Result: Pass")
else:
    print("Result: Fail")
```

2.2 Scenario 2: Shopping Cart

```
# Variables
price = 55.00
quantity = 6
balance = 500.00

# Total cost and remainder
total_cost = price * quantity
remainder = balance - total_cost

print(f"Total: ${total_cost}, Remainder: ${remainder}")

# Bonus: Logical check
if price > 50 or quantity > 5:
    print("Eligible for bulk discount!")
```

3 Lab 3: Strings

3.1 Scenario 1: Text Processing

```
text = "Machine Learning"

# Print first & last word
words = text.split()
print(f"First word: {words[0]}")
print(f"Last word: {words[-1]}")

# Count vowels
vowels = "aeiouAEIOU"
vowel_count = sum(1 for char in text if char in vowels)
print(f"Vowel count: {vowel_count}")

# Replace "Learning" with "AI"
new_text = text.replace("Learning", "AI")
print(f"Modified: {new_text}")

# Bonus: Reverse string
reversed_text = text[::-1]
print(f"Reversed: {reversed_text}")
```

3.2 Scenario 2: File Names

```
path = "/home/user/docs/file.txt"

# Extract parts using split
parts = path.rsplit("/", 1)
directory = parts[0]
filename_with_ext = parts[1]

# Split filename and extension
filename, extension = filename_with_ext.rsplit(".", 1)

print(f"Directory: {directory}")
print(f"Filename: {filename}")
print(f"Extension: {extension}")

# Convert filename to uppercase
print(f"Uppercase filename: {filename.upper()}")
```

4 Lab 4: Conditionals

4.1 Scenario 1: Bank Balance

```
balance = -50

if balance > 0:
    print("Balance: Positive")
elif balance == 0:
    print("Balance: Zero")
else:
    print("Balance: Overdraft")

# Bonus: match-case ranges
balance = 5000
match balance:
    case b if b < 1000:
        category = "Low"
    case b if b < 10000:
        category = "Medium"
    case _:
        category = "High"
print(f"Balance category: {category}")
```

4.2 Scenario 2: Ticket Pricing

```
age = 65
is_vip = True

if age < 12:
    ticket_type = "Child"
    price = 5.00
elif age < 60:
    ticket_type = "Adult"
    price = 15.00
else:
    ticket_type = "Senior"
    price = 10.00

# Bonus: VIP discount
if is_vip:
    price *= 0.8  # 20% discount

print(f"Ticket: {ticket_type}, Price: ${price:.2f}")
```

4.3 Scenario 3: Login System

```
stored_username = "admin"
stored_password = "secret123"
is_active = True
max_attempts = 3

for attempt in range(max_attempts):
    username = input("Username: ")
    password = input("Password: ")

    if username == stored_username and password == stored_password:
        if is_active:
            print("Login successful!")
            break
        else:
            print("Account is inactive!")
            break
    else:
        remaining = max_attempts - attempt - 1
        print(f"Invalid credentials. {remaining} attempts remaining.")
else:
    print("Account locked due to too many failed attempts.")
```

5 Lab 5: Loops

5.1 Scenario 1: Multiples of 3

```
# Print multiples of 3 from 1-20
multiples = []
for i in range(1, 21):
    if i % 3 == 0:
        print(i)
        multiples.append(i)

print(f"Multiples list: {multiples}")

# Bonus: List comprehension
multiples_comp = [x for x in range(1, 21) if x % 3 == 0]
print(f"Using comprehension: {multiples_comp}")
```

5.2 Scenario 2: Nested Loop Matrix

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

# Print all elements
for row in matrix:
    for element in row:
        print(element, end=" ")
    print()

# Create flat list using nested comprehension
flat_list = [elem for row in matrix for elem in row]
print(f"Flat list: {flat_list}")

# Bonus: Filter even numbers
even_numbers = [elem for row in matrix for elem in row if elem % 2 == 0]
print(f"Even numbers: {even_numbers}")
```

6 Lab 6: Functions

6.1 Scenario 1: Calculator

```
# Product of two numbers
def multiply(a, b):
    return a * b

# Factorial
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)

# Bonus: Sum list
def sum_list(lst):
    return sum(lst)

# Test
print(f"5 * 3 = {multiply(5, 3)}")
print(f"5! = {factorial(5)}")
print(f"Sum of [1,2,3,4,5] = {sum_list([1, 2, 3, 4, 5])}")
```

6.2 Scenario 2: Temperature Conversion

```
def celsius_to_fahrenheit(c):
    return (c * 9/5) + 32

def fahrenheit_to_celsius(f):
    return (f - 32) * 5/9

# Bonus: Average conversion for list of temps
def average_conversion(temps, to_fahrenheit=True):
    if to_fahrenheit:
        converted = [celsius_to_fahrenheit(t) for t in temps]
    else:
        converted = [fahrenheit_to_celsius(t) for t in temps]
    return sum(converted) / len(converted)
```

```python
# Test
print(f"25C = {celsius_to_fahrenheit(25):.1f}F")
print(f"77F = {fahrenheit_to_celsius(77):.1f}C")
temps_c = [0, 25, 100]
print(f"Average of {temps_c}C in F: {average_conversion(temps_c):.1f}F")
```

7 Lab 6B: Error Handling

7.1 Scenario 1: Safe Calculator

```python
def safe_divide(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
        return None
    except TypeError:
        print("Error: Invalid input types!")
        return None

# Bonus: Log errors to file
def safe_divide_with_log(a, b):
    try:
        result = a / b
        return result
    except (ZeroDivisionError, TypeError) as e:
        with open("error_log.txt", "a") as f:
            f.write(f"Error: {e}, Inputs: a={a}, b={b}\n")
        return None

# Test
print(safe_divide(10, 2))
print(safe_divide(10, 0))
print(safe_divide("10", 2))
```

7.2 Scenario 2: File Reader

```python
def read_file_safe(filename, max_retries=3):
    for attempt in range(max_retries):
        try:
            with open(filename, "r") as f:
                content = f.read()
            return content
        except FileNotFoundError:
            print(f"Attempt {attempt + 1}: File not found!")
            if attempt < max_retries - 1:
                filename = input("Enter correct filename: ")
        finally:
            print(f"Attempt {attempt + 1} complete.")
    return None

# Test
content = read_file_safe("data.txt")
```

```
if content:
    print(content)
```

8 Lab 7: Lists

8.1 Scenario 1: Grocery List

```
# Add 3 items
grocery = []
grocery.append("milk")
grocery.append("bread")
grocery.append("eggs")

# Remove 1 item
grocery.remove("bread")

# Print first & last
print(f"First: {grocery[0]}, Last: {grocery[-1]}")

# Bonus: Sort and reverse
grocery.append("apples")
grocery.append("cheese")
grocery.sort()
print(f"Sorted: {grocery}")
grocery.reverse()
print(f"Reversed: {grocery}")
```

8.2 Scenario 2: Numbers

```
# List of 1-10
numbers = list(range(1, 11))

# Filter even numbers using loop
even_loop = []
for num in numbers:
    if num % 2 == 0:
        even_loop.append(num)
print(f"Even (loop): {even_loop}")

# Bonus: List comprehension
even_comp = [x for x in numbers if x % 2 == 0]
print(f"Even (comprehension): {even_comp}")
```

9 Lab 7B: Tuples & Sets

9.1 Scenario 1: Coordinate System

```
import math

# Create tuple for 2D point
point = (3, 4)

# Unpack and calculate distance from origin
```

```python
x, y = point
distance = math.sqrt(x**2 + y**2)
print(f"Point: {point}, Distance from origin: {distance}")

# Bonus: List of points, find furthest
points = [(1, 1), (3, 4), (5, 12), (2, 3)]
max_distance = 0
furthest_point = None

for p in points:
    x, y = p
    d = math.sqrt(x**2 + y**2)
    if d > max_distance:
        max_distance = d
        furthest_point = p

print(f"Furthest point: {furthest_point}, Distance: {max_distance}")
```

9.2 Scenario 2: Unique Items

```python
# Given list with duplicates
list1 = [1, 2, 2, 3, 4, 4, 5]
list2 = [4, 5, 5, 6, 7, 7, 8]

# Get unique items using set
unique1 = set(list1)
unique2 = set(list2)
print(f"Unique from list1: {unique1}")
print(f"Unique from list2: {unique2}")

# Find common items
common = unique1 & unique2
print(f"Common items: {common}")

# Bonus: Remove items that appear in both (symmetric difference)
only_in_one = unique1 ^ unique2
print(f"Items in only one list: {only_in_one}")
```

10 Lab 8: Advanced Lists (Comprehensions & Map/Filter)

10.1 Scenario 1: Student Scores

```python
scores = [78, 85, 62, 90, 55]

# Filter scores > 70, square them, sum total
filtered = [s for s in scores if s > 70]
squared = [s**2 for s in filtered]
total = sum(squared)

print(f"Filtered: {filtered}")
print(f"Squared: {squared}")
print(f"Total: {total}")

# Bonus: Using lambda + map + filter
filtered_lambda = list(filter(lambda x: x > 70, scores))
```

```
squared_lambda = list(map(lambda x: x**2, filtered_lambda))
total_lambda = sum(squared_lambda)
print(f"Total (lambda): {total_lambda}")
```

10.2 Scenario 2: Product Prices

```
prices = [10, 25, 50, 5, 40]

# Apply 10% discount
discounted = [p * 0.9 for p in prices]

# Keep only discounted > 20
over_20 = [p for p in discounted if p > 20]

print(f"Discounted: {discounted}")
print(f"Over $20: {over_20}")

# Bonus: Nested comprehension with 8% tax
tax_rate = 0.08
final_prices = [p * (1 + tax_rate) for p in [pr * 0.9 for pr in prices] if p > 20]
print(f"Final prices (with tax): {[f'{p:.2f}' for p in final_prices]}")
```

11 Lab 8B: Lambda & Scope

11.1 Scenario 1: Sorting

```
# Sort list of dicts by specific key
employees = [
    {"name": "Alice", "age": 30, "salary": 50000},
    {"name": "Bob", "age": 25, "salary": 60000},
    {"name": "Charlie", "age": 35, "salary": 55000}
]

# Sort by salary
by_salary = sorted(employees, key=lambda x: x["salary"], reverse=True)
print("By salary:", [e["name"] for e in by_salary])

# Sort strings by length
words = ["python", "is", "awesome", "for", "programming"]
by_length = sorted(words, key=lambda x: len(x))
print(f"By length: {by_length}")

# Bonus: Sort by multiple criteria (age, then salary)
by_multi = sorted(employees, key=lambda x: (x["age"], -x["salary"]))
print("By age then salary:", [(e["name"], e["age"], e["salary"]) for e in by_multi
    ])
```

11.2 Scenario 2: Counter

```
# Counter using global variable
count = 0

def increment_global():
    global count
    count += 1
    return count

# Counter using closure
def make_counter():
    count = 0
    def increment():
        nonlocal count
        count += 1
        return count
    return increment

# Test global counter
print(f"Global: {increment_global()}, {increment_global()}, {increment_global()}")

# Test closure counter
counter1 = make_counter()
counter2 = make_counter()
print(f"Closure 1: {counter1()}, {counter1()}, {counter1()}")
print(f"Closure 2: {counter2()}, {counter2()}")  # Independent counter
```

12 Lab 9: Dictionaries

12.1 Scenario 1: Employee Records

```
employees = {
    "John": [25, "HR"],
    "Alice": [30, "IT"]
}

# Add new employee
employees["Bob"] = [28, "Finance"]

# Update department
employees["John"][1] = "Marketing"

print(f"All employees: {employees}")

# Bonus: Filter age > 28
over_28 = {k: v for k, v in employees.items() if v[0] > 28}
print(f"Over 28: {over_28}")
```

12.2 Scenario 2: Grades

```
grades = {
    "Alice": [85, 90, 88],
    "Bob": [78, 82, 80],
    "Charlie": [92, 95, 90]
}

# Compute average per student
averages = {name: sum(scores)/len(scores) for name, scores in grades.items()}
print(f"Averages: {averages}")

# Identify top students (avg > 85)
```

```
top_students = {name: avg for name, avg in averages.items() if avg > 85}
print(f"Top students: {top_students}")

# Bonus: Write to file
with open("top_students.txt", "w") as f:
    for name, avg in top_students.items():
        f.write(f"{name}: {avg:.2f}\n")
print("Top students written to file!")
```

13 Lab 10: File Handling

13.1 Scenario 1: Log File

```
# Write 5 log entries
log_entries = [
    "INFO: Application started",
    "ERROR: Connection failed",
    "INFO: User logged in",
    "ERROR: File not found",
    "INFO: Process completed"
]

with open("log.txt", "w") as f:
    for entry in log_entries:
        f.write(entry + "\n")

# Read and print only error lines
error_count = 0
with open("log.txt", "r") as f:
    for line in f:
        if "ERROR" in line:
            print(line.strip())
            error_count += 1

# Bonus: Count error lines
print(f"Total errors: {error_count}")
```

13.2 Scenario 2: Student Scores

```
# Write student grades
students = {
    "Alice": [85, 90, 88],
    "Bob": [78, 82, 80],
    "Charlie": [92, 95, 90]
}

with open("grades.txt", "w") as f:
    for name, scores in students.items():
        f.write(f"{name}: {','.join(map(str, scores))}\n")

# Read and compute average
with open("grades.txt", "r") as f:
    for line in f:
        name, scores_str = line.strip().split(": ")
        scores = list(map(int, scores_str.split(",")))
        avg = sum(scores) / len(scores)
        print(f"{name}: Average = {avg:.2f}")

        # Bonus: Filter students avg > 80
        if avg > 80:
            print(f"  -> Top performer!")
```

14 Lab 11: NumPy Arrays

14.1 Scenario 1: Matrix Operations

```
import numpy as np

# Create 3x3 matrix
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(f"Matrix:\n{matrix}")

# Flatten and filter > 5
flat = matrix.flatten()
filtered = flat[flat > 5]
print(f"Flattened: {flat}")
print(f"Filtered (>5): {filtered}")

# Compute sum, mean, std
print(f"Sum: {matrix.sum()}")
print(f"Mean: {matrix.mean()}")
print(f"Std: {matrix.std():.2f}")

# Bonus: Transpose and matrix multiply
transposed = matrix.T
result = matrix @ transposed
print(f"Matrix @ Transpose:\n{result}")
```

14.2 Scenario 2: Sensor Data

```
import numpy as np

# Create 1D array of 100 random readings
np.random.seed(42)
readings = np.random.rand(100) * 100  # 0-100

# Compute stats
print(f"Max: {readings.max():.2f}")
print(f"Min: {readings.min():.2f}")
print(f"Average: {readings.mean():.2f}")

# Boolean mask for readings > threshold
threshold = 75
above_threshold = readings[readings > threshold]
print(f"Above {threshold}: {len(above_threshold)} readings")

# Bonus: Reshape to 10x10 and compute row means
reshaped = readings.reshape(10, 10)
row_means = reshaped.mean(axis=1)
print(f"Row means: {row_means}")
```

15 Lab 11B: Pandas Data Structures

15.1 Scenario 1: Employee Database

```python
import pandas as pd

# Create DataFrame
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie', 'Diana', 'Eve'],
    'Age': [28, 35, 42, 31, 26],
    'Department': ['IT', 'HR', 'IT', 'Finance', 'HR'],
    'Salary': [55000, 48000, 72000, 61000, 45000]
})
print(df)

# Filter salary > 50000
high_salary = df[df['Salary'] > 50000]
print(f"\nSalary > 50000:\n{high_salary}")

# Group by department, compute average salary
avg_by_dept = df.groupby('Department')['Salary'].mean()
print(f"\nAverage salary by dept:\n{avg_by_dept}")

# Bonus: Add Bonus column (10% of salary)
df['Bonus'] = df['Salary'] * 0.1
print(f"\nWith Bonus:\n{df}")
```

15.2 Scenario 2: Sales Analysis

```python
import pandas as pd

# Create DataFrame
df = pd.DataFrame({
    'Product': ['Laptop', 'Mouse', 'Keyboard', 'Monitor', 'Laptop'],
    'Quantity': [5, 50, 30, 10, 3],
    'Price': [1000, 25, 75, 300, 1200],
    'Date': ['2026-01-01', '2026-01-01', '2026-01-02',
             '2026-01-02', '2026-01-03']
})

# Calculate total revenue
df['Revenue'] = df['Quantity'] * df['Price']
print(df)

# Find top 3 products by revenue
top_3 = df.nlargest(3, 'Revenue')[['Product', 'Revenue']]
print(f"\nTop 3 by revenue:\n{top_3}")

# Bonus: Group by date
daily_revenue = df.groupby('Date')['Revenue'].sum()
print(f"\nDaily revenue:\n{daily_revenue}")
```

16 Lab 12: JSON & Regex

16.1 Scenario 1: Student JSON

```python
import json

# JSON string of students
json_str = '''
{
    "students": [
        {"name": "Alice", "scores": [85, 90, 88]},
        {"name": "Bob", "scores": [78, 82, 75]},
        {"name": "Charlie", "scores": [92, 95, 90]}
    ]
}
'''

# Parse JSON
data = json.loads(json_str)

# Compute average per student
for student in data["students"]:
    name = student["name"]
    avg = sum(student["scores"]) / len(student["scores"])
    print(f"{name}: {avg:.2f}")

    # Bonus: Filter students avg > 80
    if avg > 80:
        print(f"  -> Honor Roll!")
```

16.2 Scenario 2: Email Validation

```python
import re

emails = [
    "user@example.com",
    "invalid-email",
    "test.user@domain.org",
    "bad@",
    "another@site.co.uk"
]

# Find valid emails using regex
pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'

valid_emails = []
for email in emails:
    if re.match(pattern, email):
        valid_emails.append(email)
        print(f"Valid: {email}")
    else:
        print(f"Invalid: {email}")

# Bonus: Replace domain
old_domain = "example.com"
new_domain = "newdomain.com"

updated = [re.sub(old_domain, new_domain, e) for e in valid_emails]
print(f"\nUpdated emails: {updated}")
```

17 Lab 13: Modules & Packages

17.1 Scenario 1: Utility Module (mymath.py)

```python
# File: mymath.py
"""
Math utility module with helper functions.
"""

def add(a, b):
    """Return the sum of two numbers."""
    return a + b

def multiply(a, b):
    """Return the product of two numbers."""
    return a * b

def factorial(n):
    """Return the factorial of n."""
    if n <= 1:
        return 1
    return n * factorial(n - 1)

def is_prime(n):
    """Check if n is a prime number."""
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

# File: main.py (using the module)
from mymath import add, multiply, factorial, is_prime

print(f"5 + 3 = {add(5, 3)}")
print(f"5 * 3 = {multiply(5, 3)}")
print(f"5! = {factorial(5)}")
print(f"Is 17 prime? {is_prime(17)}")
```

17.2 Scenario 2: Standard Library

```python
import os
import datetime
from collections import Counter, defaultdict

# os module for file operations
current_dir = os.getcwd()
files = os.listdir(current_dir)
print(f"Current directory: {current_dir}")
print(f"Files: {files[:5]}")  # First 5 files

# datetime for date calculations
today = datetime.date.today()
future = today + datetime.timedelta(days=30)
print(f"Today: {today}")
print(f"30 days from now: {future}")
```

```

# Bonus: collections module
words = ["apple", "banana", "apple", "cherry", "banana", "apple"]
word_counts = Counter(words)
print(f"Word counts: {word_counts}")

# defaultdict for grouping
grades = defaultdict(list)
grades["Alice"].append(85)
grades["Bob"].append(78)
grades["Alice"].append(90)
print(f"Grades: {dict(grades)}")
```