

<b>Contents</b>	
<b>Part 1: Procedural Programming, Pascal</b>	<b>4</b>
<b>Pascal Definition</b>	<b>5</b>
<b>Describe Pascal grammar and compare it with C grammar</b>	<b>5,6,7</b>
<b>Program skeleton in Pascal</b>	<b>8</b>
<b>Pascal Control Statements</b>	<b>9,10</b>
<b>Area of the Circle Program</b>	<b>11</b>
<b>Prime Numbers Program</b>	<b>11</b>
<b>Vowel Character Program</b>	<b>12</b>
<b>The largest number out of 3 Program</b>	<b>12</b>
<b>Word Counter Program</b>	<b>13</b>
<b>Multiplication table Program</b>	<b>13</b>
<b>Squared Array Program</b>	<b>14</b>
<b>Calculate the tangent line Program</b>	<b>14</b>
<b>Month Name Program</b>	<b>15</b>
<b>Calculating the factorial number Program</b>	<b>15</b>
<b>Simple calculator Program</b>	<b>16</b>
<b>Temperature conversion program</b>	<b>17</b>
<b>Simple password validator Program</b>	<b>18</b>
<b>Binary to decimal converter Program</b>	<b>19</b>
<b>Simple math problem generator Program</b>	<b>20</b>
<b>Sorting Program</b>	<b>21</b>
<b>Countries Program</b>	<b>22</b>
<b>Even and odd numbers Program</b>	<b>23</b>
<b>Part 2: Functional programming, Scheme</b>	<b>24</b>
<b>Scheme Definition</b>	<b>25</b>
<b>Program skeleton in Scheme</b>	<b>25</b>
<b>Area of the Circle Program</b>	<b>26</b>
<b>Prime Numbers Program</b>	<b>26</b>
<b>Vowel Character Program</b>	<b>27</b>
<b>The largest number out of 3 Program</b>	<b>28</b>
<b>Word Counter Program</b>	<b>28</b>
<b>Multiplication table Program</b>	<b>29</b>
<b>Squared Array Program</b>	<b>30</b>
<b>Calculate the tangent line Program</b>	<b>31</b>
<b>Month Name Program</b>	<b>32</b>
<b>Calculating the factorial number Program</b>	<b>33</b>
<b>Simple calculator Program</b>	<b>34</b>
<b>Temperature conversion program</b>	<b>35</b>

<b>Simple password validator Program</b>	<b>36</b>
<b>Binary to decimal converter Program</b>	<b>37</b>
<b>Simple math problem generator Program</b>	<b>38</b>
<b>Sorting Program</b>	<b>39</b>
<b>Countries Program</b>	<b>40</b>
<b>Even and odd numbers Program</b>	<b>41</b>
<b>Part 3: Logic programming, Prolog</b>	<b>42</b>
<b>Prolog Definition</b>	<b>43</b>
<b>Program skeleton in Prolog</b>	<b>43</b>
<b>Area of the Circle Program</b>	<b>44</b>
<b>Prime Numbers Program</b>	<b>44</b>
<b>Vowel Character Program</b>	<b>44</b>
<b>The latest number out of 3 Program</b>	<b>45</b>
<b>Word Counter Program</b>	<b>45</b>
<b>Multiplication table Program</b>	<b>46</b>
<b>Squared Array Program</b>	<b>46</b>
<b>Calculate the tangent line Program</b>	<b>47</b>
<b>Month Name Program</b>	<b>48</b>
<b>Calculating the factorial number Program</b>	<b>49</b>
<b>Simple calculator Program</b>	<b>50</b>
<b>Temperature conversion program</b>	<b>51</b>
<b>Simple password validator Program</b>	<b>52</b>
<b>Binary to decimal converter Program</b>	<b>53</b>
<b>Simple math problem generator Program</b>	<b>54</b>
<b>Sorting Program</b>	<b>55</b>
<b>Countries Program</b>	<b>56</b>
<b>Even and odd numbers Program</b>	<b>57</b>
<b>Comparison (Pascal)</b>	<b>58</b>
<b>Comparison (Scheme)</b>	<b>59</b>
<b>Comparison (Prolog)</b>	<b>60</b>
<b>Final Comparison</b>	<b>61</b>
<b>Conclusion</b>	<b>62</b>

**Part 1:**  
**Procedural Programming, Pascal**

## **What is Pascal?**

Pascal is a general purpose high level language that was originally developed by Nicklaus Wirth in the early 1970s.

It was developed for teaching programming as a systematic discipline and to develop reliable and efficient programs, it is easy to understand and maintain the Pascal programs.

The computer programming languages C and Pascal have similar times of origin, influences, and purposes. Both were used to design and compile.

Both are descendants of the ALGOL language series.

ALGOL introduced

programming language support for structured programming, where programs are constructed of single entry and single exit constructs such as if, while, for and case.

## **Comparing Pascal grammar and C grammar:**

The computer programming languages C and Pascal have similar times of origin, influences, and purposes. Both were used to design and compile. Both are descendants of the ALGOL language series. ALGOL introduced programming language support for structured programming, where programs are constructed of single entry and single exit constructs such as if, while, for and case.

## **Pascal vs C**

### **Readability:**

Pascal: Generally more readable due to its strong typing and clear syntax, making it easier for beginners to understand.

C: More concise but can be less readable due to its flexibility and use of pointers, which may confuse new programmers.

## **Writability:**

Pascal: Encourages structured programming with a focus on clarity, which can lead to easier maintenance.

C: Offers more freedom and power for low-level programming, allowing for more efficient code, but can lead to complex and less maintainable code.

## **Reliability:**

Pascal: Strong typing and error-checking features enhance reliability and reduce runtime errors.

C: Offers less built-in error checking, making it easier to introduce bugs, but experienced programmers can manage this effectively.

## **Cost:**

Pascal: Generally lower cost for development in educational contexts due to its simplicity, but limited in terms of libraries and system-level programming.

C: Higher initial investment in terms of complexity but often more cost-effective for large-scale or system-level projects due to its performance and extensive libraries.

## **Pascal vs C in syntax:**

### **Comments:**

Pascal: { block comments } single- line comment

(\* block comments \*) multi- line comment

C: // single- line comment

/\* block comments \*/ multi- line comment

## **Declaration of variable:**

Pascal: var

variable\_list : type; x : integer;

C: type variable\_list;

int x;

## **Operators and Expression:**

Pascal: s := (a + b)

if (a and b)

C: s = a + b

if (a && b)

## **For loop:**

Pascal:

for < variable-name > := <

initial\_value > to [down to] <

final\_value > do

EX : for a := 10 to 50 do

C:

for ( initializationStatement ; condition; updateStatement )

{ // statements inside the body of loop }

EX : for (i = 1; i < 50; ++i) {}

## **Print:**

Pascal: writeln('.....');

C: printf("%d", .....);

## Program skeleton in Pascal:

```
1 program SkeletonProgram; { The name of the program }
2
3 { Variable declarations }
4 var
5     x: Integer;
6
7 { Function declarations }
8 Function DoSomething;
9 begin
10    { Placeholder for Function logic }
11 end;
12
13 { Main program block }
14 begin
15    { Initialization code }
16
17    { Call Functions }
18    DoSomething;
19
20    { Additional main program logic }
21
22    { End of program }
23 end.
24
```

## Pascal Control Statements:

**While loop:**

```
1 while (condition) do
2 begin
3 {statement to execute while
4 the condition is true }
5 end;
```

**For loop:**

```
1 for counter := initial_value to
2 final_value do
3 begin
4 { statement to execute for a
5 specified number of iterations }
6 end;|
```

**Repeat-until loop:**

```
1 {repeat-until loop syntax:}
2
3 program {Name};
4 begin
5   {initial value};
6   repeat
7     {statements}
8   until {condition};
9 end.
```

## If statement:

```
1 if condition then
2 begin
3   { statements to execute if condition is true }
4 end
5 else
6 begin
7   { statements to execute if condition is false }
8 end;
```

## Case statement:

```
case expression of
value1:{code block};
value2:{code block};
...
end;
```

## 1- Area of the circle

A program that calculates the area of a circle using user-provided radius and  $\pi$ , displaying the result formatted to two decimal places.

```
1  program CircleArea;
2
3  const
4      PI = 3.14159;
5
6  var
7      radius, area: real;
8
9  begin
10    write('Enter the radius of the circle: ');
11    readln(radius);
12
13    area := PI * radius * radius;
14
15    writeln('The area of the circle is: ', area:0:2);
16 end.
15 lines compiled, 0.0 sec
Enter the radius of the circle: 5
The area of the circle is: 78.54
```

## 2- Prime numbers

This program checks if a positive integer is prime by testing divisibility and displays the result.

```
1  program CheckPrime;
2
3  var
4      number, i: integer;
5      isPrime: boolean;
6
7  begin
8      write('Enter a number: ');
9      readln(number);
10
11     isPrime := (number > 1);
12
13     for i := 2 to trunc(sqrt(number)) do
14         if (isPrime and (number mod i = 0)) then
15             isPrime := false;
16
17     if isPrime then
18         writeln(number, ' is a prime number.')
19     else
20         writeln(number, ' is not a prime number.');
21
22 end.
23
```

Enter a number: 7  
7 is a prime number.

### 3- Vowel character

This program prompts the user to enter a character, checks if it is a vowel (a, e, i, o, u), and displays the result.

The screenshot shows the Delphi IDE interface. The top part contains the Pascal code for a vowel checker. The bottom part shows the output window with the message "input" and the program's response to the input "y".

```
1 program CheckVowel;
2
3 var
4   ch: char;
5   isVowel: boolean;
6
7 begin
8   write('Enter a character: ');
9   readln(ch);
10
11   ch := LowerCase(ch);
12
13   isVowel := (ch = 'a') or (ch = 'e') or (ch = 'i') or (ch = 'o') or (ch = 'u');
14
15   if isVowel then
16     writeln(ch, ' is a vowel.')
17   else
18     writeln(ch, ' is not a vowel.');
19 end.
```

input  
18 lines compiled, 0.0 sec  
Enter a character: y  
y is not a vowel.

### 4- The largest number out of 3

This program compares between 3 numbers entered by user then displays the largest number of them.

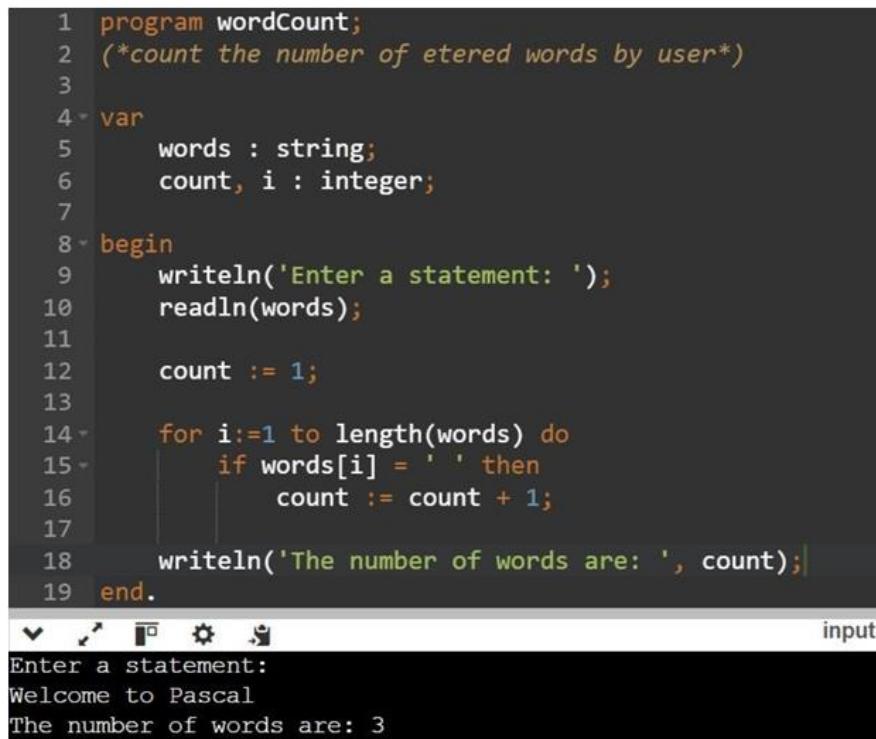
The screenshot shows the Delphi IDE interface. The top part contains the Pascal code for finding the largest of three numbers. The bottom part shows the output window with the message "input" and the program's response to the input "3", "2", and "4".

```
1 program largest_of_three;
2 (*to find the largest number out of three*)
3
4 var
5   a,b,c : integer;
6 begin
7   writeln('Enter 3 numbers to find the largest of them: ');
8   readln(a,b,c);
9
10  if ((a > b) and (a > c)) then
11    writeln('The largest is: ', a)
12  else if ((b > a) and (b > c)) then
13    writeln('The largest is: ', b)
14  else
15    writeln('The largest is: ', c);
16
17 end.
```

input  
Enter 3 numbers to find the largest of them:  
3  
2  
4  
The largest is: 4

## 5- Word counter

This program count words in a statement entered by user then displays the number of the words.

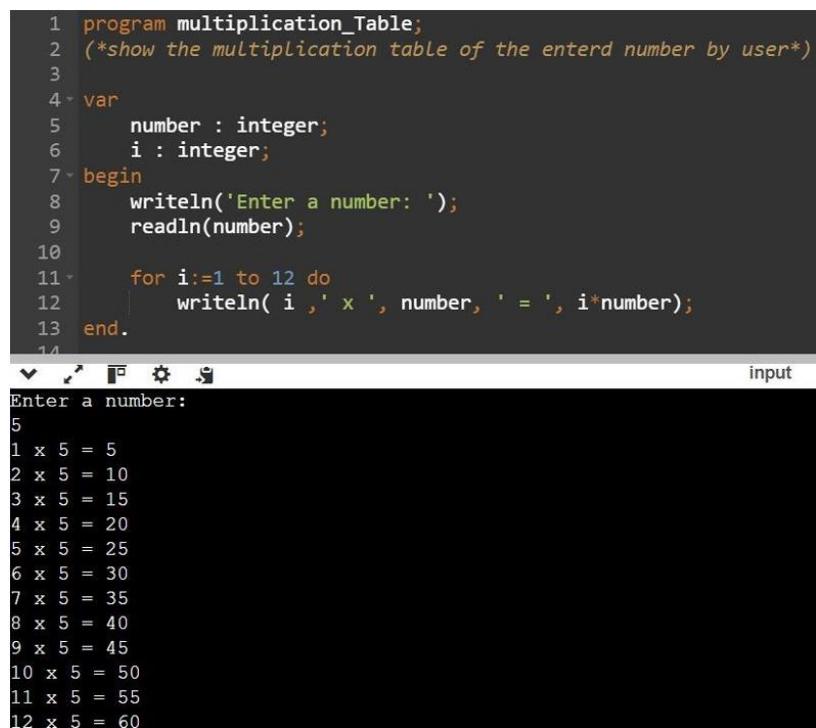


```
1 program wordCount;
2 (*count the number of etered words by user*)
3
4 var
5   words : string;
6   count, i : integer;
7
8 begin
9   writeln('Enter a statement: ');
10  readln(words);
11
12  count := 1;
13
14  for i:=1 to length(words) do
15    if words[i] = ' ' then
16      count := count + 1;
17
18  writeln('The number of words are: ', count);
19 end.
```

The screenshot shows the Pascal code for a word counter. The code uses a for loop to iterate through each character of the input string. If a space character is found, it increments a counter. Finally, it outputs the total number of words. The execution window shows the user entering the statement "Welcome to Pascal", and the program outputting "The number of words are: 3".

## 6- Multiplication table

This program displays the multiplication table of a number that entered by user.



```
1 program multiplication_Table;
2 (*show the multiplication table of the enterd number by user*)
3
4 var
5   number : integer;
6   i : integer;
7
8 begin
9   writeln('Enter a number: ');
10  readln(number);
11
12  for i:=1 to 12 do
13    writeln( i , ' x ', number, ' = ', i*number);
14 end.
```

The screenshot shows the Pascal code for a multiplication table generator. It prompts the user to enter a number, then prints its multiplication table from 1 to 12. For the number 5, the output is:

```
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45
10 x 5 = 50
11 x 5 = 55
12 x 5 = 60
```

## 7- Squared Array

This Pascal program efficiently squares each number in a list. It takes a list of numbers and squares each one, simplifying calculations where squared values are needed.

```
1  program SquareArray;
2  const
3  MAX_SIZE = 100;
4  type
5  IntArray = array[1..MAX_SIZE] of Integer;
6  var
7  arr: IntArray;
8  size, i: Integer;
9  begin
10 Write('Enter the size of the array: ');
11 Read(size);
12 Write('Enter ', size, ' elements: ');
13 for i := 1 to size do
14 Read(arr[i]);
15
16 arr[i] := arr[i] * arr[i];
17 Write('Squared array: ');
18 for i := 1 to size do
19 Write(' ', arr[i]);
20 end.
```

```
Enter the size of the array: 1
Enter 1 elements: 2
Squared array: 4
```

## 8- Calculate the tangent line

This Pascal program calculates the value of ( y ) on a tangent line given an initial point (x\_0, y\_0) and the slope (m), for a specified ( x ) value, using the formula for the equation of a line: ( y = y\_0 + m \* (x - x\_0) ).

```
1  program TL ;
2  var y,y0,x,x0,m : real ;
3  begin
4  Write ('Enter the value of x0 ') ;
5  ReadLn(x0);
6  Write ('Enter the value of y0 ') ;
7  ReadLn(y0);
8  Write ('Enter the slope of m ') ;
9  ReadLn(m);
10 Write('Enter the value of x for which you want to find y: ');
11 ReadLn(x);
12 y := y0 + m * ( x - x0 ) ;
13 WriteLn('The value of y at x = ', x:0:2, ' is: ', y:0:2);
14 end.
```

```
Enter the value of x0 2
Enter the value of y0 3
Enter the slope of m 2
Enter the value of x for which you want to find y: 4
The value of y at x = 4.00 is: 7.00
```

## 9- Month Name

This Pascal code is a simple program that takes a user-inputted month number (1-12) and outputs the corresponding month name.

```
1 program MonthName;
2 var
3 monthNumber: integer;
4 begin
5 writeln('Enter a month number (1-12): ');
6 readln(monthNumber);
7 case monthNumber of
8 1: writeln('January');
9 2: writeln('February');
10 3: writeln('March');
11 4: writeln('April');
12 5: writeln('May');
13 6: writeln('June');
14 7: writeln('July');
15 8: writeln('August');
16 9: writeln('September');
17 10: writeln('October');
18 11: writeln('November');
19 12: writeln('December');
20 else
21 writeln('Invalid month number');
22 end;
23 end.|
```

```
Enter a month number (1-12):
8
August
```

## 10- Calculating the factorial number

A program that calculates the product of all positive integers up to a given number. For example, the factorial of 5 is  $5 * 4 * 3 * 2 * 1 = 120$ .

```
1 program FactorailNum;
2
3 var num ,fact , i : integer;
4 begin
5 fact :=1;
6 write('Write integer number:');
7 readln(num);
8 if num < 0 then
9 writeln('Error no factorail for nagativ number.') else
10 begin
11 for i := 1 to num do
12 fact := fact * i;
13 writeln('The factorail number for ', num , 'is:', fact);
14 end;
15 end.
```

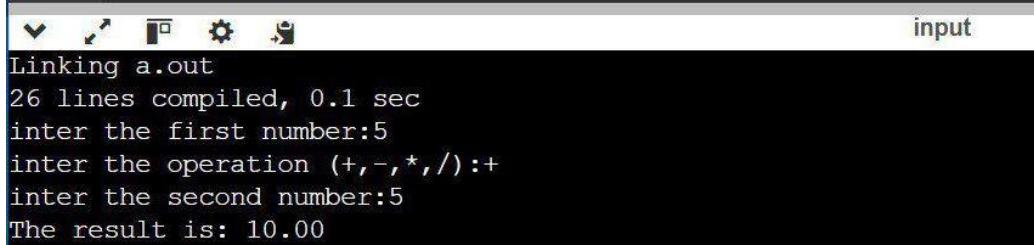
input

```
14 lines compiled, 0.0 sec
Write integer number:3
The factorail number for 3is:6
```

## 11- Simple calculator

A simple calculator program allows the user to perform basic arithmetic operations (addition, subtraction, multiplication, and division) between two numbers. The program first asks the user to enter two numbers, then, it asks the user to select an operation (+, -, \*, or /). Based on the selected operation, the program performs the appropriate calculation using conditional statements (if-else).

```
1 program SimpleCalc;
2 var
3 num1 , num2 , result: real;
4 operation: char;
5 begin
6 write('inter the first number:');
7 readln(num1);
8 write('inter the operation (+,-,*,/):');
9 readln(operation);
10 write('inter the second number:');
11 readln(num2);
12 if operation = '+' then
13 result := num1 + num2
14 else if operation = '-' then
15 result := num1 - num2
16 else if operation = '*' then
17 result := num1 * num2
18 else if (operation = '/') and (num2 <> 0) then
19 result := num1 / num2
20 else
21 begin
22 writeln('Invalid oprator or division by zero.');
23 exit;
24 end;
25 writeln('The result is: ', result:0:2);
26 end.
```



The screenshot shows the Delphi IDE interface. The top half displays the source code for a Pascal program named 'SimpleCalc'. The code prompts the user to input two numbers and a mathematical operator (+, -, \*, or /), then performs the corresponding calculation and outputs the result. The bottom half shows the execution window with the following text:  
Linking a.out  
26 lines compiled, 0.1 sec  
inter the first number:5  
inter the operation (+,-,\*,/):+  
inter the second number:5  
The result is: 10.00

## 12- Temperature conversion program

This program converts temperatures between Celsius and Fahrenheit. The user is prompted to select a conversion type: 1.Celsius to Fahrenheit 2.Fahrenheit to Celsius. Based on the choice, the program reads the input temperature and converts it using the appropriate formula: -Celsius to Fahrenheit:  $F = (C * 9/5) + 32$  -Fahrenheit to Celsius:  $C = (F - 32) * 5/9$  ,finally, the program prints the converted temperature.

```
1 program TemperatureConverting;
2 var
3   temperature , convtemp: real;
4   number: integer;
5
6   function CelToFah(cel: real):real;
7   begin
8     CelToFah := (cel * 9 / 5) + 32;
9   end;
10  function FahToCel(fah: real): real;
11  begin
12    FahToCel := (fah - 32) * 5 / 9;
13  end;
14  begin;
15    writeln('Select conversion type:');
16    writeln('1:from celsius to fahrenheit-');
17    writeln('2:from fahrenheit to celsius-');
18    write('inter your choise: 1 or 2:');
19    readln(number);
20    case number of 1:begin
21      write('inter Temperature in celsius:');
22      readln(temperature);
23      convtemp := CelToFah(temperature);
24      writeln('temperature in fahrenheit is:', convtemp:0:2);
25    end;
26    writeln('temperature in fahrenheit is:', convtemp:0:2);
27    end;
28    2: begin
29      write('inter Temperature in fahrenheit:');
30      readln(temperature);
31      convtemp := FahToCel(temperature);
32      writeln('temperature in celsius is:', convtemp:0:2);
33    end;
34    else
35      writeln('wrong choice');
36    end;
37  end.
```

input

```
Free Pascal Compiler version 3.2.2+dfsg-9ubuntul [2022/04/11] for x86_64
Copyright (c) 1993-2021 by Florian Klaempfl and others
Target OS: Linux for x86-64
Compiling main.pas
Linking a.out
35 lines compiled, 0.1 sec
Select conversion type:
1:from celsius to fahrenheit-
2:from fahrenheit to celsius-
inter your choise: 1 or 2: 1
inter Temperature in celsius: 0
temperature in fahrenheit is:32.00
```

## 13- Simple password validator

This program allows the user to enter a password which then the program checks if the password is valid according to the conditions. Conditions for a valid password are: Should have at least one number. Should have at least one uppercase letter. Should be more or equal to 8 characters long.

```
1  program SimplePasswordValidator;
2  var
3      password: string;
4      i: integer;
5      hasUpper, hasDigit: boolean;
6  begin
7      writeln('Simple Password Validator');
8      write('Enter your password: ');
9      readln(password);
10     hasUpper := false;
11     hasDigit := false;
12     if Length(password) < 8 then
13     begin
14         writeln('Password must be at least 8 characters long.');
15     end
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

```
else
begin
for i := 1 to Length(password) do
begin
    if password[i] in ['A'..'Z'] then
        hasUpper := true
    else if password[i] in ['0'..'9'] then
        hasDigit := true;
end;
if hasUpper and hasDigit then
    writeln('Password is valid.')
else
    writeln('Password must contain at least one uppercase letter and one digit.');
end;
```

```
29 lines compiled, 0.0 sec
Simple Password Validator
Enter your password: password
Password must contain at least one uppercase letter and one digit.
```

## 14- Binary to decimal converter

A program that converts a binary number into its decimal equivalent. It allows the user to enter a binary number, processes each bit from right to left, and calculates the corresponding decimal value by adding the powers of 2 for each '1' encountered. The final decimal result is then displayed to the user.

```
1 program BinaryToDecimalConverter;
2 var
3     binaryStr: string;
4     decimalValue, i, power: integer;
5 begin
6     writeln('Binary to Decimal Converter');
7     write('Enter a binary number: ');
8     readln(binaryStr);
9     decimalValue := 0;
10    power := 1;
11    for i := Length(binaryStr) downto 1 do
12    begin
13        if binaryStr[i] = '1' then
14            decimalValue := decimalValue + power;
15        power := power * 2;
16    end;
17    writeln('The decimal value is: ', decimalValue);
18 end.
```

```
25 lines compiled, 0.0 sec
Binary to Decimal Converter
Enter a binary number: 101
The decimal value is: 5
```

## 15- Simple math problem generator

A program that creates basic arithmetic problems by randomly selecting two integers (1 to 100) and one of four operations: addition, subtraction, multiplication, or division (with checks to avoid division by zero). Users solve the problem and receive instant feedback on their answer, including the correct solution if needed.

```
1  program SimpleMathProblemGenerator;
2  var
3      num1, num2, result: integer;
4      operator1: char;
5      userAnswer: integer;
6  begin
7      Randomize;
8      num1 := Random(100) + 1;
9      num2 := Random(100) + 1;
10     case Random(4) of
11         0: begin
12             operator1 := '+';
13             result := num1 + num2;
14         end;
15         1: begin
16             operator1 := '-';
17             result := num1 - num2;
18         end;
19         2: begin
20             operator1 := '*';
21             result := num1 * num2;
22         end;
23         3: begin
24             operator1 := '/';
25             num2 := Random(99) + 1;
26             result := num1 div num2;
27         end;
28     end;
29     writeln('Solve the following problem:');
30     writeln(num1, ' ', operator1, ' ', num2, ' = ?');
31     write('Your answer: ');
32     readln(userAnswer);
33
34     if userAnswer = result then
35         writeln('Correct!');
36     else
37         writeln('Incorrect. The correct answer is ', result);
38
39 end.
```

39 lines compiled, 0.1 sec  
Solve the following problem:  
90 + 25 = ?  
Your answer: 33  
Incorrect. The correct answer is 115

## 16- Sorting

This program implements the insertion sort algorithm to sort an array of integers. It defines an array of 5 elements, initially set to {10, 20, 30, 40, 50}. The sorting process moves through the array, shifting elements that are larger than the current key to the right, and placing the key in its correct position. The program prints the array before and after sorting.

```
program Sorting;

Var x: Integer;
Var arr : array[1..5] of Integer;

procedure Sort(size : Integer );
Var i, j, key : Integer;
Begin
  For i := 2 to size do
  Begin
    key := arr[i];
    j := i;
    While ((j > 1) AND (arr[j-1] > key)) do
    Begin
      arr[j] := arr[j-1];
      j := j - 1;
    End;
    arr[j] := key;
  End;
End;

Begin
  arr[1] := 10;
  arr[2] := 20;
  arr[3] := 30;
  arr[4] := 40;
  arr[5] := 50;

  writeln(' unsorted ');
  for x:= 1 to 5 do
  write(' ', arr[x] );

  Sort(5);
  writeln();
  writeln(' sorted ');

  for x:= 1 to 5 do
  write(' ', arr[x] );

End.
```

```
unsorted
10 20 30 40 50
sorted
10 20 30 40 50
```

## 17- Countries

This program prompts the user to select one of three countries (Egypt, Jordan, or Bahrain) and displays details about the chosen country, such as distance from Riyadh, flight duration, capital city, climate, and main airport. If the user enters an invalid number, the program shows an error message.

```
1 program CountriesInfo;
2 var
3     Number: integer;
4
5 function CheckCountries(num: integer): integer;
6 begin
7     if(num = 1) then
8         writeln('Egypt:');
9         +#13#10+'Distance from Riyadh - 1200 KM';
10        +#13#10+'Flight duration - 2 hours and 30 minutes';
11        +#13#10+'Capital city - Cairo';
12        +#13#10+'Climate - hot desert climate with mild winters';
13        +#13#10+'Airport - Cairo International Airport');
14    else if (num = 2) then
15        writeln('Jordan:');
16        +#13#10+'Distance from Riyadh - 900 KM';
17        +#13#10+'Flight duration - 1 hour and 45 minutes';
18        +#13#10+'Capital city - Amman';
19        +#13#10+'Climate - Mediterranean climate with hot summers';
20        +#13#10+'Airport - Queen Alia International Airport');
21    else if(num = 3) then
22        writeln('Bahrain:');
23        +#13#10+'Distance from Riyadh - 400 KM';
24        +#13#10+'Flight duration - 1 hour';
25        +#13#10+'Capital city - Manama';
26        +#13#10+'Climate - hot, humid summers and mild winters';
27        +#13#10+'Airport - Bahrain International Airport');
28    else
29        writeln('Wrong entry!');
30 end;
31
32 begin
33     writeln('Welcome!');
34     writeln('Choose one of these countries:');
35     writeln('1- Egypt');
36     writeln('2- Jordan');
37     writeln('3- Bahrain');
38     readln(Number);
39     CheckCountries(Number);
40 end.
```

```
Welcome!
Choose one of these countries:
1 - Egypt
2 - Jordan
3 - Bahrain
2
Jordan:
Distance from Riyadh - 1,315 kilometers
Flight duration - Approximately 2 hours
Capital city - Amman
Climate - Moderate climate with hot summers and cold winters
Airport - Queen Alia International Airport
```

## 18- Even and odd numbers

This program asks the user to enter six numbers, then displays them. It separates and prints the even and odd numbers based on a simple modulus check (mod 2). The program concludes with a thank-you message.

```
1 program Even_Odd;
2 var
3     i: integer;
4     num: array[1..6] of integer;
5 begin
6     writeln();
7     writeln('Hello, please enter any 6 numbers: ');
8     for i := 1 to 6 do
9     begin
10         readln(num[i]);
11     end;
12     writeln('Original numbers: ');
13     for i := 1 to 6 do
14     begin
15         write(num[i], ' ');
16     end;
17     writeln();
18     writeln('Even numbers: ');
19     for i := 1 to 6 do
20     begin
21         if (num[i] mod 2 = 0) then
22             write(num[i], ' ');
23     end;
24     writeln();
25     writeln('Odd numbers: ');
26     for i := 1 to 6 do
27     begin
28         if (num[i] mod 2 <> 0) then
29             write(num[i], ' ');
30     end;
31     writeln();
32     writeln('Thank you!');
33 end.
34
```

```
Hello, please enter any 6 numbers:
1
2
3
4
5
6
Original numbers:
1 2 3 4 5 6
Even numbers:
2 4 6
Odd numbers:
1 3 5
Thank you!
```

**Part 2:**  
**Functional programming, Scheme**

## What's Scheme?

Scheme is a functional programming language derived from Lisp. It is simple, minimalistic, and designed for teaching programming concepts, handling symbolic computations, and creating small to medium-sized applications. Scheme emphasizes recursion, higher-order functions, and clean, expressive syntax, making it popular in academic and research settings.

## Program skeleton in Scheme:

```
1 ; Define a variable
2 (define x 0)
3
4 ; Define a function
5 (define (do-something)
6     ; Placeholder for function logic
7 )
8
9 ; Main program block
10 (begin
11     ; Initialization code
12
13     ; Call functions
14     (do-something)
15
16     ; Additional main program logic
17 )
18
```

## 1-Area of the circle

This Scheme program calculates the area of a circle by prompting the user for the radius, computing the area using  $\pi \times r^2$ , and displaying the result rounded to two decimal places.

```
1 (define PI 3.14159) |
2
3 (define (circle-area)
4   (display "Enter the radius of the circle: ")
5   (let ((radius (read)))
6     (let ((area (* PI radius radius)))
7       (display "The area of the circle is: ")
8       (display (format "~,2f" area)))
9     (newline)))
10
11 (circle-area)
```

```
Enter the radius of the circle: 5
The area of the circle is: 78.54
```

## 2-Prime numbers

The program checks divisibility for numbers up to the square root and then outputs whether the number is prime.

```
1 (define (check-prime)
2   (display "Enter a number: ")
3   (let* ((number (read))
4         (isPrime (> number 1)))
5     (define (is-divisible n i)
6       (= (modulo n i) 0))
7
8     (define (check-prime-loop n i)
9       (if (<= i (sqrt n))
10           (if (is-divisible n i)
11               (set! isPrime #f)
12               (check-prime-loop n (+ i 1))))
13
14     (check-prime-loop number 2)
15
16     (if isPrime
17         (display (string-append (number->string number) " is a prime
18                               number.\n"))
19         (display (string-append (number->string number) " is not a
20                               prime number.\n")))))
21
22 (check-prime)
```

```
▼ Run
Enter a number: 20
20 is not a prime number.
```

### 3-Vowel character

This program prompts the user to enter a character, checks if it is a vowel (a, e, i, o, u), and displays the result.

```
1  (define (is-vowel? ch)
2    (or (char=? ch #\a)
3        (char=? ch #\e)
4        (char=? ch #\i)
5        (char=? ch #\o)
6        (char=? ch #\u)
7        (char=? ch #\A)
8        (char=? ch #\E)
9        (char=? ch #\I)
10       (char=? ch #\O)
11       (char=? ch #\U)))
12
13 (define (check-vowel)
14   (display "Enter a character: ")
15   (newline)
16   (let ((ch (read-char)))
17     (if (is-vowel? (char-downcase ch)) ;
18         (display (string-append (string ch) " is a vowel.\n"))
19         (display (string-append (string ch) " is not a vowel.\n")))))
20
21 (check-vowel)
```

```
Enter a character:
A
A is a vowel.
GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation, Inc.

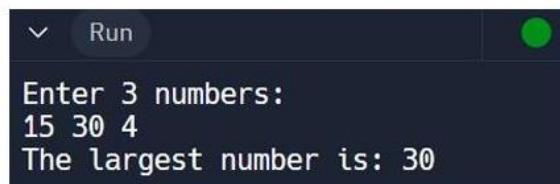
Guile comes with ABSOLUTELY NO WARRANTY; for details type `,show
w'.
This program is free software, and you are welcome to redistribute
it
under certain conditions; type `,show c' for details.

Enter `,help' for help.
> []
```

## 4-The largest number out of 3

This program asks the user to input three numbers then determines the largest among them using a conditional structure, after that displays the largest number to the user.

```
3 ; Largest of three numbers program
4
5 (define (largest-of-three a b c)
6   (cond
7     ((and (>= a b) (>= a c)) a)
8     ((and (>= b a) (>= b c)) b)
9     (else c)))
10
11 (display "Enter 3 numbers: ")
12 (newline)
13 (let ((a (read)) (b (read)) (c (read)))
14   (display "The largest number is: ")
15   (display (largest-of-three a b c))
16   (newline))
17 (newline)
```

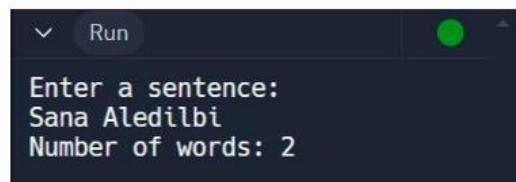


```
Run
Enter 3 numbers:
15 30 4
The largest number is: 30
```

## 5-Word counter

This program reads a sentence from the user and counts the number of words by checking for spaces between characters, then displays the total word count based on the input.

```
3 ; Word Counter program
4
5 (define (word-count str)
6   (define (count-words s count)
7     (if (null? (cdr s))
8         count
9         (count-words (cdr s) (if (char=? #\Space (car s)) (+ count 1) count))))
10  (count-words (string->list str) 1))
11
12 (display "Enter a sentence: ")
13 (newline)
14 (let ((input (read-line (current-input-port))))
15   (display "Number of words: ")
16   (display (word-count input))
17   (newline))
18 (newline)
```



```
Run
Enter a sentence:
Sana Aledilbi
Number of words: 2
```

## 6-Multiplication table

This program asks the user to enter a number to print the multiplication table for that number, from 1 to 12, then generates and displays each row of the table in the format "num \* i = result."

```
3 ; Multiplication Table program
4
5 (define (print-multiplication-table num)
6   (define (print-row i)
7     (when (<= i 12)
8       (display (string-append (number->string num) " x "
9                  (number->string i) " = " (number->string (* num i))))
10      (newline)
11      (print-row (+ i 1))))
12    (print-row 1))
13
14 (display "Enter a number: ")
15 (newline)
16 (let ((number (read)))
17   (print-multiplication-table number))
18 (newline)
```

The screenshot shows a terminal window with a dark background. At the top, there is a menu bar with a dropdown arrow icon and a 'Run' button. Below the menu bar, the text 'Enter a number:' is displayed. The user has typed '6' and pressed Enter. The terminal then outputs the multiplication table for 6, starting with '6 x 1 = 6' and continuing up to '6 x 12 = 72'. The output is as follows:

```
Enter a number:
6
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
6 x 11 = 66
6 x 12 = 72
```

## 7-Squared Array

This code prompts the user to enter the size of an array, reads the size, then prompts the user to enter elements of the array one by one. For each entered element, it calculates and displays its square.

The screenshot shows a code editor window with the following interface elements at the top:

- Execute
- Beautify
- Share
- Source Code
- Help

The code itself is numbered from 1 to 21. It defines a function `(main)` which performs three main steps:

- Asks the user for the size of the array.
- Reads the size entered by the user.
- Asks the user to enter the elements of the array separated by spaces.

For each element, it calculates and displays its square.

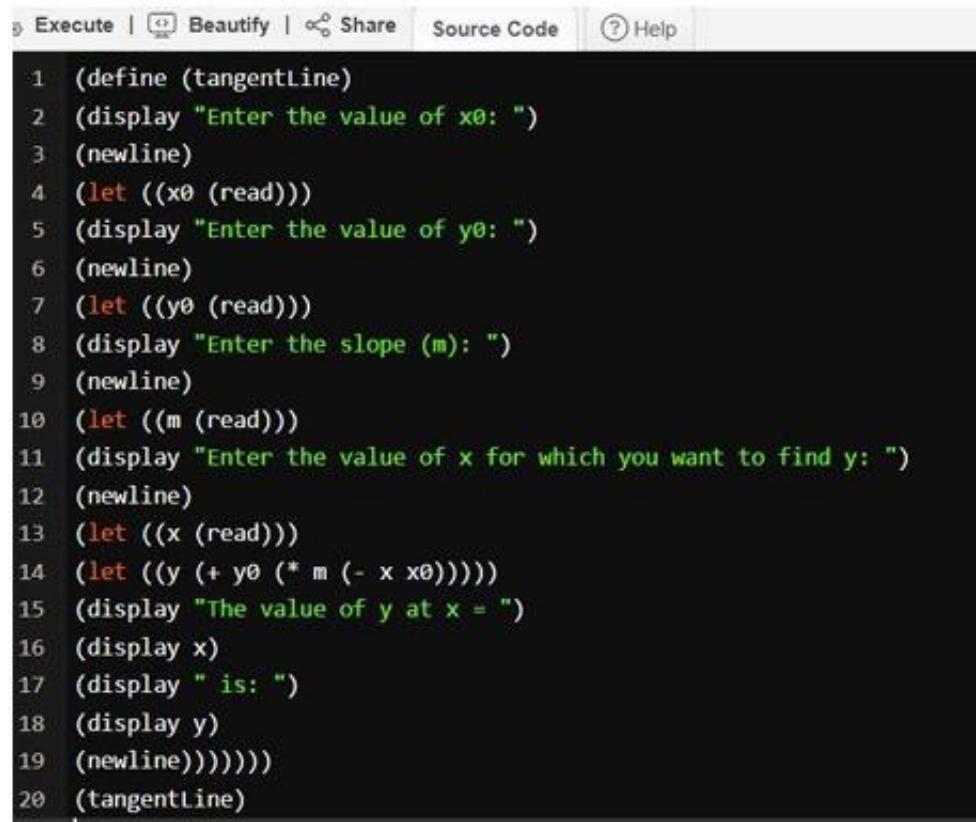
```
1 | (define (main)
2 ; Step 1: Ask the user to enter the size of the array
3 (display "Enter the size of the array: ")
4 ; Step 2: Read the size entered by the user
5 (let ((size (read)))
6 ; Move to the next line
7 (newline)
8 ; Step 3: Ask the user to enter the elements of the array
9 ; (display "Enter the elements of the array separated by spaces: ")
10 ; Read each element of the array
11 (let loop ((i 0))
12 (if (< i size)
13 (let ((element (read)))
14 ; Calculate and display the square of the element
15 (display "Squared element: ")
16 (display (* element element))
17 (newline)
18 (loop (+ i 1)))
19 ; No more elements to read
20 (newline))))))
21 (main)
```

The terminal window displays the following interaction:

```
1
Enter the size of the array:
2
Squared element: 4
```

## 8- Calculate the tangent line

This Scheme program calculates the value of ( y ) on a tangent line given an initial point (x\_0, y\_0) and the slope (m), for a specified ( x ) value, using the formula for the equation of a line: ( y = y\_0 + m \* (x - x\_0)

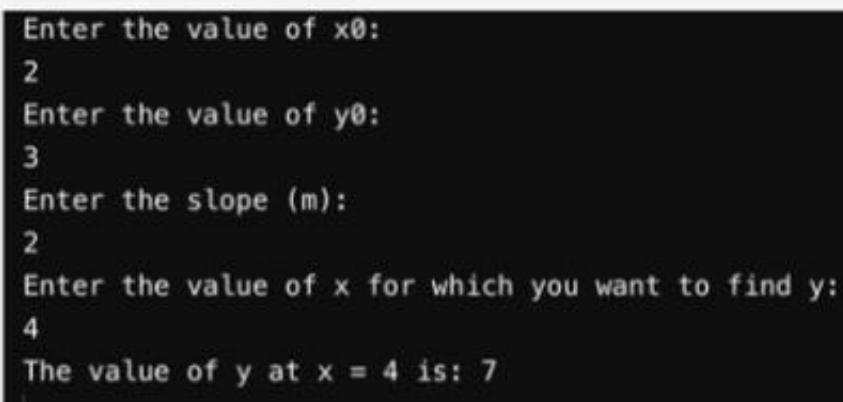


The screenshot shows a Scheme code editor window with the following interface elements at the top:

- Execute | Beautify | Share | Source Code |

The code itself is numbered from 1 to 20, representing the line numbers of the Scheme script:

```
1 (define (tangentLine)
2 (display "Enter the value of x0: ")
3 (newline)
4 (let ((x0 (read)))
5 (display "Enter the value of y0: ")
6 (newline)
7 (let ((y0 (read)))
8 (display "Enter the slope (m): ")
9 (newline)
10 (let ((m (read)))
11 (display "Enter the value of x for which you want to find y: ")
12 (newline)
13 (let ((x (read)))
14 (let ((y (+ y0 (* m (- x x0)))))
15 (display "The value of y at x = ")
16 (display x)
17 (display " is: ")
18 (display y)
19 (newline))))))
20 (tangentLine)
```

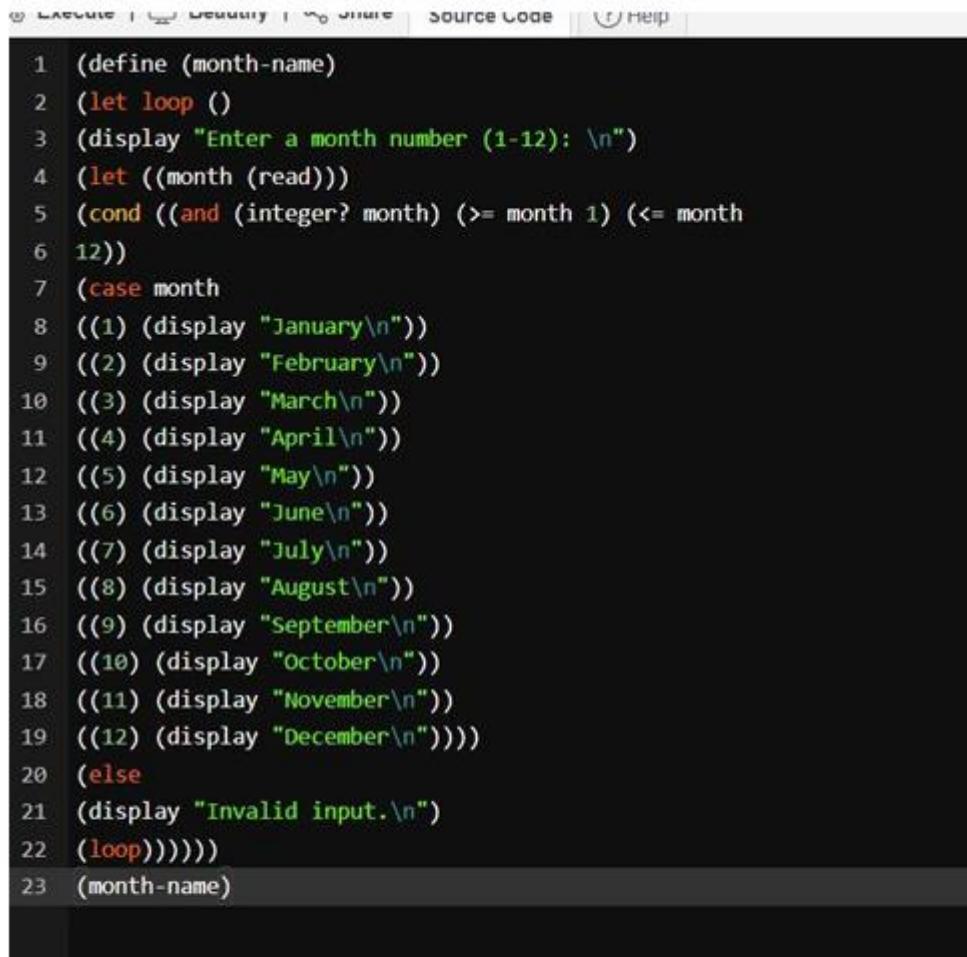


The terminal window displays the following interaction:

```
Enter the value of x0:
2
Enter the value of y0:
3
Enter the slope (m):
2
Enter the value of x for which you want to find y:
4
The value of y at x = 4 is: 7
```

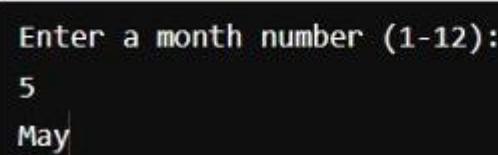
## 9-Month Name

This scheme code is a simple program named "MonthName" that takes a user inputted month number (1-12) and outputs the corresponding month name. with error handling for invalid inputs.



The screenshot shows a Scheme development environment with the following code:

```
1 (define (month-name)
2 (let loop ()
3 (display "Enter a month number (1-12): \n")
4 (let ((month (read)))
5 (cond ((and (integer? month) (>= month 1) (<= month
6 12))
7 (case month
8 ((1) (display "January\n"))
9 ((2) (display "February\n"))
10 ((3) (display "March\n"))
11 ((4) (display "April\n"))
12 ((5) (display "May\n"))
13 ((6) (display "June\n"))
14 ((7) (display "July\n"))
15 ((8) (display "August\n"))
16 ((9) (display "September\n"))
17 ((10) (display "October\n"))
18 ((11) (display "November\n"))
19 ((12) (display "December\n")))))
20 (else
21 (display "Invalid input.\n"))
22 (loop))))))
23 (month-name)
```

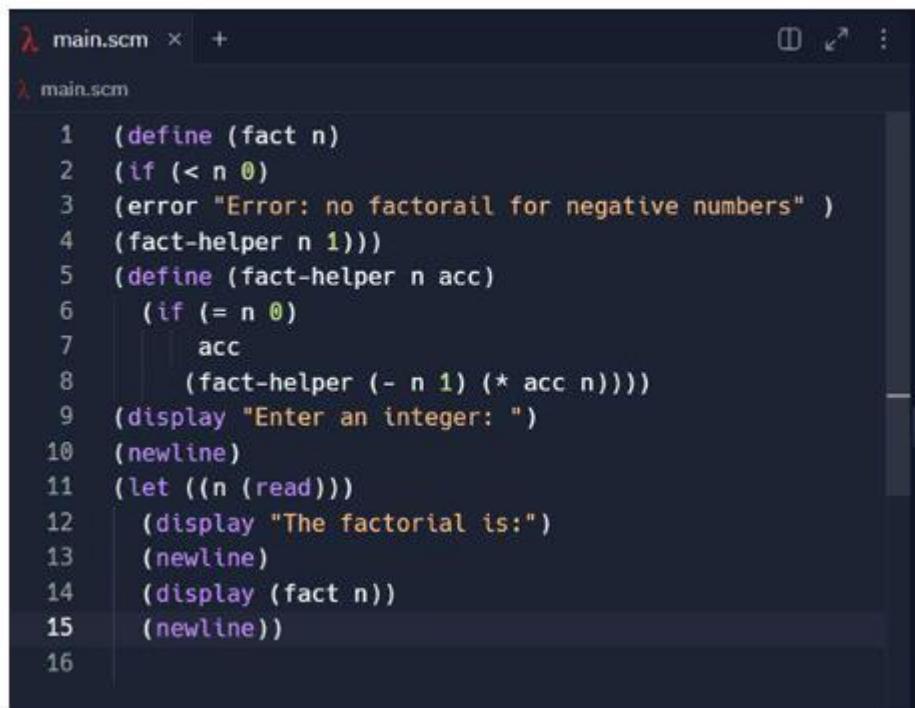


The terminal window displays the following interaction:

```
Enter a month number (1-12):
5
May
```

## 10- Calculating the factorial number

A program that calculates the product of all positive integers up to a given number. For example, the factorial of 5 is  $5 * 4 * 3 * 2 * 1 = 120$ .



```
λ main.scm × + : 
main.scm

1 (define (fact n)
2 (if (< n 0)
3   (error "Error: no factorial for negative numbers" )
4   (fact-helper n 1)))
5 (define (fact-helper n acc)
6   (if (= n 0)
7     acc
8     (fact-helper (- n 1) (* acc n))))
9 (display "Enter an integer: ")
10 (newline)
11 (let ((n (read)))
12   (display "The factorial is:")
13   (newline)
14   (display (fact n))
15   (newline))
16
```



```
AI Console × Shell + ⓘ ⌂ : 
Show Only Latest Clear History
Run Ask AI

Enter an integer:
5
The factorial is:
120
GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation,
Inc.

Guile comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This program is free software, and you are welcome
to redistribute it
under certain conditions; type `show c' for details.

Enter `help' for help.
> █
```

## 11- Simple calculator

A simple calculator program allows the user to perform basic arithmetic operations (addition, subtraction, multiplication, and division) between two numbers.

The screenshot shows a code editor with a file named `main.scm` containing Scheme code for a simple calculator. Below the editor is a terminal window showing the execution and output of the script.

```
λ main.scm × +  
λ main.scm  
1  (define (simpleCalc)  
2    (display "Enter the first number: ")  
3    (newline)  
4    (let ((num1 (read))  
5        (num2 0))  
6      (operation #\space)  
7        (result 0))  
8      (display "Enter the operation (+, -, *, /): ")  
9      (newline)  
10     (set! operation (read))  
11     (display "Enter the second number: ")  
12     (newline)  
13     (set! num2 (read))  
14     (if (equal? operation '+)  
15         (set! result (+ num1 num2))  
16     (if (equal? operation '-)  
17         (set! result (- num1 num2))  
18     (if (equal? operation '*)  
19         (set! result (* num1 num2))  
20     (if (and(equal? operation '/') (not (= num2 0)))  
21         (set! result (/ num1 num2))  
22         (begin (display "Error operator or division  
by zero")  
23             (newline)  
24             (exit))))))  
25     (display "The result is: ")  
26     (display result)  
27     (newline))  
28   (simpleCalc)
```

Console output:

```
Enter the first number:  
5  
Enter the operation (+, -, *, /):  
+  
Enter the second number:  
3  
The result is: 8  
GNU Guile 3.0.8  
Copyright (C) 1995-2021 Free Software Foundation, Inc.  
  
Guile comes with ABSOLUTELY NO WARRANTY; for details type `show w'.  
This program is free software, and you are welcome to redistribute it  
under certain conditions; type `show c' for details.  
  
Enter `,help' for help.  
> █
```

## 12- Temperature conversion program

This program converts temperatures between Celsius and Fahrenheit. The user is prompted to select a conversion type.

The image shows two screenshots of a development environment. The top screenshot is a code editor with a dark theme, displaying Scheme code in a file named 'main.scm'. The code defines functions for Celsius-to-Fahrenheit and Fahrenheit-to-Celsius conversions, and a main loop for user interaction. The bottom screenshot is a terminal window showing the execution of the program. It prompts the user for a choice (1 or 2), takes input for temperature, and displays the converted value. The terminal also shows the GNU Guile version and copyright information.

```
main.scm  +  main.scm
1  (define (CelToFah cel)
2    (+ (* cel (/ 9 5)) 32))
3  (define (FahToCel fah)
4    (* (- fah 32) (/ 5 9)))
5  (define (temp-conversion)
6    (display "Enter your choise: ")
7    (newline)
8    (display "1: from celsius to fahrenheit")
9    (newline)
10   (display "2: from fahrenheit to celsius")
11   (newline)
12   (let ((choise (read)) (temp 0) (converted-temp 0))
13     (case choise((1)
14       (display "Enter temperatur in celsius: ")
15       (newline)
16       (set! temp (read))
17       (set! converted-temp (CelToFah temp))
18       (display "Temprautur in fahrenheit:")
19       (display converted-temp))
20     ((2)
21       (display "Enter temperatur in fahrenheit: ")
22       (newline)
23       (set! temp (read))
24       (set! converted-temp (FahToCel temp))
25       (display "Temprautur in celsius:")
26       (display converted-temp))
27     (else (display "Invalid choise ")))
28     (newline)))
29   (temp-conversion))

AI  Console  Shell  +  Show Only Latest  Clear History
Run  Ask AI  Run
Enter your choise:
1: from celsius to fahrenheit
2: from fahrenheit to celsius
1
Enter temperatur in celsius:
8
Temprautur in fahrenheit:32
GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation,
Inc.

Guile comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This program is free software, and you are welcome
to redistribute it
under certain conditions; type `show c' for details.

Enter `,help' for help.
> []
```

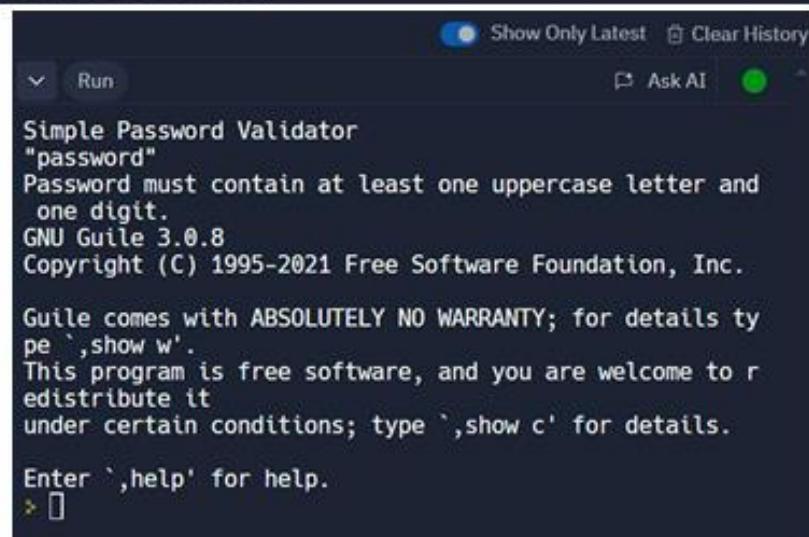
### 13- Simple password validator

This Scheme program is a simple password validator that checks if a password meets specific criteria. It prompts the user to enter a password and verifies that it is at least 8 characters long, contains at least one uppercase letter, and includes at least one digit. The program uses helper functions to identify uppercase letters and numeric characters. If the password meets all the requirements, it displays a message indicating that the password is valid; otherwise, it provides feedback on the specific criteria that were not met.

```
1  (define (simple-password-validator)
2    (display "Simple Password Validator\n")
3    (display "Enter your password: ")
4    (let ((password (read)))
5      (has-upper? #f)
6      (has-digit? #f))
7    (if (< (string-length password) 8)
8        (display "Password must be at least 8 characters long.\n")
9        (begin
10          (for-each
11            (lambda (ch)
12              (cond ((char-upper-case? ch) (set! has-upper? #t))
```

```
13                ((char-numeric? ch) (set! has-digit? #t))))
14              (string->list password))
15              (if (and has-upper? has-digit?))
16              (display "Password is valid.\n")
17              (display "Password must contain at least one uppercase letter and
one digit.\n")))))
18
19  (define (char-upper-case? ch)
20  (and (char>=? ch #\A) (char<=? ch #\Z)))
21
22  (define (char-numeric? ch)
23  (and (char>=? ch #\0) (char<=? ch #\9)))
24
25  (simple-password-validator)
```



The screenshot shows a terminal window with a dark background. At the top, there are two buttons: 'Show Only Latest' (selected) and 'Clear History'. Below that is a 'Run' button with a dropdown arrow. To the right of the run button are 'Ask AI' and a green circular icon. The main area of the terminal displays the output of the Scheme program:

```
Simple Password Validator
"password"
Password must contain at least one uppercase letter and
one digit.
GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation, Inc.

Guile comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This program is free software, and you are welcome to r
edistribute it
under certain conditions; type `show c' for details.

Enter `help' for help.
> []
```

## 14- Binary to decimal converter

This Scheme program converts a binary string to its decimal equivalent. It processes the binary digits from right to left, multiplying each digit by increasing powers of 2 and summing the results when the digit is 1. The program lets the user enter a binary string, then computes and displays the corresponding decimal value.

```
1 (define (binary-to-decimal binary-str)
2   (define (process-binary lst power decimal-value)
3     (if (null? lst)
4         decimal-value
5         (let ((current-digit (car lst)))
6           (process-binary
7             (cdr lst)
8             (* power 2)
9             (if (char=? current-digit #\1)
10                 (+ decimal-value power)
11                 decimal-value))))))
12
13 (let ((binary-list (string->list binary-str)))
14   (process-binary (reverse binary-list) 1 0)))
```

The screenshot shows a code editor window titled "main.scm". The code defines a function `(run-binary-to-decimal-converter)` which prompts the user for a binary string, calls `(binary-to-decimal)`, and displays the result. The code is as follows:

```
15
16 (define (run-binary-to-decimal-converter)
17   (display "Binary to Decimal Converter\n")
18   (display "Enter a binary number (as a string in quotes): ")
19   (let ((binary-str (read)))
20     (display "The decimal value is: ")
21     (display (binary-to-decimal binary-str))
22     (newline)))
23
24 (run-binary-to-decimal-converter)
```

The screenshot shows a terminal window with tabs for "AI", "Console", and "Shell". The "Console" tab is active, displaying the output of the Scheme program. The output shows the program running, prompting for input, and then displaying the decimal value of the entered binary string. The terminal also shows the GNU Guile version and copyright information, along with a standard disclaimer and help instructions.

```
Binary to Decimal Converter
"10110"
The decimal value is: 22
GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation, Inc.

Guile comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This program is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.

Enter `,help' for help.
> []
```

## 15- Simple math problem generator

This Scheme program generates a simple math problem using random numbers and basic operators. It first takes user input to seed the random number generator, then randomly selects two numbers and an operator (addition, subtraction, multiplication, or division). The user is prompted to solve the problem, and their answer is compared to the correct result.

The screenshot shows a Scheme development environment with two panes. The left pane displays the source code for `main.scm`, and the right pane shows the output of running the program in the console.

**Source Code (`main.scm`):**

```
1  (define (mod a b)
2    (- a (* b (quotient a b))))
3  (define (random-in-range min max)
4    (+ min (random (+ 1 (- max min)))))
5
6  (define (randomize-based-on-input)
7    (display "Enter any number to help randomize: ")
8    (let ((seed-input (read)))
9      (do ((i 0 (+ i 1)))
10        ((= i (mod seed-input 10)))
11        (random 100))))
12
13 (define (generate-math-problem)
14   (define num1 (random-in-range 1 100))
15   (define num2 (random-in-range 1 100))
16   (define operator (random 4))
17   (define result 0)
18
19   (cond
20     ((= operator 0)
21      (set! result (+ num1 num2))
22      (set! operator #\+))
23     ((= operator 1)
24      (set! result (- num1 num2))
25      (set! operator #\-))
26     ((= operator 2)
27      (set! result (* num1 num2))
28      (set! operator #\*))
29     ((= operator 3)
30      (set! num2 (random-in-range 1 99))
31      (set! result (quotient num1 num2))
32      (set! operator #\/)))
33
34   (display "Solve the following problem:\n")
35   (display num1)
36   (display " ")
37   (display operator))
```

**Console Output:**

```
Simple Math Problem Generator
54
Solve the following problem:
70 * 46 = ?
23
Incorrect. The correct answer is 3220
GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation, Inc.
Guile comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This program is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
Enter `help' for help.
>
```

## 16- Sorting

This program implements the insertion sort algorithm to sort an array of integers. It defines an array of 5 elements, initially set to {10, 20, 30, 40, 50}. The sorting process moves through the array, shifting elements that are larger than the current key to the right, and placing the key in its correct position. The program prints the array before and after sorting.

```
1  (define (arrange lis1 lis2)
2    (if (null? lis1)
3        lis2
4        (if (null? lis2)
5            lis1
6            (if (< (car lis1) (car lis2))
7                (cons (car lis1) (arrange (cdr lis1) lis2))
8                (cons (car lis2) (arrange (cdr lis1)
9                  lis2))))))
10 (define (sort x)
11   (if (null? x)
12     '()
13     (arrange (list (car x)) (sort (cdr x)))))
14
15 (display " unsorted ")
16 (newline)
17 (define Original '(40 50 30 10 20))
18 (display Original)
19 (newline)
20
21 (display " sorted ")
22 (newline)
23 (display (sort Original))
24 (newline)
```

```
unsorted
(40 50 30 10 20)
sorted
(10 20 30 40 50)
GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation, Inc.
```

## 17- Countries

This program prompts the user to select one of three countries (Egypt, Jordan, or Bahrain) and displays details about the chosen country, such as distance from Riyadh, flight duration, capital city, climate, and main airport. If the user enters an invalid number, the program shows an error message.

```
1  (define (CheckCountry value)
2    (cond
3      ((= value 1) (display "Egypt:
4 Distance from Riyadh - 1614 KM
5 Flight duration - 2 hours and 45 minutes
6 Capital city - Cairo
7 Climate - Hot desert climate
8 Airport - Cairo International Airport"))
9
10     ((= value 2) (display "Jordan:
11 Distance from Riyadh - 1242 KM
12 Flight duration - 2 hours
13 Capital city - Amman
14 Climate - Mediterranean climate with a hot dry summer
15 Airport - Queen Alia International Airport"))
16
17     ((= value 3) (display "Bahrain:
18 Distance from Riyadh - 420 KM
19 Flight duration - 1 hour
20 Capital city - Manama
21 Climate - Hot desert climate, humid summers
22 Airport - Bahrain International Airport"))
22 Airport - Bahrain International Airport"))
23
24     (else (display "Wrong entry!")))
25   )
26 )
27
28 (display "\nWelcome!\nChoose one of these countries:\n1-
29 Egypt\n2- Jordan\n3- Bahrain\n")
30 (display "-->")
31 (define num (read))
31 (CheckCountry num)

Welcome!
Choose one of these countries:
1 - Egypt
2 - Jordan
3 - Bahrain
2
Jordan:
Distance from Riyadh - 1,315 kilometers
Flight duration - Approximately 2 hours
Capital city - Amman
Climate - Moderate climate with hot summers and cold winters
Airport - Queen Alia International Airport
```

## 18- Even and odd numbers

This program asks the user to enter six numbers, then displays them. It separates and prints the even and odd numbers based on a simple modulus check ( $\text{mod } 2$ ). The program concludes with a thank-you message.

```
1  (define (CheckEven v1)
2    (cond
3      ((zero? v1) ""))
4      ((negative? v1) ""))
5      ((even? v1) (display v1) " ")
6      (else "")))
7  (define (CheckOdd v2)
8    (cond
9      ((zero? v2) ""))
10     ((negative? v2) ""))
11     ((odd? v2) (display v2) " ")
12     (else "")))
13 (display "Hello, please enter any 6 numbers:")
14 (newline)
15 (define value1 (read))
16 (define value2 (read))
17 (define value3 (read))
18 (define value4 (read))
19 (define value5 (read))
20 (define value6 (read))
21 (display "Original numbers:")
22 (newline)
23 (display value1)(display " ")
24 (display value2)(display " ")
25 (display value3)(display " ")
26 (display value4)(display " ")
27 (display value5)(display " ")
28 (display value6)
29 (newline)
30 (display "Even values are:")
31 (newline)
32 (display (CheckEven value1))
33 (display (CheckEven value2))
34 (display (CheckEven value3))
35 (display (CheckEven value4))
36 (display (CheckEven value5))
37 (display (CheckEven value6))
38 (newline)
```

```
Hello, please enter any 6 numbers:
1
2
3
4
5
6
Original numbers:
1 2 3 4 5 6
Even values are:
2 4 6
Odd values are:
1 3 5
Thank You!GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation, Inc.
```

**Part 3:**  
**Logic programming, Prolog**

## What is Prolog?

Prolog is a logical programming language commonly used in the field of artificial intelligence and natural language processing. In Prolog, relationships (facts and rules) are represented primarily.

## Program skeleton in Prolog:

```
1  % Facts
2  fact1.
3  fact2.
4
5  % Rules
6  rule1(X) :- condition1(X).
7  rule2(Y) :- condition2(Y).
8
9  % Main predicate
10 main :-
11     rule1(Result1), write(Result1), nl,
12     rule2(Result2), write(Result2), nl.
13
14 % Queries to run:
15 % ?- main.          % Runs the main program.
16 % ?- rule1(X).      % Queries rule1.
17 % ?- rule2(Y).      % Queries rule2.
18
```

## 1-Area of the circle

A program that calculates the area of a circle using user-provided radius and  $\pi$ , displaying the result formatted to two decimal places.

The screenshot shows the SWI-Prolog interface with a code editor containing the following Prolog code:

```
main.pl
1 area_of_circle(Radius, Area) :- 
2     Area is pi * Radius * Radius,
3     format('The area of the circle is: ~2f', [Area]).
4 
5 ?- write('Enter the radius of the circle: '), read(Radius),
6     area_of_circle(Radius, Area), nl, nl.
```

The right pane shows the output of running the program:

- Warning: /home/runner/DelectableUntimelyInterpreter/main.pl:5:
- Warning: Singleton variables: [Area]
- Enter the radius of the circle: 3.
- The area of the circle is: 28.27

## 2-Prime numbers

This program checks if a positive integer is prime by testing divisibility and displays the result.

The screenshot shows the SWI-Prolog interface with a code editor containing the following Prolog code:

```
main.pl
1 is_prime(2).
2 is_prime(3).
3 is_prime(P) :-
4     P > 3,
5     P mod 2 =\= 0,
6     P mod 3 =\= 0,
7     not_divisible_by_rest(P, 5).
8 
9 not_divisible_by_rest(P, D) :-
10    D * D > P.
11 not_divisible_by_rest(P, D) :-
12    P mod D =\= 0,
13    D1 is D + 2,
14    not_divisible_by_rest(P, D1).
15 
16 ?- write('Enter a number: '), read(N), (is_prime(N) -> format('~w is a prime number.', [N]), nl ; format('~w is not a prime number.', [N]), nl, nl).
```

The right pane shows the output of running the program:

- Enter a number: 7.
- 7 is a prime number.
- Welcome to SWI-Prolog (threaded, 64 bits, version 8.3.29)
- SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
- Please run ?- license. for legal details.
- For online help and background, visit <https://www.swi-prolog.org>
- For built-in help, use ?- help(Topic) . or ?- apropos(Word).
- ?- ■

## 3-Vowel character

This program prompts the user to enter a character, checks if it is a vowel (a, e, i, o, u), and displays the result.

The screenshot shows the SWI-Prolog interface with a code editor containing the following Prolog code:

```
main.pl
1 is_vowel(X) :-
2     member(X, ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']).
3 
4 ?- write('Enter a character: '), read(Char), (is_vowel(Char) ->
5     format('~w is a vowel.', [Char]), nl ; format('~w is not a vowel.', [Char]), nl, nl).
```

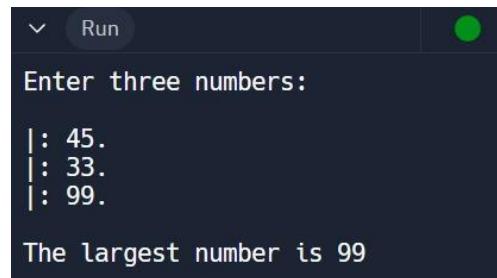
The right pane shows the output of running the program:

- Enter a character: y.
- y is not a vowel.
- Welcome to SWI-Prolog (threaded, 64 bits, version 8.3.29)
- SWI-Prolog comes with ABSOLUTELY NO WARRANTY.

## 4-The largest number out of 3

This program comparing between 3 numbers entered by user then displays the largest number of them.

```
1  %Find_the_Largest_of_Three_Numbers
2
3  largest_of_three(A, B, C, Largest) :- 
4      (A >= B, A >= C -> Largest = A ;
5      B >= A, B >= C -> Largest = B ;
6      Largest = C).
7
8  find_largest :-
9      writeln('Enter three numbers:'),nl,
10     read(X),
11     read(Y),
12     read(Z),nl,
13     largest_of_three(X, Y, Z, Largest),
14     write('The largest number is '), writeln(Largest),nl.
15
16 ?- find_largest.
```



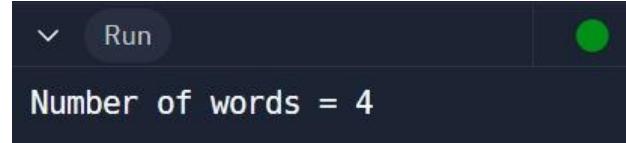
The screenshot shows a terminal window with the following interaction:

```
Run
Enter three numbers:
|: 45.
|: 33.
|: 99.
The largest number is 99
```

## 5-Word counter

This program count words in a statement entered by user then displays the number of the words.

```
1  %Word_Count_Program
2
3  word_count(String, Count) :-
4      split_string(String, " ", "", Words),
5      length(Words, Count).
6
7  ?- word_count("Hello world from Prolog", Count),
8      write("Number of words = "), write(Count), nl, nl.
```



The screenshot shows a terminal window with the following interaction:

```
Run
Number of words = 4
```

## 6-Multiplication table

This program displays the multiplication table of a number that entered by user.

```
1 %Multiplication_Table_Program
2
3 print_multiplication_table(_, 13) :- !.
4 print_multiplication_table(Number, I) :-
5     Format = '~w x ~w = ~w~n',
6     format(Format, [Number, I, Number * I]),
7     NextI is I + 1,
8     print_multiplication_table(Number, NextI).
9
10 ?- print_multiplication_table(5, 1),nl.
11
```

The screenshot shows the output of the Prolog program. It consists of a terminal window with a 'Run' button at the top. Below the button, there is a scroll bar. The terminal output lists the multiplication table of 5:

```
5 x 1 = 5*1
5 x 2 = 5*2
5 x 3 = 5*3
5 x 4 = 5*4
5 x 5 = 5*5
5 x 6 = 5*6
5 x 7 = 5*7
5 x 8 = 5*8
5 x 9 = 5*9
5 x 10 = 5*10
5 x 11 = 5*11
5 x 12 = 5*12
```

## 7-Squared Array

This program efficiently squares each number in a list. It takes a list of numbers and squares each one, simplifying calculations where squared values are needed.

```
1 % Predicate to read a list of integers and print their squares
2 main :- 
3 write('Enter the size of the array: '),
4 read_integer(Size),
5 calculate_squared_elements(Size, []).
6 % Predicate to read an integer from input
7 read_integer(X) :-
8 read_line_to_codes(user_input, Codes),
9 string_to_atom(Codes, Atom),
10 atom_number(Atom, X).
11 % Predicate to calculate the square of an element
12 calculate_squared_elements(0, SquaredList) :-
13 write('Squared elements: '), writeln(SquaredList).
14 calculate_squared_elements(Size, SquaredList) :-
15 Size > 0,
16 write('Enter element: '),
17 read_integer(Element),
18 Squared is Element * Element,
19 write('Squared element: '), writeln(Squared),
20 Size1 is Size - 1,
21 calculate_squared_elements(Size1, [Squared|SquaredList]).
22 :- main.
```

The screenshot shows the execution of the Prolog program. It consists of a terminal window with a toolbar at the top. The terminal output shows the user entering the size of the array (1) and individual elements (2), which are then squared and printed as a list:

```
Enter the size of the array: 1
Enter element: 2
Squared element: 4
Squared elements: [4]
```

## 8- Calculate the tangent line

This program calculates the value of ( y ) on a tangent line given an initial point (x\_0, y\_0) and the slope (m), for a specified ( x ) value, using the formula for the equation of a line: ( y = y\_0 + m \* (x - x\_0)).

```
1 tangent_line :-  
2 write('Enter the value of x0: '), read_line_to_codes(user_input, x0_input),  
3 atom_codes(x0_atom, x0_input),  
4 (x0_atom == stop ->  
5 halt;  
6 (atom_number(x0_atom, X0) ->  
7 read_y0(X0) ;  
8 writeln('Invalid input. Please enter a number or "stop".'), tangent_line  
9 ) ).  
10 read_y0(X0) :-  
11 write('Enter the value of y0: '), read_line_to_codes(user_input, y0_input),  
12 atom_codes(y0_atom, y0_input),  
13 (y0_atom == stop ->  
14 halt ;  
15 (atom_number(y0_atom, Y0) ->  
16 read_slope(X0, Y0);  
17 writeln('Invalid input. Please enter a number or "stop".'), read_y0(X0)  
18 ) ).  
19 read_slope(X0, Y0) :-  
20 write('Enter the slope (m) (or type "stop" to finish): '), read_line_to_codes(user_input,  
21 atom_codes(M_atom, M_input),  
22 (M_atom == stop ->  
23 halt ;  
24 (atom_number(M_atom, M) ->  
25 read_x(X0, Y0, M) ;  
26 writeln('Invalid input. Please enter a number or "stop".'), read_slope(X0, Y0)  
27 ) ).  
28 read_x(X0, Y0, M) :-  
29 write('Enter the value of x (or type "stop" to finish): '), read_line_to_codes(user_input  
30 atom_codes(Atom, Input),  
31 (Atom == stop ->  
32 halt ;  
33 (atom_number(Atom, X) ->  
34 read_slope(X0, Y0);  
35 writeln('Invalid input. Please enter a number or "stop".'), read_x(X0, Y0)  
36 ) ).  
37 read_slope(X0, Y0) :-  
38 write('Enter the slope (m) (or type "stop" to finish): '), read_line_to_codes(user_input,  
39 atom_codes(M_atom, M_input),  
40 (M_atom == stop ->  
41 halt ;  
42 (atom_number(M_atom, M) ->  
43 read_x(X0, Y0, M) ;  
44 writeln('Invalid input. Please enter a number or "stop".'), read_slope(X0, Y0)  
45 ) ).  
46 read_x(X0, Y0, M) :-  
47 write('Enter the value of x (or type "stop" to finish): '), read_line_to_codes(user_input  
48 atom_codes(Atom, Input),  
49 (Atom == stop ->  
50 halt ;  
51 (atom_number(Atom, X) ->  
52 calculate_y(X0, Y0, M, X, Y),  
53 format('The value of y at x = ~2f is: ~2f~n', [X, Y]),  
54 read_x(X0, Y0, M) ;  
55 writeln('Invalid input. Please enter a number or "stop".'), read_x(X0, Y0, M)  
56 ) ).  
57 calculate_y(X0, Y0, M, X, Y) :-  
58 Y is Y0 + M * (X - X0).  
59 main :-  
60 tangent_line.  
61 :- initialization(main, main).
```

Enter the value of x0: 2  
Enter the value of y0: 3  
Enter the slope (m) (or type "stop" to finish): 2  
Enter the value of x (or type "stop" to finish): 4  
The value of y at x = 4.00 is: 7.00  
Enter the value of x (or type "stop" to finish): stop

## 9-Month Name

This code is a simple program that takes a user-inputted month number (1-12) and outputs the corresponding month name.

```
1 month_name_loop :-  
2 writeln('Enter a month number : ' ),  
3 read_month_number(MonthNumber),  
4 ( MonthNumber == stop ->  
5 true  
6 ;  
7 ( month_name(MonthNumber, MonthName) ->  
8 format('Month name: ~w~n', [MonthName])  
9 ;  
10 writeln('Invalid month number')  
11 ),  
12 month_name_loop  
13 ).  
14 read_month_number(MonthNumber) :-  
15 read_line_to_codes(user_input, Input),  
16 atom_codes(Atom, Input),  
17 ( Atom == 'stop' ->  
18 MonthNumber = stop  
19 ;  
20 atom_number(Atom, MonthNumber)  
21 ).  
22 month_name(1, 'January').  
23 month_name(2, 'February').  
24 month_name(3, 'March').  
25 month_name(4, 'April').  
26 month_name(5, 'May').  
27 month_name(6, 'June').  
28 month_name(7, 'July').  
29 month_name(8, 'August').  
30 month_name(9, 'September').  
31 month_name(10, 'October').  
32 month_name(11, 'November').  
33 month_name(12, 'December').  
34 Example eighth :  
35  
36 Program to determine the month name from entered number:  
37  
38 | Page 38  
39  
40 main :-  
41 month_name_loop,  
42 halt.  
43 :- initialization(main, main).
```

```
Enter a month number :  
|: 8  
Month name: August  
Enter a month number :  
|: stop  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

## 10- Calculating the factorial number

A program that calculates the product of all positive integers up to a given number. For example, the factorial of 5 is  $5 * 4 * 3 * 2 * 1 = 120$ .

The screenshot shows a Prolog development environment. At the top, there are icons for an owl, a warning sign, and a plus sign, followed by the word "Program" and a close button. Below this is a code editor window containing the following Prolog code:

```
1 fact(0, 1).
2
3 fact(N, Result) :-
4     N > 0,
5     N1 is N - 1,
6     fact(N1, R1),
7     Result is N * R1.
```

Below the code editor is a query window with a gear icon and the query `fact(5 , Result).`. The result is displayed as `Result = 120`. Below the result are buttons for "Next", "10", "100", "1,000", and "Stop". At the bottom of the query window is another query line: `?- fact(5 , Result).`

## 11- Simple calculator

A simple calculator program allows the user to perform basic arithmetic operations (addition, subtraction, multiplication, and division) between two numbers.

```
Program X +  
1 simpleCalc :-  
2     write('Enter the first number: '), nl,  
3     read(Num1),  
4     write('Enter the operator (+, -, *, /): '), nl,  
5     read(Op),  
6     write('Enter the second number: '), nl,  
7     read(Num2),  
8     ( Op = '+'  
9         -> Result is Num1 + Num2  
10    ; Op = '-'  
11        -> Result is Num1 - Num2  
12    ; Op = '*'  
13        -> Result is Num1 * Num2  
14    ; Op = '/'  
15        -> ( Num2 =\= 0  
16            -> Result is Num1 / Num2  
17            ; write('Error: Division by zero.'), nl, fail  
18        )  
19    ; write('Error: Invalid operator.'), nl, fail  
20 ),  
21     write('The result is: '), write(Result), nl.
```

The screenshot shows a Prolog IDE interface. At the top, there's a toolbar with icons for file operations and a search bar. Below the toolbar, the title bar says "simpleCalc." with a gear icon. The main area has three tabs: "Examples▲", "History▲", and "Solutions▲". The "History▲" tab is active and displays the following session:

```
?- simpleCalc.  
true  
The result is: 15
```

At the bottom right of the main area, there are buttons for "table results" and "Run!".

## 12- Temperature conversion program

This program converts temperatures between Celsius and Fahrenheit. The user is prompted to select a conversion type:

1.Celsius to Fahrenheit 2.Fahrenheit to Celsius. Based on the choice, the program reads the input temperature and converts it using the appropriate formula: -Celsius to Fahrenheit:  $F = (C * 9/5) + 32$  - Fahrenheit to Celsius:  $C = (F - 32) * 5/9$  ,finally, the program prints the converted temperature.

The screenshot shows a Prolog IDE interface. At the top, there are icons for a file (document), a warning sign, 'Program', a close button, and a plus sign. Below this is the code for the temperature conversion program:

```
1 celToFah(Celsius, Fahrenheit) :-  
2     Fahrenheit is Celsius * 9 / 5 + 32.  
3 fahToCel(Fahrenheit, Celsius) :-  
4     Celsius is (Fahrenheit - 32) * 5 / 9.  
5 temperature_conversion :-  
6     write('Choose conversion type:\n'), nl,  
7     write('1: Celsius to Fahrenheit'), nl,  
8     write('2: Fahrenheit to Celsius'), nl,  
9     read(Choice),  
10    ( Choice ==:= 1  
11        -> write('Enter temperature in Celsius: '), nl,  
12            read(Celsius),  
13            celToFah(Celsius, Fahrenheit),  
14            write('Temperature in Fahrenheit: '), write(Fahrenheit), nl  
15    ; Choice ==:= 2  
16        -> write('Enter temperature in Fahrenheit: '), nl,  
17            read(Fahrenheit),  
18            fahToCel(Fahrenheit, Celsius),  
19            write('Temperature in Celsius: '), write(Celsius), nl ;  
20            write('Invalide choice'), nl, fail ).
```

The execution window below shows the following interaction:

```
temperature_conversion.  
Choose conversion type:  
1: Celsius to Fahrenheit  
2: Fahrenheit to Celsius  
1  
Enter temperature in Celsius:  
0  
Temperature in Fahrenheit: 32  
true  
?- temperature_conversion.
```

At the bottom of the interface, there are buttons for 'Examples', 'History', 'Solutions', 'table results' (unchecked), and a 'Run!' button.

## 13- Simple password validator

This program allows the user to enter a password which then the program checks if the password is valid according to the conditions. Conditions for a valid password are: Should have at least one number. Should have at least one uppercase letter. Should be more or equal to 8 characters long.



The screenshot shows the SWISH IDE interface. At the top is a menu bar with File, Edit, Examples, and Help. Below the menu is a toolbar with icons for file operations. The main area contains a code editor with the following Prolog code:

```
1 is_uppercase(Char) :- char_code(Char, Code), Code >= 65, Code <= 90.
2 is_digit(Char) :- char_code(Char, Code), Code >= 48, Code <= 57.
3
4 validate_password(Password) :-
5     string_length(Password, Length),
6     Length >= 8,
7     string_chars(Password, Chars),
8     member(Upper, Chars),
9     is_uppercase(Upper),
10    member(Digit, Chars),
11    is_digit(Digit),
12    writeln('Password is valid.').
13
14 validate_password(Password) :-
15     string_length(Password, Length),
16     Length < 8,
17     writeln('Password must be at least 8 characters long.').
18
19 validate_password(Password) :-
20     string_chars(Password, Chars),
21     \+ (member(Upper, Chars), is_uppercase(Upper)),
22     writeln('Password must contain at least one uppercase letter.').
23
24 validate_password(Password) :-
25     string_chars(Password, Chars),
26     \+ (member(Digit, Chars), is_digit(Digit)),
27     writeln('Password must contain at least one digit.').
28
29 simple_password_validator :-
30     writeln('Simple Password Validator'),
31     write('Enter your password: '),
32     read_line_to_string(user_input, Password),
33     validate_password(Password).
34
```

Below the code editor are two terminal windows. The first window shows the user entering a password and the program outputting an error message:

```
simple_password_validator.
Simple Password Validator
Enter your password:
Password
Password must contain at least one digit.
true
```

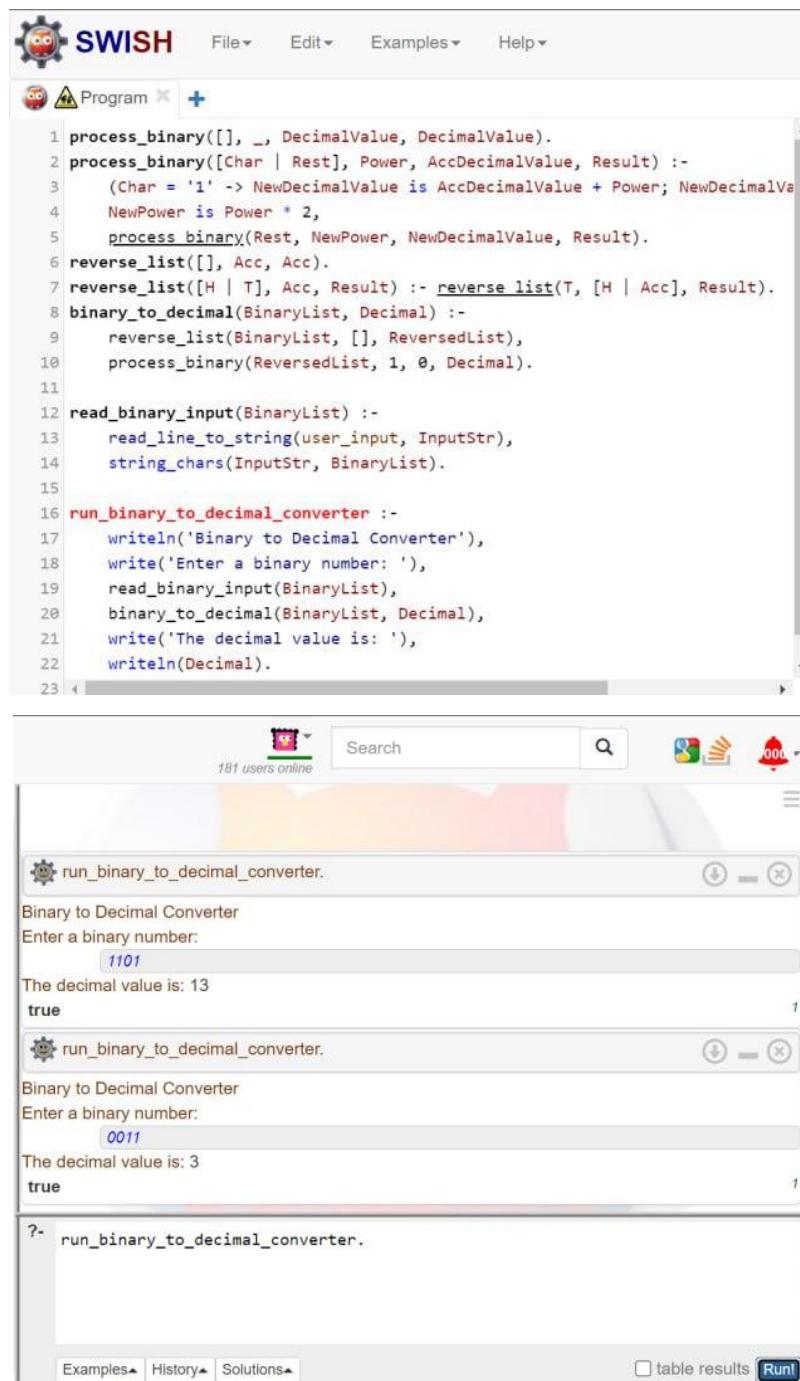
The second window shows the user entering a password and the program outputting a success message:

```
simple_password_validator.
Simple Password Validator
Enter your password:
Password1
Password is valid.
true
```

At the bottom of the interface, there are buttons for Next, 10, 100, 1,000, and Stop, along with a status bar showing the query ?- simple\_password\_validator.

## 14- Binary to decimal converter

A program that converts a binary number into its decimal equivalent. It allows the user to enter a binary number, processes each bit from right to left, and calculates the corresponding decimal value by adding the powers of 2 for each '1' encountered. The final decimal result is then displayed to the user.



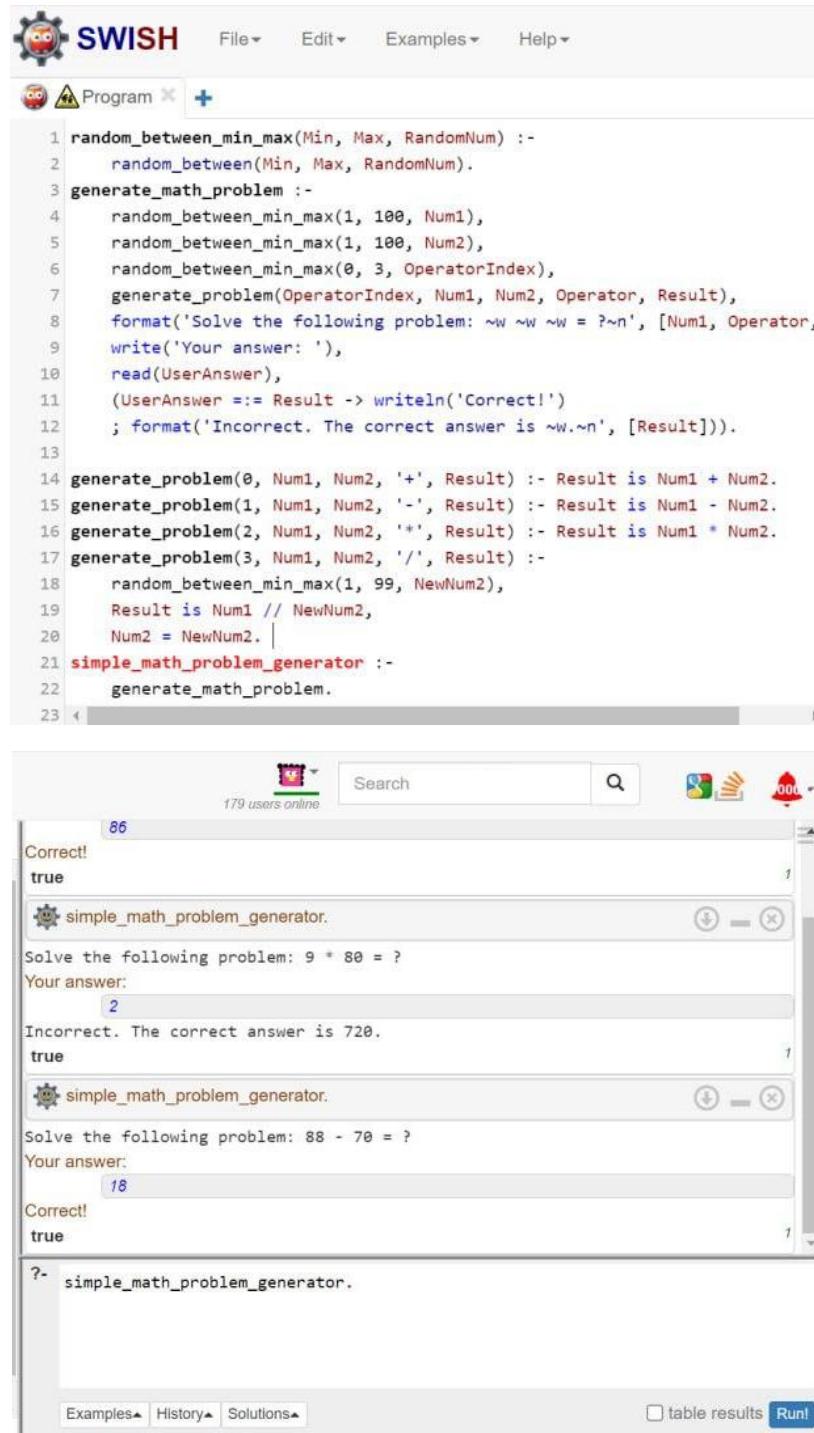
The screenshot shows the SWISH interface. At the top is a menu bar with File, Edit, Examples, and Help. Below it is a toolbar with icons for file operations. The main area has tabs for Program, Examples, and Solutions. The Program tab is selected, displaying the following Prolog code:

```
1 process_binary([], _, DecimalValue, DecimalValue).
2 process_binary([Char | Rest], Power, AccDecimalValue, Result) :-
3     (Char = '1' -> NewDecimalValue is AccDecimalValue + Power; NewDecimalValue is AccDecimalValue,
4      NewPower is Power * 2,
5      process_binary(Rest, NewPower, NewDecimalValue, Result)).
6 reverse_list([], Acc, Acc).
7 reverse_list([H | T], Acc, Result) :- reverse_list(T, [H | Acc], Result).
8 binary_to_decimal(BinaryList, Decimal) :-
9     reverse_list(BinaryList, [], ReversedList),
10    process_binary(ReversedList, 1, 0, Decimal).
11
12 read_binary_input(BinaryList) :-
13     read_line_to_string(user_input, InputStr),
14     string_chars(InputStr, BinaryList).
15
16 run_binary_to_decimal_converter :-
17     writeln('Binary to Decimal Converter'),
18     write('Enter a binary number: '),
19     read_binary_input(BinaryList),
20     binary_to_decimal(BinaryList, Decimal),
21     write('The decimal value is: '),
22     writeln(Decimal).
23
```

Below the code editor are two terminal windows. The top window is titled "run\_binary\_to\_decimal\_converter." and displays the output of the program. The bottom window is also titled "run\_binary\_to\_decimal\_converter." and shows the input prompt. At the bottom of the interface are buttons for Examples, History, Solutions, and a Run! button.

## 15- Simple math problem generator

A program that creates basic arithmetic problems by randomly selecting two integers (1 to 100) and one of four operations: addition, subtraction, multiplication, or division (with checks to avoid division by zero). Users solve the problem and receive instant feedback on their answer, including the correct solution if needed.



The screenshot shows the SWISH interface with the following details:

- Top Bar:** SWISH logo, Program tab, File, Edit, Examples, Help.
- Code Editor:** Displays the Prolog code for generating math problems. The code defines predicates like `random_between_min_max`, `generate_math_problem`, and `simple_math_problem_generator`. It generates random numbers between 1 and 100, selects an operator (+, -, \*, /), formats the problem, and checks the user's answer against the result.
- Execution Results:** Two separate windows show the execution of the generator. The first window shows a multiplication problem: "Solve the following problem: 9 \* 80 = ?" with the user answer "2". It is marked as incorrect with the message "Incorrect. The correct answer is 720." The second window shows a subtraction problem: "Solve the following problem: 88 - 70 = ?" with the user answer "18". It is marked as correct with the message "Correct!".
- Bottom Bar:** Examples, History, Solutions, Run button.

## 16- Sorting

This program implements the insertion sort algorithm to sort an array of integers. It defines an array of 5 elements, initially set to {10, 20, 30, 40, 50}. The sorting process moves through the array, shifting elements that are larger than the current key to the right, and placing the key in its correct position. The program prints the array before and after sorting.

```
% find the minimum number in a list
min([N], N).

min([H1, H2 | T], N) :- 
    H1 < H2,
    min([H1 | T], N).

min([H1, H2 | T], N) :- 
    H1 > H2,
    min([H2 | T], N).

% removing the minimum number from the list
remove_element(X, [X | T], T).
remove_element(X, [H | T], [H | Res]) :- 
    remove_element(X, T, Res).

% sorting the list
my_sort([], []).
my_sort(List, [X | Res]) :- 
    min(List, X),
    remove_element(X, List, List2),
    my_sort(List2, Res).
```

 `my_sort([20, 40, 30, 10, 50], Sorted).` ✖

**Sorted** = [10, 20, 30, 40, 50]

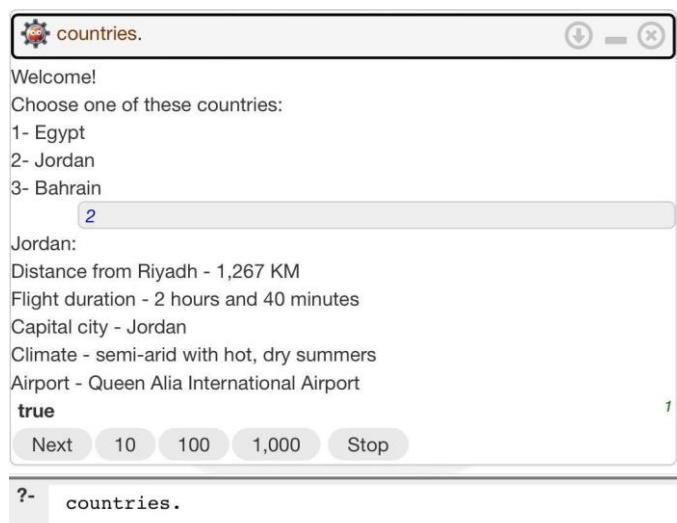
Next    10    100    1,000    Stop

?- `my_sort([20, 40, 30, 10, 50], Sorted).`

## 17- Countries

This program prompts the user to select one of three countries (Egypt, Jordan, or Bahrain) and displays details about the chosen country, such as distance from Riyadh, flight duration, capital city, climate, and main airport. If the user enters an invalid number, the program shows an error message.

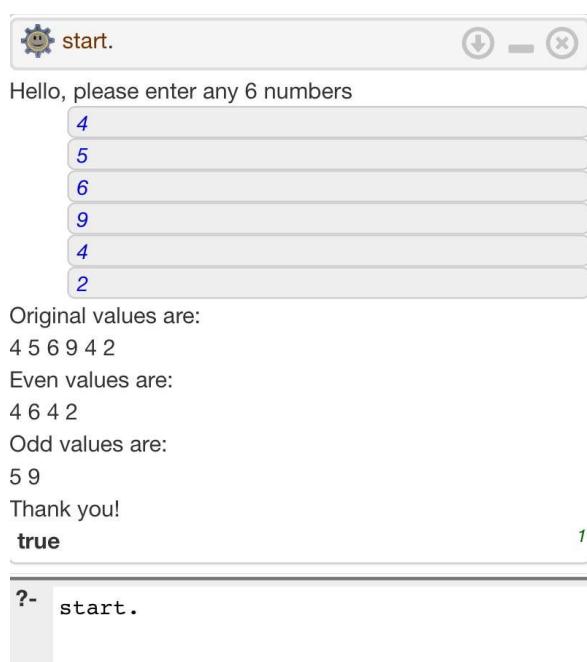
```
countries :-  
    write("Welcome!"), nl,  
    write("Choose one of these countries: "), nl,  
    write("1- Egypt"), nl,  
    write("2- Jordan"), nl,  
    write("3- Bahrain"), nl,  
    read(N),  
  
    ( N ==: 1 ->  
        write("Egypt:"), nl,  
        write("Distance from Riyadh - 1,214 KM"), nl,  
        write("Flight duration - 2 hours and 30 minutes"), nl,  
        write("Capital city - Cairo"), nl,  
        write("Climate - hot desert climate with mild winters"), nl,  
        write("Airport - Cairo International Airport");  
    N ==: 2 ->  
        write("Jordan:"), nl,  
        write("Distance from Riyadh - 1,267 KM"), nl,  
        write("Flight duration - 2 hours and 40 minutes"), nl,  
        write("Capital city - Amman"), nl,  
        write("Climate - semi-arid with hot, dry summers"), nl,  
        write("Airport - Queen Alia International Airport");  
    N ==: 3 ->  
        write("Bahrain:"), nl,  
        write("Distance from Riyadh - 467 KM"), nl,  
        write("Flight duration - 1 hour"), nl,  
        write("Capital city - Manama"), nl,  
        write("Climate - desert climate with very hot summers"), nl,  
        write("Airport - Bahrain International Airport");  
    write("Invalid choice, please enter 1, 2, or 3.")  
).
```



## 18- Even and odd numbers

This program asks the user to enter six numbers, then displays them. It separates and prints the even and odd numbers based on a simple modulus check ( $\text{mod } 2$ ). The program concludes with a thank-you message.

```
start :-  
    write("Hello, please enter any 6 numbers"), nl,  
    read(Value1),  
    read(Value2),  
    read(Value3),  
    read(Value4),  
    read(Value5),  
    read(Value6),  
  
    write("Original values are: "), nl,  
    write(Value1), write(" "), write(Value2), write(" "),  
    write(Value3), write(" "), write(Value4), write(" "),  
    write(Value5), write(" "), write(Value6), nl,  
  
    write("Even values are: "), nl,  
    even_checker(Value1), write(" "), even_checker(Value2), write(" "),  
    even_checker(Value3), write(" "), even_checker(Value4), write(" "),  
    even_checker(Value5), write(" "), even_checker(Value6), nl,  
  
    write("Odd values are: "), nl,  
    odd_checker(Value1), write(" "), odd_checker(Value2), write(" "),  
    odd_checker(Value3), write(" "), odd_checker(Value4), write(" "),  
    odd_checker(Value5), write(" "), odd_checker(Value6), nl,  
  
    write("Thank you!").  
  
even_checker(X) :-  
    X mod 2 == 0 -> write(X); write("").  
  
odd_checker(X) :-  
    X mod 2 == 1 -> write(X); write("") .
```



## **Comparison (Pascal):**

### **Readability:**

High for structured and simple programs due to clear syntax, but verbose for complex applications.

### **Writability:**

Easy for small, structured tasks but lacks flexibility for modern programming needs.

### **Reliability:**

Strongly reliable due to strict syntax and strong typing, though less so for large, complex systems.

### **Cost:**

Low-cost, with free compilers and a simple learning curve, but limited modern applicability may increase integration costs.

### **Portability:**

Highly portable, with broad support across platforms, though minor compiler differences exist.

### **Generality:**

Limited; best suited for education, algorithm development, and structured programming.

### **Well-Definedness:**

Highly well-defined, with consistent syntax, though variations like Turbo Pascal introduce some differences.

## **Comparison (Scheme):**

### **Readability:**

Scheme's syntax is uniform and simple, improving readability for those familiar with its structure, but challenging for beginners.

### **Writability:**

Highly writable for functional programming tasks; expressive, concise, and flexible for symbolic manipulation and recursion-heavy problems.

### **Reliability:**

Reliable when used in a functional paradigm, but lacks strict typing, which can lead to runtime errors.

### **Cost:**

Low development cost for functional or symbolic tasks, moderate runtime cost due to typically slower performance.

### **Portability:**

Portable in principle, but differences between implementations can limit portability for complex programs.

### **Generality:**

Highly general-purpose, supporting multiple paradigms like functional, procedural, and symbolic programming.

### **Well-Definedness:**

Clear and minimal specification, but inconsistencies can arise due to variations in implementations.

## **Comparison (Prolog):**

### **Readability:**

Prolog is readable within its domain, leveraging a clear declarative style, but it has a steep learning curve for beginners.

### **Writability:**

Highly writable for rule-based, logic-driven applications but limited for procedural or state-intensive tasks.

### **Reliability:**

Reliable for logic-based applications due to its declarative nature and absence of mutable state, though runtime errors related to unification or recursion are possible.

### **Cost:**

Prolog has a low development cost for logic-heavy applications, moderate maintenance cost due to clear logical structure, but potentially high runtime costs due to computationally expensive backtracking.

### **Portability:**

Moderately portable, but differences between implementations can limit consistency.

### **Generality:**

Specialized for logic programming, AI, and knowledge representation; less suited for general-purpose tasks.

### **Well-Definedness:**

Declarative syntax and semantics are clear, but implementation-specific features can cause inconsistencies.

## **Final Comparison:**

Pascal is the easiest language to learn, read, and write, especially for beginners, due to its clear and structured syntax designed for procedural programming. Scheme is more powerful and concise but harder to grasp initially because of its functional paradigm and heavy use of parentheses. Prolog, while ideal for solving logic-based problems like AI, is the most challenging to learn due to its declarative approach and reliance on concepts like backtracking. In summary, Pascal is best for beginners, Scheme for functional programming enthusiasts, and Prolog for advanced logical problem-solving.

## **Conclusion:**

In conclusion, understanding the procedural, functional, and logic programming types helps developers select the most suitable approach for their tasks. Procedural languages like Pascal are great for structured steps, functional languages like Scheme simplify complex computations, and logic languages like Prolog solve problems through reasoning. Choosing the right type ensures efficiency, low resource usage, and optimal solutions, allowing for versatile and maintainable software.