CPSC 250 - Programming for Data Manipulation
Final Exam Review – Solutions

Part I: Multiple Choice (30 points)

1. **C** – A single-element tuple must use a trailing comma: (5,) is a tuple.
2. **C** – ___lt___ implements the less-than operator <.
3. **D** – x is y is True because they point to the same list object.
4. **A** – plt.hist() creates a histogram from data.
5. **B** – Encapsulation hides internal state and data using private attributes.
6. **C** – Default arguments must come after required ones.
7. **A** – df.loc[0] returns the first row (as a Series).
8. **B** – In a balanced BST, search is O(log n).
9. **B** – OLS() from statsmodels.api performs regression.
10. **B** – The derived class method overrides the base method.
11. **B** – Inheritance lets a class reuse code from another class.
12. **A** – Python does not support true overloading, but default args allow flexibility.
13. **C** – The set comprehension eliminates duplicates: {1, 2, 3}.
14. **C** – Tuples are immutable.
15. **C** – np.linspace() generates evenly spaced values.

Part II: Error Identification (15 points)

Original Buggy Code:

```
def count_words(filename):
    with open(filename) as f:
        words = f.read().split()

    counts = {}
    for word in words:
        if word in counts:
            counts[word] = 1
        else:
            counts[word] += 1

    return counts
```

Fixed Code:

```
def count_words(filename):
    with open(filename) as f:
        words = f.read().split()

    counts = {}
    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1

    return counts
```

Explanation: The original logic reversed the update: it reset the count to 1 if the word already existed. Also, the default case incorrectly tried to increment a non-existent key.

Part III: Code Writing (30 points)

Q1. Recursive factorial (6 points)

```
def factorial(n):
    if n <= 1:
        return 1
    else:
        return n * factorial(n - 1)
```

Q2. Book class with __str__ (8 points)

```
class Book:
    def __init__(self, title, author):
        self.__title = title
        self.__author = author

    def __str__(self):
        return f'"{self.__title}" by {self.__author}'
```

Q3. Read CSV and plot (8 points)

```
import pandas as pd
import matplotlib.pyplot as plt

def read_and_plot(filename):
    df = pd.read_csv(filename)
    plt.scatter(df['x'], df['y'])
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Scatterplot of y vs x")
    plt.show()
```

Q4. Shape polymorphism (8 points)

```
class Shape:
    def area(self):
        return 0

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
```

```
        return self.side ** 2

class Triangle(Shape):
    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height

# Demonstrate polymorphism
shapes = [Square(4), Triangle(3, 6), Square(2)]
for s in shapes:
    print(s.area())
```

Part IV: Code Comprehension + Commenting (25 points)

Original Function:

```
import statsmodels.api as sm
import pandas as pd

def regress(df, yname, xnames):
    X = df[xnames]                    # Select independent variables
    X = sm.add_constant(X)            # Add intercept term
    y = df[yname]                     # Select dependent variable
    model = sm.OLS(y, X)              # Create OLS regression model
    results = model.fit()             # Fit the model
    print(results.summary())          # Print regression summary
```

Explanation: This function performs a multiple linear regression using the specified dependent variable (‘yname‘) and a list of independent variables (‘xnames‘) from a DataFrame. It adds a constant term (intercept), fits the model using OLS (ordinary least squares), and prints a detailed summary with coefficients, R-squared value, and p-values.