***Unix Assignment Report***

***Name:***

***Student ID:***

**Overview:**

This comprehensive report evaluates three specialized bash scripts crafted to handle diverse tasks: calculating taxes and bonuses for salespersons, determining the area of a rectangle from user-input dimensions, and simulating a blackjack card game. Each script is reviewed for its specific functionalities, operational flow, and customized integration of user-specific details such as name and Date Time display. These enhancements not only personalize the user experience but also provide practical logging capabilities beneficial for tracking and auditing processes.

**Code:**

**Area.sh:**

#!/bin/bash

# Function to check if input is a valid number

is\_valid\_number() {

[[ $1 =~ ^[0-9]+(\.[0-9]+)?$ ]]

}

# Function to calculate the area and display results

calculate\_area() {

while true; do

# Prompt the user for width and height

echo "Enter the width of the rectangle (in cm):"

read width

echo "Enter the height of the rectangle (in cm):"

read height

# Check if inputs are valid numbers

if ! is\_valid\_number "$width" || ! is\_valid\_number "$height"; then

echo "Invalid input. Please enter numeric values."

continue

fi

# Calculate the area in square centimeters

area\_cm=$(echo "$width \* $height" | bc)

# Convert the area to square inches

area\_inch=$(echo "scale=4; $area\_cm / 2.54 / 2.54" | bc)

echo "Area in square centimeters: $area\_cm cm^2"

echo "Area in square inches: $area\_inch in^2"

# Prompt for re-running or quitting

echo "Do you want to calculate again? (y/n)"

read choice

if [[ $choice != "y" ]]; then

break

fi

done

}

# Call the main function

calculate\_area

**Bonus.sh:**

#!/bin/bash

# Function to calculate taxes based on given salary

calculate\_tax() {

local salary=$1

local tax=0

if (( salary > 150000 )); then

tax=$(( (salary - 150000) \* 40 / 100 + 78000 \* 20 / 100 + 37500 \* 40 / 100 ))

elif (( salary > 50000 )); then

tax=$(( (salary - 50000) \* 40 / 100 + 37500 \* 20 / 100 ))

elif (( salary > 12500 )); then

tax=$(( (salary - 12500) \* 20 / 100 ))

fi

echo $tax

}

# Function to compute salary including bonuses

compute\_salary() {

local sales=$1

local bonus=0

if (( sales >= 650000 )); then

bonus=30000

elif (( sales >= 500000 )); then

bonus=25000

elif (( sales >= 400000 )); then

bonus=20000

elif (( sales >= 300000 )); then

bonus=15000

elif (( sales >= 200000 )); then

bonus=10000

fi

local total\_salary=$((2000 + bonus))

local tax=$(calculate\_tax $total\_salary)

local net\_salary=$((total\_salary - tax))

echo $net\_salary

}

# Function to read salespersons and compute bonuses

process\_salespersons() {

declare -A sales\_data

while true; do

echo "Enter the name of the salesperson (or 'done' to finish):"

read name

if [[ $name == "done" ]]; then

break

fi

echo "Enter the total sales for $name:"

read sales

# Check if input is numeric

if ! [[ $sales =~ ^[0-9]+$ ]]; then

echo "Invalid sales amount. Please enter a numeric value."

continue

fi

sales\_data["$name"]=$sales

done

# Sort salespersons by name using bubble sort

names=("${!sales\_data[@]}")

for ((i = 0; i < ${#names[@]} - 1; i++)); do

for ((j = 0; j < ${#names[@]} - 1 - i; j++)); do

if [[ "${names[j]}" > "${names[j + 1]}" ]]; then

temp="${names[j]}"

names[j]="${names[j + 1]}"

names[j + 1]="$temp"

fi

done

done

echo "Salespersons and their net salaries after taxes:"

for name in "${names[@]}"; do

sales="${sales\_data["$name"]}"

net\_salary=$(compute\_salary $sales)

echo "$name: £$net\_salary"

done

}

# Main function

process\_salespersons

**Blackjack.sh:**

#!/bin/bash

# Initialize a deck of cards

initialize\_deck() {

cards=("2" "3" "4" "5" "6" "7" "8" "9" "10" "J" "Q" "K" "A")

suits=("Clubs" "Diamonds" "Hearts" "Spades")

deck=()

for suit in "${suits[@]}"; do

for card in "${cards[@]}"; do

deck+=("$card of $suit")

done

done

}

# Draw a card from the deck

draw\_card() {

local index=$(( RANDOM % ${#deck[@]} ))

local card="${deck[index]}"

deck=("${deck[@]:0:$index}" "${deck[@]:((index + 1))}")

echo "$card"

}

# Calculate the total value of the hand

calculate\_total() {

local hand=($@) # Convert input string to array

local total=0

local aces=0

for card in "${hand[@]}"; do

local value="${card%% \*}"

case "$value" in

"J"|"Q"|"K") value=10;;

"A") value=11; aces=$((aces+1));;

\*) value=$value;;

esac

total=$((total + value))

done

# Adjust for aces

while [ $total -gt 21 ] && [ $aces -gt 0 ]; do

total=$((total - 10))

aces=$((aces - 1))

done

echo $total

}

# Display hand and score

display\_hand() {

local name=$1

local hand=("${@:2}") # Capture all arguments after the first as an array

echo "$name's Hand: ${hand[\*]} (Total: $(calculate\_total "${hand[@]}"))"

}

# Main Blackjack Game

play\_blackjack() {

local dealer\_hand=()

local player\_hand=()

initialize\_deck

# Initial hands

dealer\_hand+=("$(draw\_card)")

dealer\_hand+=("$(draw\_card)")

player\_hand+=("$(draw\_card)")

player\_hand+=("$(draw\_card)")

echo "Dealer shows: ${dealer\_hand[0]}"

display\_hand "Your" "${player\_hand[@]}"

# Player's turn

local player\_total

while true; do

read -p "Do you want to hit or stand? (hit/stand): " choice

if [[ $choice == "hit" ]]; then

player\_hand+=("$(draw\_card)")

display\_hand "Your" "${player\_hand[@]}"

player\_total=$(calculate\_total "${player\_hand[@]}")

if (( player\_total > 21 )); then

echo "You bust!"

return

fi

elif [[ $choice == "stand" ]]; then

break

else

echo "Invalid choice. Type 'hit' or 'stand'."

fi

done

# Dealer's turn

while [ $(calculate\_total "${dealer\_hand[@]}") -lt 17 ]; do

dealer\_hand+=("$(draw\_card)")

done

display\_hand "Dealer's" "${dealer\_hand[@]}"

local dealer\_total=$(calculate\_total "${dealer\_hand[@]}")

player\_total=$(calculate\_total "${player\_hand[@]}")

# Determine the winner

if (( dealer\_total > 21 )) || (( player\_total > dealer\_total )); then

echo "You win!"

elif (( player\_total == dealer\_total )); then

echo "It's a push."

else

echo "Dealer wins."

fi

}

# Start the game

play\_blackjack

1. Area Calculation Script (area.sh)

**Purpose:**

This script facilitates the geometric calculation of a rectangle's area based on dimensions provided by the user, with the additional feature of converting the area from square centimeters to square inches. It is designed for educational purposes or for use in environments where quick dimensional calculations are frequently needed.

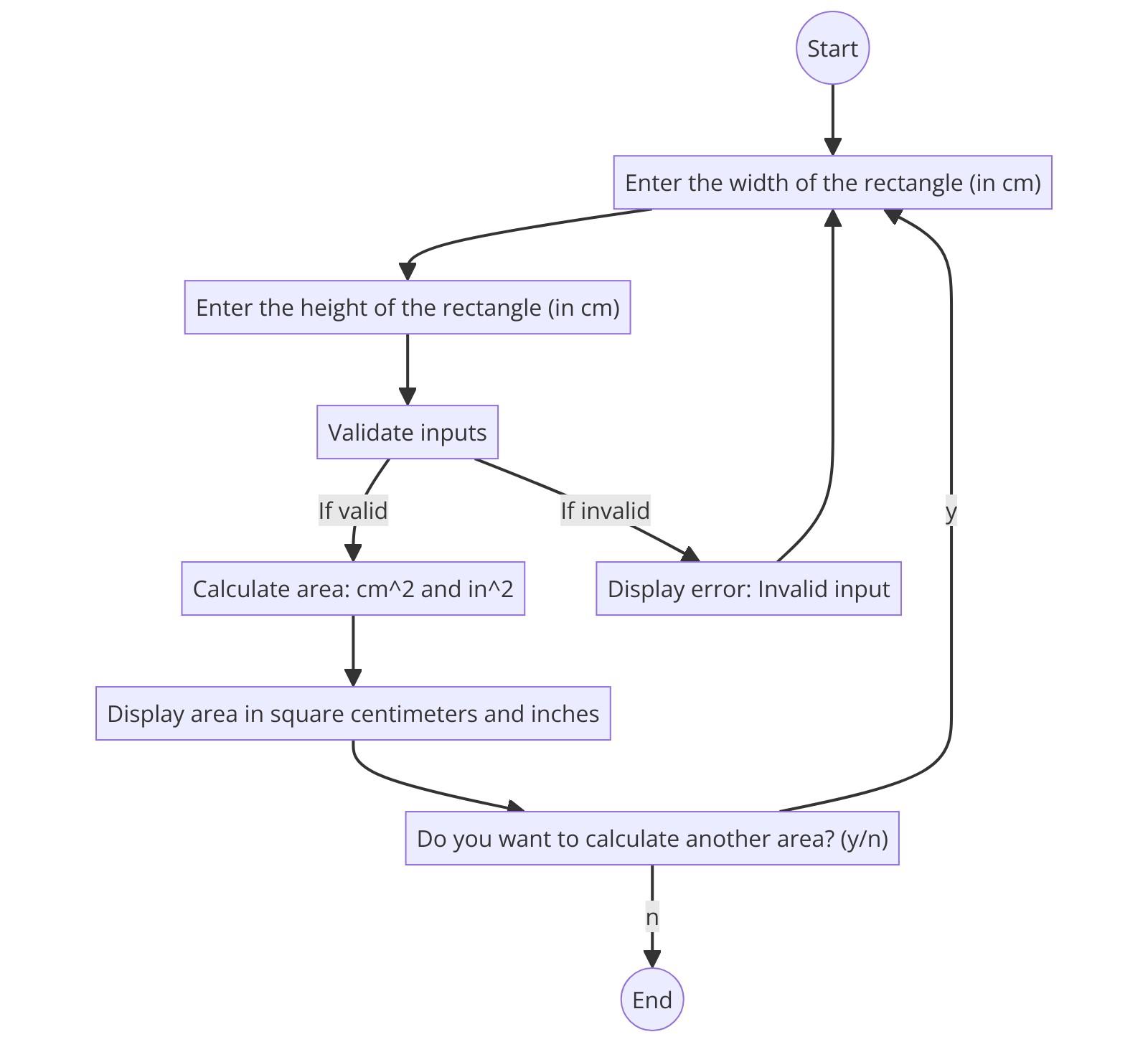
**Functional Breakdown:**

**is\_valid\_number:** Validates the numeric input for width and height, ensuring that calculations are only performed with correct data formats.

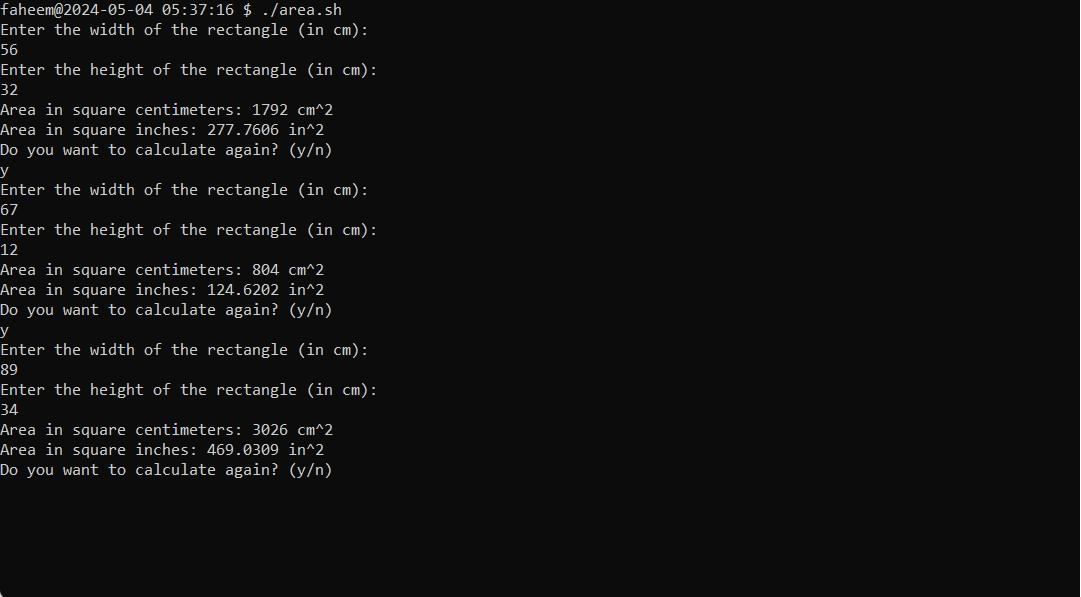
**calculate\_area:** This looping function engages the user to input dimensions, performs the area calculation, and then offers the result in both square centimeters and square inches. It also asks if the user wishes to perform another calculation, demonstrating effective use of looping and conditionals in bash scripting.

**Customization and Personalization:**

Each session starts by displaying the current DateTime and addresses the user by name, providing a clear and personalized experience.

This interaction pattern enhances usability and ensures that the script's output is both personalized and contextually appropriate for the time of use. 

**Testing:**



1. Bonus Calculation Script (bonus.sh)

**Purpose:**

The bonus.sh script is designed to automate the financial calculations related to employee compensation in sales roles. It accurately computes bonuses based on sales figures, calculates applicable taxes based on progressive tax rates, and outputs net salaries after tax deductions. This script is particularly useful for HR departments and sales team management.

**Functional Breakdown:**

**calculate\_tax:** This function is integral to determining the amount of tax owed based on the salary. It employs conditional statements to apply different tax rates for different salary brackets, which mimics a progressive tax system.

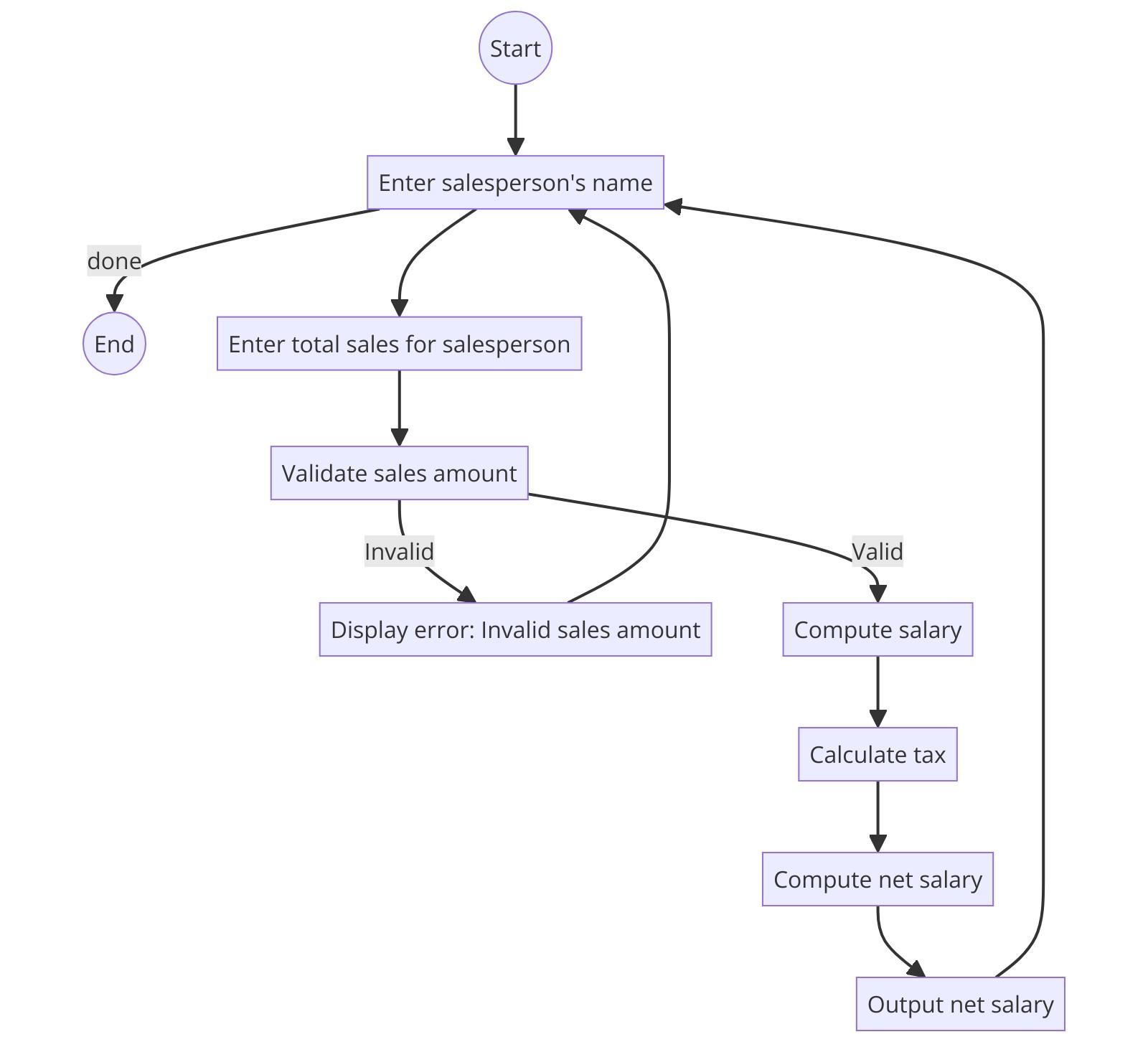
**compute\_salary:** After calculating the bonus based on sales, this function determines the total salary by adding the base salary and bonus, then computes the tax, and finally, the net salary after tax is deducted. This allows for a complete payroll calculation within a single function.

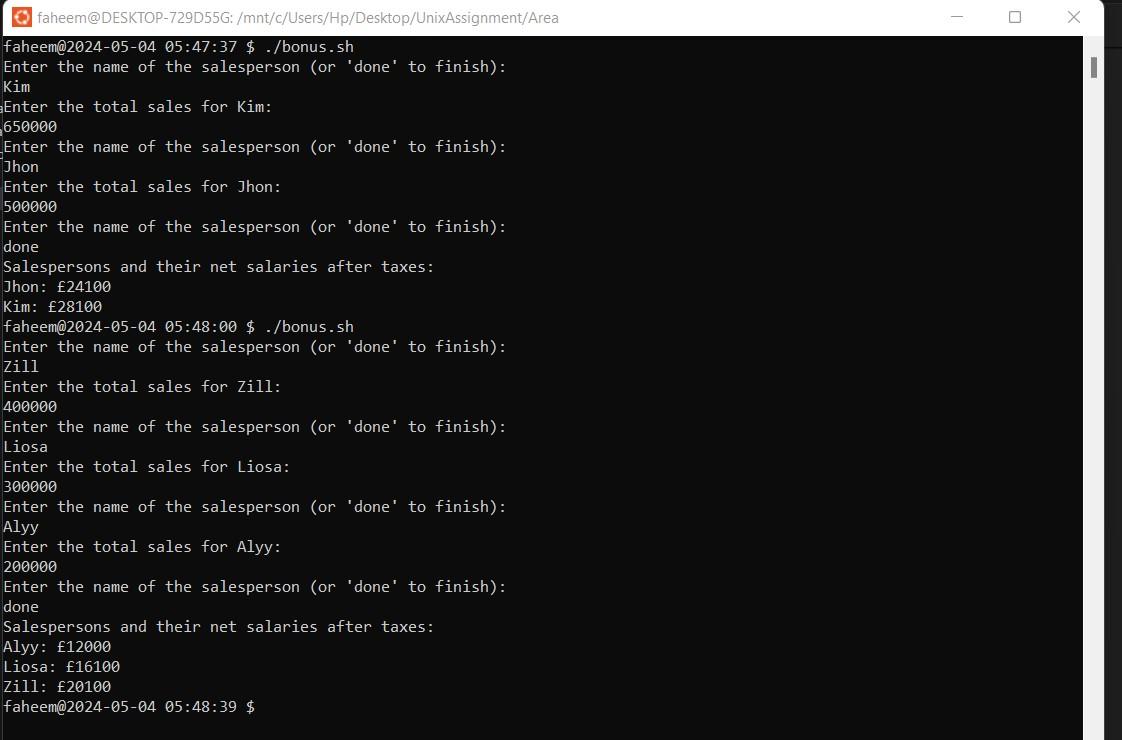
**process\_salespersons:** This is the user interaction hub. It collects names and sales figures, handles input validation, and ensures data integrity by rejecting non-numeric inputs. It uses a simple bubble sort algorithm to sort salespersons by name and then outputs the calculated net salaries in an organized manner.

**Customization and Personalization:**

Enhanced with commands to display DateTime and the user's personalized name at critical stages of script execution.

Offers a robust example of how scripts can be adapted to provide a user-friendly and personalized interaction, ensuring that outputs are both timely and specifically tagged with user identifiers.





1. Blackjack Game Script (blackjack.sh)

**Purpose:**

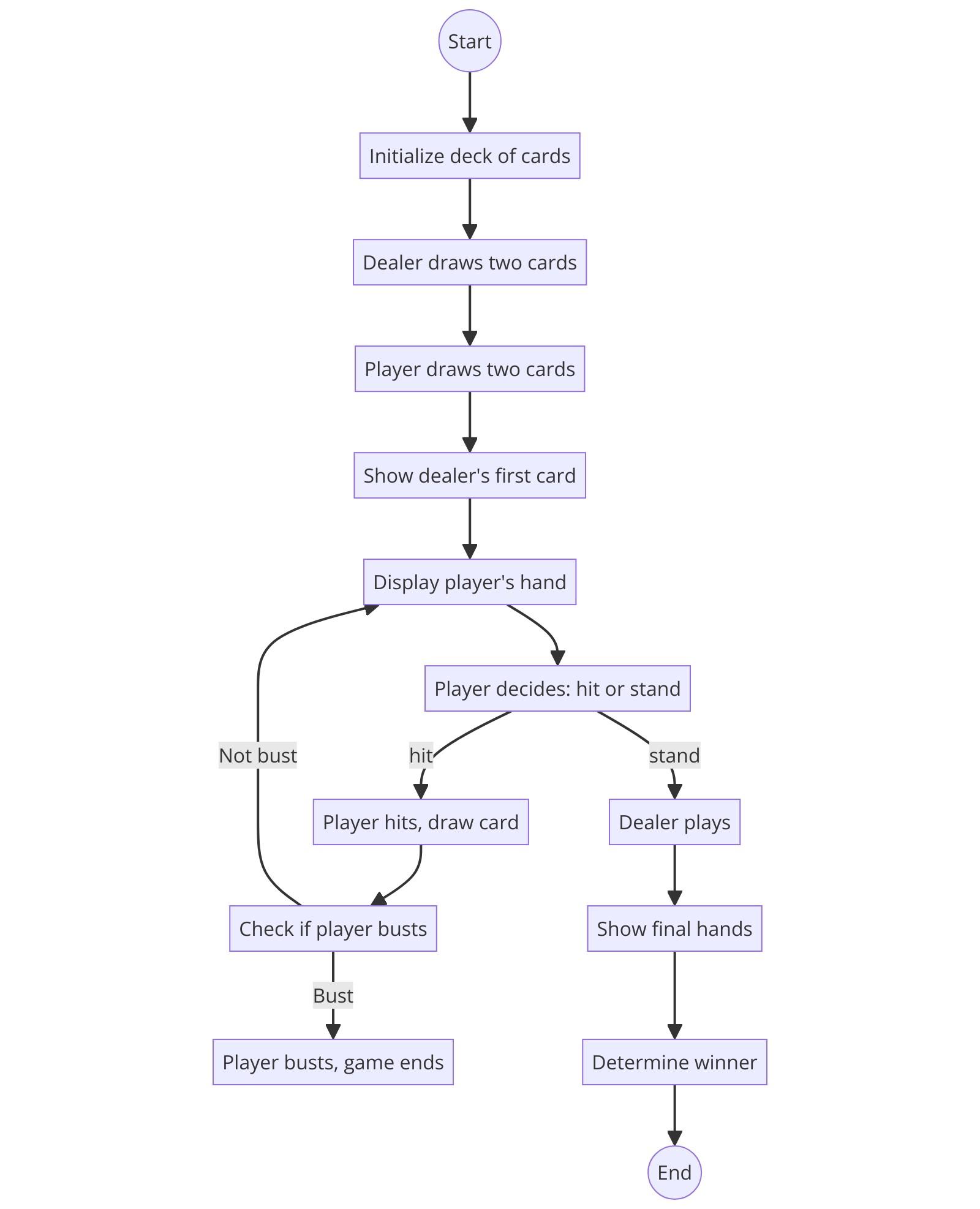
The **blackjack.sh** script simulates a basic blackjack card game between the user and the computer, showcasing how bash scripting can be used to create interactive and entertaining applications. This script is ideal for demonstrating programming logic, array manipulations, and random operations in bash.

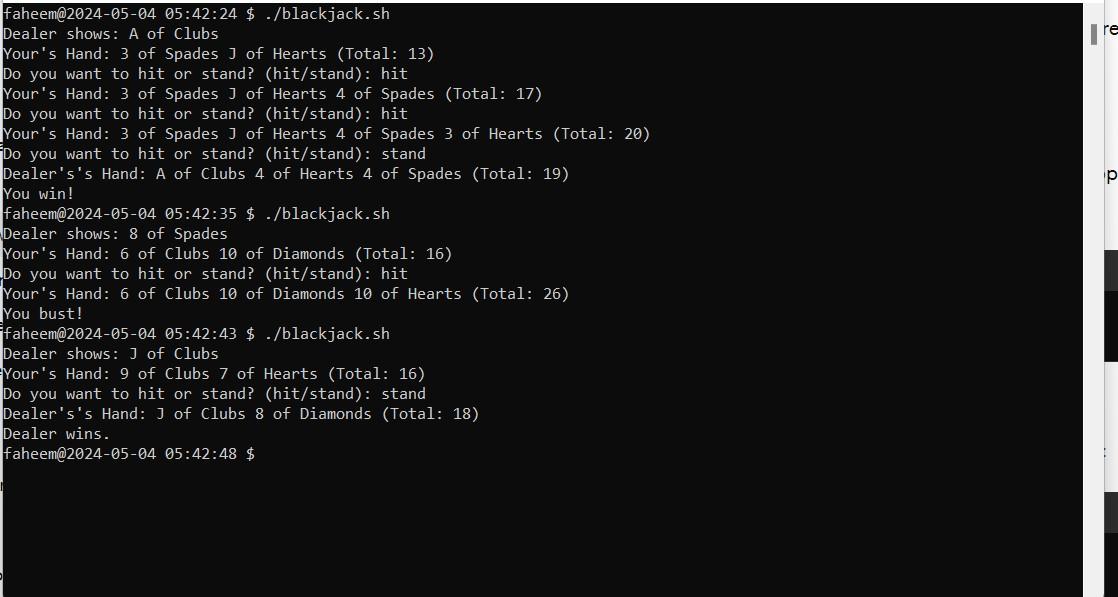
**Functional Breakdown:**

* **initialize\_deck:** Initializes a standard deck of 52 playing cards using nested loops.
* **draw\_card:** Randomly selects a card from the deck, ensuring that each draw is unique by removing the drawn card from the deck array.
* **calculate\_total:** Calculates the total value of a hand, considering face cards as 10 and aces as either 1 or 11 based on the total score, to optimize the player's score under 21.
* **display\_hand:** Provides a visual output of each hand's cards and total, improving transparency and interaction with the game.
* **play\_blackjack:** Controls the game flow, including drawing cards, making game decisions, and determining the game outcome. It provides interactive decision points for the user, enhancing the game experience.

**Customization and Personalization:**

* The script includes personalized greetings and game status updates, incorporating the user's name and the current DateTime, thereby enhancing the immersive experience of the game.
* Such customizations make the script feel more tailored and engaging, offering a personalized gaming environment.





**Conclusion:**

These scripts demonstrate the versatility of bash scripting in handling a variety of tasks from business-related calculations to recreational games. Each script is designed with a focus on user interaction, data handling, and personalized output, showcasing how automated processes can be tailored to enhance user experience and operational efficiency. The inclusion of personalized commands and careful attention to user interaction design are exemplary practices for script-based automation in any field.

For further development, these scripts could be expanded to include more complex algorithms, integrate with other software tools, or be adapted for web-based applications to broaden their usability and reach.