



**SAULT**  
COLLEGE

## Sault College

Name: Jangbu Sherpa

Student ID: 22054069

Professor: Nawaz Chowdhury

Course: Programming Concepts 2

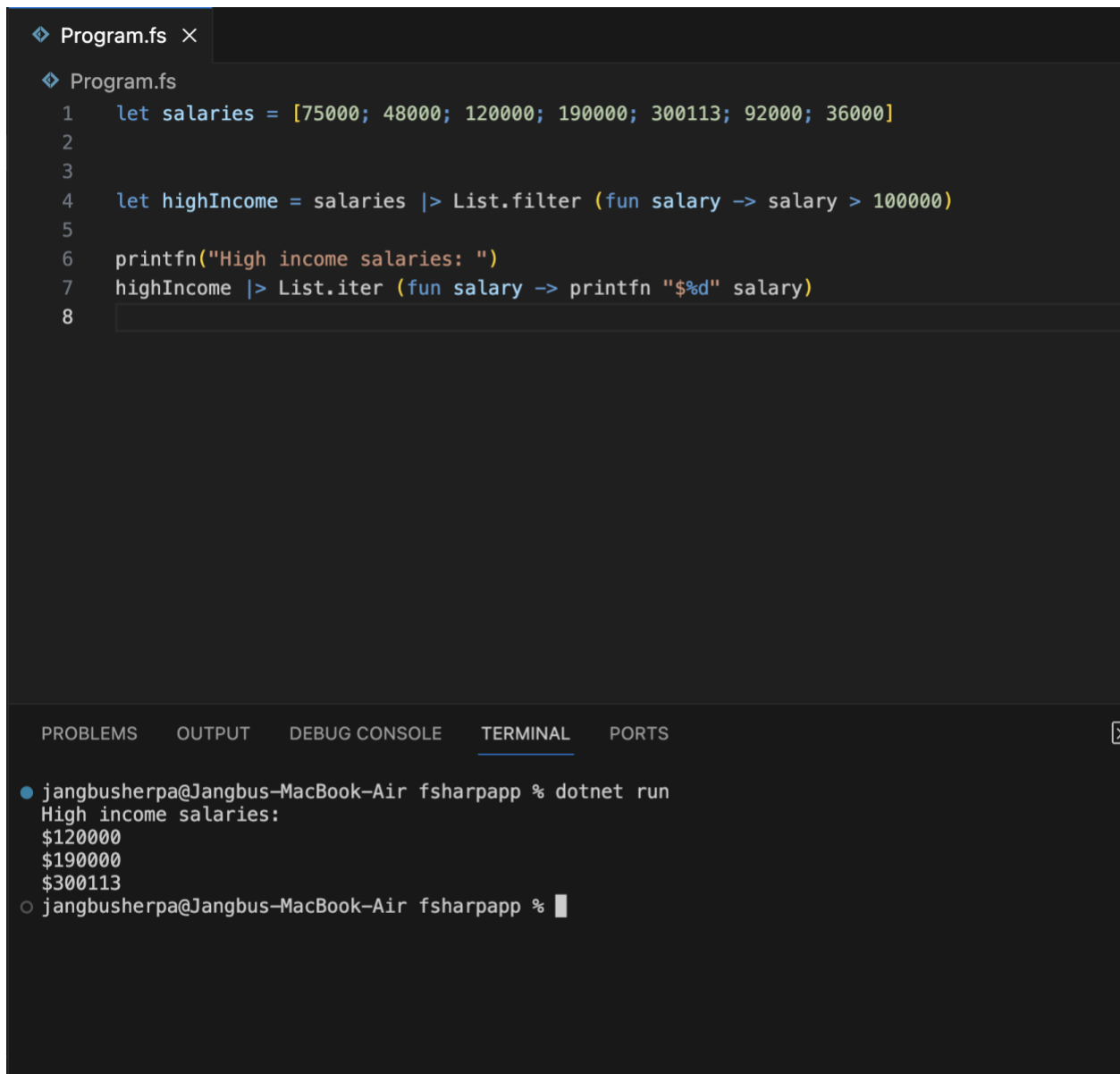
Code: CSD-215

Due Date: In-class

## Table of Contents

<b>1. Create a list that displays the salaries of a company's employees. For this list, let salaries = [\$75,000; \$48,000; \$120,000; \$190,000; \$300,113; \$92,000; \$36,000].</b>	<b>3</b>
a. Filter through the list to find high-income salaries. For this list, salaries above \$100,000 are considered high.	3
b. Use the map function to calculate tax for all salaries based on the table provided.	4
c. Filter salaries less than \$49,020 and add \$20,000 to these salaries using the map function.	5
d. Filter salaries between \$50,000 and \$100,000 and sum them all using the reduce/fold function.	6
<b>2. Tail Recursion Use tail recursion to write a program that will calculate the sum of all multiples of 3 up to a given number. Assume that we pass only a multiple of 3 as a parameter to this function. No validation is needed.</b>	<b>7</b>
<b>3. Uploading Repo to github</b>	<b>8</b>

1. Create a list that displays the salaries of a company's employees.  
For this list, let salaries = [\$75,000; \$48,000; \$120,000; \$190,000; \$300,113; \$92,000; \$36,000].
  - a. Filter through the list to find high-income salaries. For this list, salaries above \$100,000 are considered high.



The screenshot shows a Visual Studio Code editor with a file named `Program.fs`. The code defines a list of salaries, filters for high-income salaries (above \$100,000), and prints them. The terminal output shows the execution of the program, displaying the high-income salaries: \$120,000, \$190,000, and \$300,113.

```
Program.fs
1 let salaries = [75000; 48000; 120000; 190000; 300113; 92000; 36000]
2
3
4 let highIncome = salaries |> List.filter (fun salary -> salary > 100000)
5
6 printfn("High income salaries: ")
7 highIncome |> List.iter (fun salary -> printfn "$%d" salary)
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● jangbusherpa@Jangbus-MacBook-Air fsharpapp % dotnet run
High income salaries:
$120000
$190000
$300113
○ jangbusherpa@Jangbus-MacBook-Air fsharpapp %
```

b. Use the map function to calculate tax for all salaries based on the table provided.

```
Program.fs x
Program.fs
1  let TaxCalc salary =
2      match salary with
3      | s when s <= 49020 -> int (float s * 0.15)
4      | s when s <= 98040 -> int (float s * 0.205)
5      | s when s <= 151978 -> int (float s * 0.26)
6      | s when s <= 216511 -> int (float s * 0.29)
7      | _ -> int (float salary * 0.33)
8
9      let salaries = [75000; 48000; 120000; 190000; 300113; 92000; 36000]
10
11      let taxes = salaries |> List.map TaxCalc
12
13      taxes |> List.iteri (fun i tax -> printfn "Salary: %d, Tax: %d" (List.item i salaries) tax)
14
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS zsh + -

```
● jangbusherpa@Jangbus-MacBook-Air fsharpapp % dotnet run
Salary: $75000, Tax: $15374
Salary: $48000, Tax: $7200
Salary: $120000, Tax: $31200
Salary: $190000, Tax: $55099
Salary: $300113, Tax: $99037
Salary: $92000, Tax: $18860
Salary: $36000, Tax: $5400
○ jangbusherpa@Jangbus-MacBook-Air fsharpapp %
```

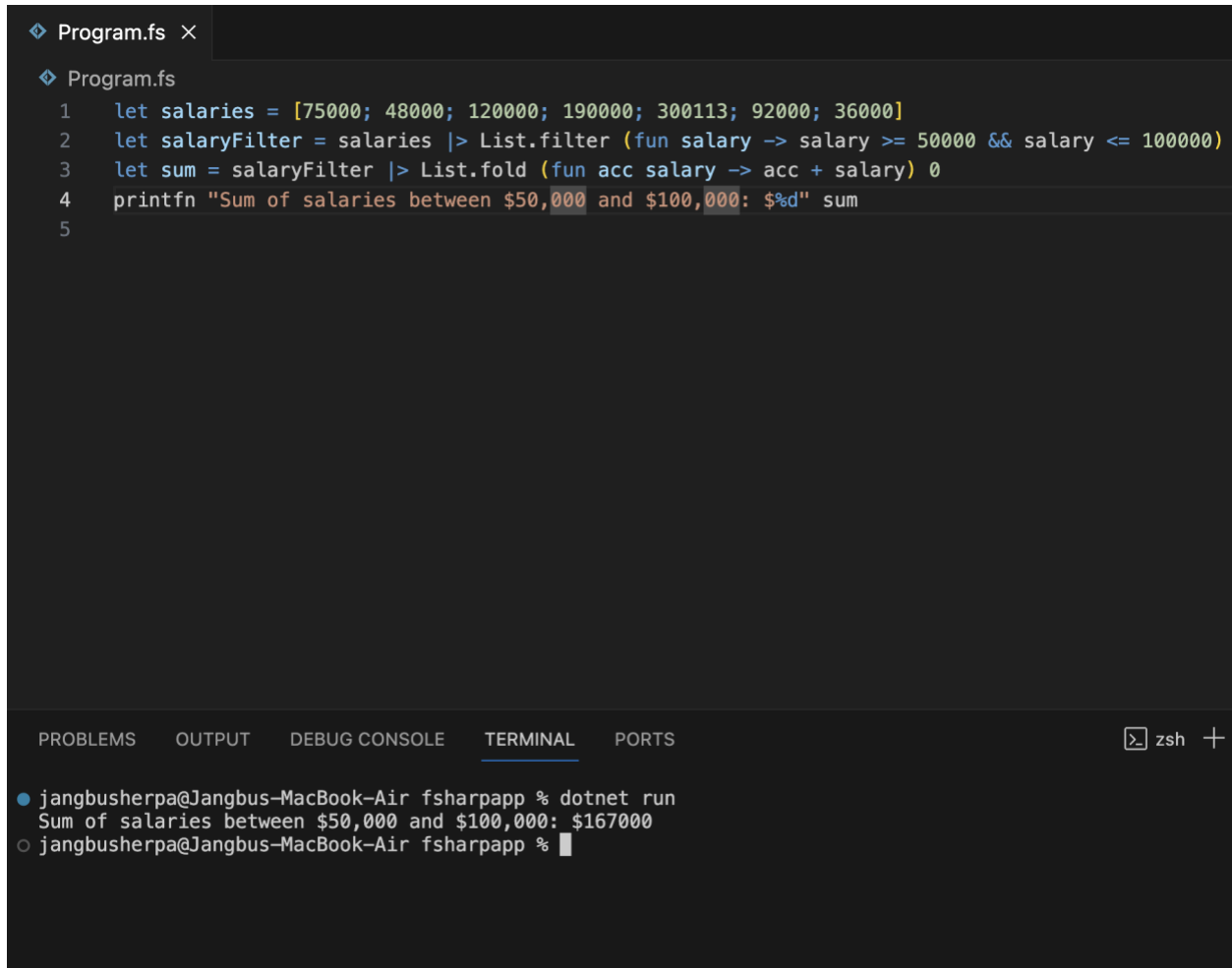
- c. Filter salaries less than \$49,020 and add \$20,000 to these salaries using the map function.

```
Program.fs X
Program.fs
1  let salaries = [75000; 48000; 120000; 190000; 300113; 92000; 36000]
2
3
4  let afterSalary =
5      salaries
6      |> List.filter (fun salary -> salary < 49020)
7      |> List.map (fun salary -> salary + 20000)
8
9
10 afterSalary |> List.iter (fun salary -> printfn "Salaries After Addition: %d" salary)
11
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● jangbusherpa@Jangbus-MacBook-Air fsharpapp % dotnet run
Salaries After Addition: $68000
Salaries After Addition: $56000
○ jangbusherpa@Jangbus-MacBook-Air fsharpapp %
```

- d. Filter salaries between \$50,000 and \$100,000 and sum them all using the reduce/fold function.



The image shows a Visual Studio Code editor window with a file named `Program.fs`. The code in the editor is as follows:

```
1 let salaries = [75000; 48000; 120000; 190000; 300113; 92000; 36000]
2 let salaryFilter = salaries |> List.filter (fun salary -> salary >= 50000 && salary <= 100000)
3 let sum = salaryFilter |> List.fold (fun acc salary -> acc + salary) 0
4 printfn "Sum of salaries between $50,000 and $100,000: %d" sum
5
```

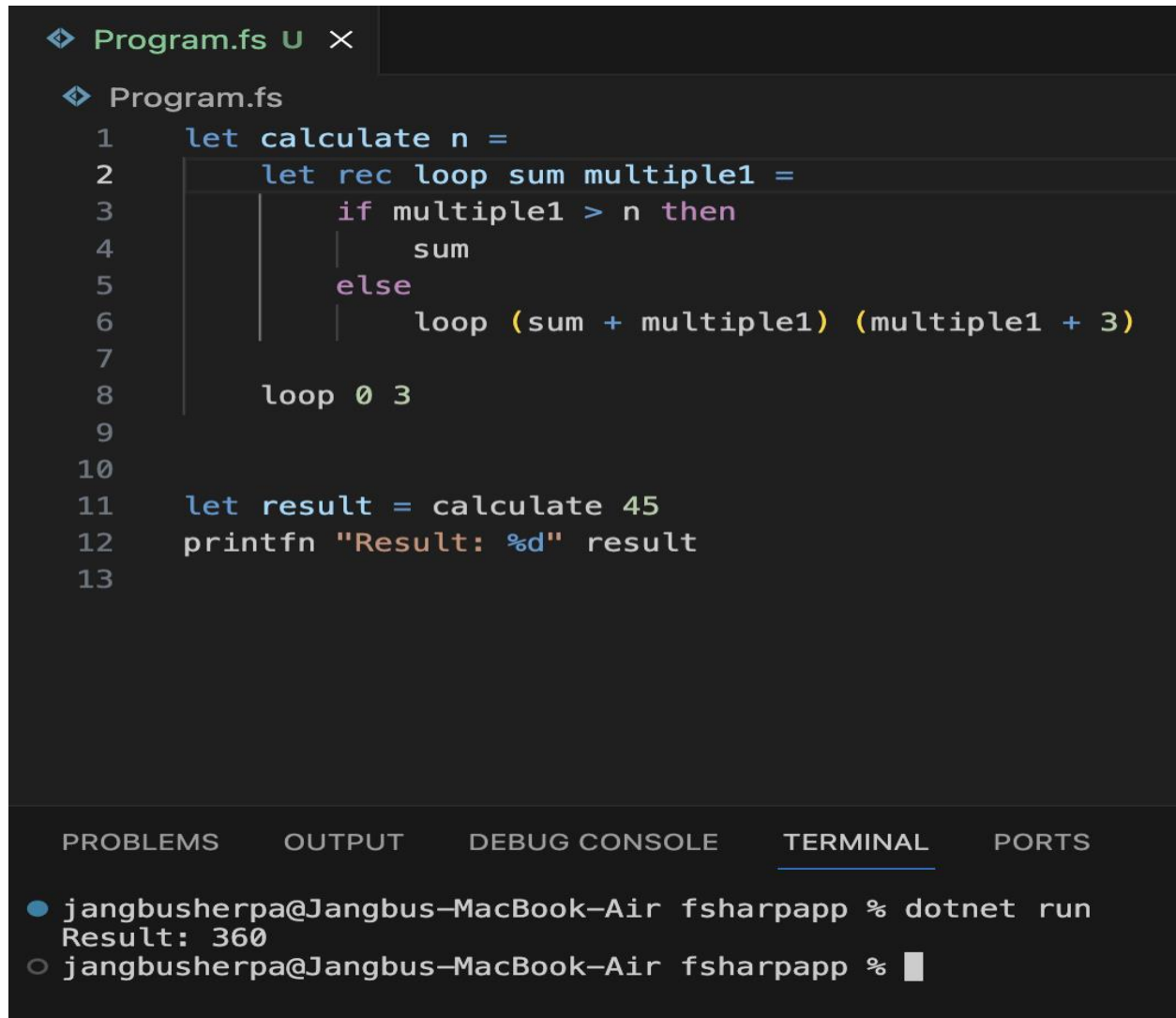
Below the editor, the `TERMINAL` tab is active, showing the output of the command `dotnet run`:

```
● jangbusherpa@Jangbus-MacBook-Air fsharpapp % dotnet run
Sum of salaries between $50,000 and $100,000: $167000
○ jangbusherpa@Jangbus-MacBook-Air fsharpapp %
```

## 2. Tail Recursion

Use tail recursion to write a program that will calculate the sum of all multiples of 3 up to a given number. Assume that we pass only a multiple of 3 as a parameter to this function. No validation is needed.

Example: If the parameter is 27, then the result of the function should be:  $3+6+9+12+15+18+21+24+27 = 135$ .



The screenshot shows a Visual Studio Code editor with a file named `Program.fs`. The code defines a recursive function `calculate` that sums multiples of 3 up to `n`. The function uses a `rec loop` construct with parameters `sum` and `multiple1`. The base case is when `multiple1 > n`, returning `sum`. The recursive case is `loop (sum + multiple1) (multiple1 + 3)`. The program calls `calculate 45` and prints the result. The terminal output shows the command `dotnet run` being executed, resulting in `Result: 360`.

```
Program.fs
1  let calculate n =
2      let rec loop sum multiple1 =
3          if multiple1 > n then
4              sum
5          else
6              loop (sum + multiple1) (multiple1 + 3)
7      loop 0 3
8
9
10
11  let result = calculate 45
12  printfn "Result: %d" result
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● jangbusherpa@Jangbus-MacBook-Air fsharpapp % dotnet run
Result: 360
○ jangbusherpa@Jangbus-MacBook-Air fsharpapp %
```

### 3. Uploading Repo to github

