1. `load_data_from_s3`:

```
Function load_data_from_s3():
    Initialize local_file as 'heart_disease.csv'
    Download the file from S3 bucket (S3_BUCKET, S3_FILE) to
local_file using S3 client
    Read the downloaded CSV into a pandas DataFrame (df) using
pd.read_csv(local_file)
    Return df
```

2. `clean_data`:

```
Function clean_data(df):
    Define columns_to_retain as a list containing necessary column
names from the DataFrame
    Create df_cleaned by retaining only the columns in
columns_to_retain

    For column in ['painloc', 'painexer']:
        Fill missing values in column using mode of column

    For column 'trestbps':
        Clip values less than 100 to 100 (minimum threshold)

    For column 'oldpeak':
        Clip values below 0 to 0 and values above 4 to 4

    For column in ['thaldur', 'thalach']:
        Fill missing values using mean of respective column

    For columns in ['fbs', 'prop', 'nitr', 'pro', 'diuretic']:
        Fill missing values using mode of the column
        Clip values greater than 1 to 1

    For binary column 'exang' and 'slope' with discrete values:
        Fill missing values using mode of the column
```

```
    Fill missing values in other continuous columns (age, sex, cp,
trestbps, oldpeak, target) using mode or mean

    Trim df_cleaned to the first 899 rows to correct for improper
formatting beyond this point

    Return df_cleaned
```

3. impute_smoking_1:

```
Function impute_smoking_1(url, df_cleaned):
    Send GET request to url to fetch webpage
    If response status code is 200 (success):
        Parse response text using BeautifulSoup to extract the
required table
        Initialize empty dictionary smoking_rate_by_age

        Loop through each row in the relevant table to extract age
group and smoking rates
        Populate smoking_rate_by_age with age group as key and smoking
rate as value

        Add a new column 'smoke_source_1' as a copy of the 'smoke'
column in df_cleaned

        Define a helper function get_age_group(age):
            Map age to its corresponding age group (e.g., '15–17',
'18–24', etc.)

        Define a helper function impute_smoking_rate(row):
            If 'smoke_source_1' is NaN:
                Use get_age_group to find corresponding age group
                Look up the smoking rate for the age group in
smoking_rate_by_age
                Return the corresponding smoking rate
            Else return the original 'smoke_source_1' value
```

```
        Apply impute_smoking_rate to df_cleaned to fill missing values
in 'smoke_source_1'

    Return df_cleaned with updated 'smoke_source_1'
```

4. `impute_smoking_2`:

```
Function impute_smoking_2(df_cleaned):
    Define smoking_rate_female as 0.1 (smoking rate for females)
    Define smoking_rate_male as 0.132 (smoking rate for males)
    Define smoking_rate_by_age dictionary with age groups and
corresponding smoking rates

    Add a new column 'smoke_source_2' as a copy of the 'smoke' column
in df_cleaned

    Define a helper function get_age_group(age):
        Map age to its corresponding age group (e.g., '18–24',
'25–44', etc.)

    Define a helper function impute_smoking_rate(row):
        If 'smoke_source_2' is NaN:
            Use get_age_group to map age to age group
            If age group exists in smoking_rate_by_age:
                If sex is female (0), return the corresponding smoking
rate from smoking_rate_by_age
                Else if sex is male (1), adjust the smoking rate by
multiplying with the ratio of male to female smoking rates
        Else return the original 'smoke_source_2' value

    Apply impute_smoking_rate to df_cleaned to fill missing values in
'smoke_source_2'

    Return df_cleaned with updated 'smoke_source_2'
```

5. `train_heart_disease_model`:

```
Function train_heart_disease_model(df_cleaned):
```

```
    Split df_cleaned into features (X) and target (y) based on
DataFrame columns

    Initialize models:
        - Logistic Regression (log_reg)
        - Random Forest Classifier (rf_model)
        - Support Vector Classifier (svm_model)
        - XGBoost Classifier (xgb_model)

    Perform 90-10 train-test split using StratifiedKFold to maintain
class balance

    For each model:
        Perform 5-fold cross-validation to evaluate model performance
(cross_val_score)
        Tune hyperparameters using GridSearchCV to optimize model
performance

    Choose the best model based on highest F1-score or ROC-AUC score

    Train the best model on the entire training set and make
predictions on the test set

    Calculate and print metrics:
        - Accuracy score
        - F1-score
        - Classification report
        - ROC-AUC score

    Return the trained model
```

6. `main`:

```
Function main():
    Print "Loading data from S3..."
    df = load_data_from_s3()

    Print "Cleaning data..."
```

```
df_cleaned = clean_data(df)

Print "Imputing missing smoking data (method 1)..."
df_cleaned = impute_smoking_1(url, df_cleaned)

Print "Imputing missing smoking data (method 2)..."
df_cleaned = impute_smoking_2(df_cleaned)

Print "Training heart disease prediction model..."
model = train_heart_disease_model(df_cleaned)

Save the trained model to disk using pickle (optional)

Print "Process completed successfully!"
```