# UNIVERSITY IBN TOFAIL

*Algorithms II*

*Problem Set I*

## Exercise 1:

Write an algorithm that reads three integers $a$, $b$, and $c$ and determines the maximum and minimum values among these three numbers. Translate this algorithm into Python.

**Correction**

### Algorithm

**Require:** Three integers $a$, $b$, and $c$
**Ensure:** Maximum and minimum values
  Read integers $a$, $b$, and $c$
  Initialize max_val $= a$
  **if** $b >$ max_val **then**
    max_val $= b$
  **end if**
  **if** $c >$ max_val **then**
    max_val $= c$
  **end if**
  Initialize min_val $= a$
  **if** $b <$ min_val **then**
    min_val $= b$
  **end if**
  **if** $c <$ min_val **then**
    min_val $= c$
  **end if**
  Output max_val and min_val

### Python Implementation

```python
# Read three integers from input
a = int(input("Enter the first integer: "))
b = int(input("Enter the second integer: "))
c = int(input("Enter the third integer: "))

# Find maximum
max_val = a
if b > max_val:
    max_val = b
if c > max_val:
    max_val = c

# Find minimum
min_val = a
if b < min_val:
    min_val = b
if c < min_val:
    min_val = c

# Output results
print(f"Maximum: {max_val}")
print(f"Minimum: {min_val}")

```

Listing 1: Python Code for Exercise 1

# Exercise 2:

Write an algorithm and its Python implementation to calculate:

- The power $x^n$ where $x$ is a real number and $n$ is a positive integer.
- The factorial of a positive integer.
- The sum: $\sum_{i=1}^{n} i$

**Correction**

### 1. Power Calculation ($x^n$)

**Require:** A real number $x$ and a positive integer $n$
**Ensure:** Result of $x^n$
  Read $x$ and $n$
  Initialize result $= 1$
  **for** $i = 1$ to $n$ **do**
    result $=$ result $\times x$
  **end for**
  Output result

```
def power(x, n):
result = 1
for _ in range(n):
result *= x
return result

x = float(input("Enter base (x): "))
n = int(input("Enter exponent (n): "))
print(f"{x}^{n} = {power(x, n)}")
```

Listing 2: Power Calculation

### 2. Factorial Calculation

**Require:** A positive integer $n$
**Ensure:** Factorial of $n$
  Read $n$
  Initialize factorial $= 1$
  **for** $i = 1$ to $n$ **do**
    factorial $=$ factorial $\times i$
  **end for**
  Output factorial

```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

n = int(input("Enter a positive integer for factorial: "))
print(f"{n}! = {factorial(n)}")

```

Listing 3: Factorial Calculation

### 3. Sum of First $n$ Natural Numbers

**Correction**

**Require:** A positive integer $n$
**Ensure:** Sum $S = 1 + 2 + \cdots + n$

  Read $n$
  Initialize sum $= 0$
  **for** $i = 1$ to $n$ **do**
    sum $=$ sum $+ i$
  **end for**
  Output sum

```python
def sum_natural_numbers(n):
    return n * (n + 1) // 2  # Using formula for efficiency

n = int(input("Enter a positive integer for sum: "))
print(f"Sum 1+2+...+{n} = {sum_natural_numbers(n)}")

```

Listing 4: Sum of First n Natural Numbers

# Exercise 3:

Write algorithms and Python programs to calculate the value of the following expressions:

1. $A = (1+2) \times (1+2+3) \times (1+2+3+4) \times \ldots \times (1+2+3+\ldots+(N-2)+(N-1)+N)$, where $N \geq 2$
2. $B = 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + \ldots + \frac{1}{1+2+3+\ldots+N}$, where $N \geq 2$

**Correction**

### 1. Product $A$

**Require:** An integer $N \geq 2$
**Ensure:** Product $A$
  Read $N$
  Initialize product $= 1$
  **for** $k = 2$ to $N$ **do**
    sum_k $= \frac{k(k+1)}{2}$ {Triangular number formula}
    product $=$ product $\times$ sum_k
  **end for**
  Output product

```
def calculate_product_A(N):
    product = 1
    for k in range(2, N + 1):
        sum_k = k * (k + 1) // 2  # Triangular number
        product *= sum_k
    return product

N = int(input("Enter N (    2): "))
if N < 2:
    print("Error: N must be     2")
else:
    print(f"Product A = {calculate_product_A(N)}")

```

Listing 5: Product A Calculation

### 2. Sum $B$

**Require:** A positive integer $n$
**Ensure:** Sum $B$
  Read $n$
  Initialize sum_B $= 0$
  **for** $i = 1$ to $n$ **do**
    denominator $= \frac{i(i+1)}{2}$ {Triangular number}
    sum_B $=$ sum_B $+ \frac{1}{\text{denominator}}$
  **end for**
  Output sum_B

```
def calculate_sum_B(n):
    sum_B = 0.0
    for i in range(1, n + 1):
        denominator = i * (i + 1) // 2  # Triangular number
        sum_B += 1 / denominator
    return sum_B

n = int(input("Enter n (positive integer): "))
if n < 1:
    print("Error: n must be     1")
else:
    print(f"Sum B = {calculate_sum_B(n):.6f}")

```

Listing 6: Sum B Calculation

# Exercise 4:

Write an algorithm to determine all divisors of an integer $N$ entered by the user. Translate this algorithm into Python.

## Correction

### Algorithm

**Require:** An integer $N$
**Ensure:** List of all positive divisors of $|N|$ (excluding 0)
  Read $N$
  Initialize $N_{\text{abs}} = |N|$
  Create empty list divisors
  **for** $i = 1$ to $N_{\text{abs}}$ **do**
    **if** $N_{\text{abs}} \mod i = 0$ **then**
      Append $i$ to divisors
    **end if**
  **end for**
  Output divisors

### Python Implementation

```python
def find_divisors(n):
    n_abs = abs(n)
    divisors = []
    for i in range(1, n_abs + 1):
        if n_abs % i == 0:
            divisors.append(i)
    return divisors

try:
    N = int(input("Enter an integer: "))
    result = find_divisors(N)
    print(f"Divisors of {N}: {result}")
except ValueError:
    print("Invalid input. Please enter an integer.")

```

Listing 7: Divisor Calculation

# Exercise 5:

Write an algorithm that takes a positive integer as input and verifies whether this number is prime or not. Translate this algorithm into Python

**Correction**

**Algorithm**

**Require:** A positive integer $N$
**Ensure:** Boolean result: $N$ is prime or not

```
Read N
if N ≤ 1 then
    Output "Not prime"
    return
end if
if N = 2 then
    Output "Prime"
    return
end if
if N mod 2 = 0 then
    Output "Not prime"
    return
end if
Initialize is_prime = True
for i = 3 to √N step 2 do
    if N mod i = 0 then
        is_prime = False
    end if
end for
if is_prime then
    Output "Prime"
else
    Output "Not prime"
end if
```

**Python Implementation**

**Correction**

```python
import math

def is_prime(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, math.isqrt(n) + 1, 2):
        if n % i == 0:
            return False
    return True

try:
    N = int(input("Enter a positive integer: "))
    if is_prime(N):
        print(f"{N} is a prime number.")
    else:
        print(f"{N} is not a prime number.")
except ValueError:
    print("Invalid input. Please enter an integer.")

```

Listing 8: Prime Number Checker

# Exercise 6:

Write an algorithm to calculate the sum of all odd numbers between two values $N$ and $M$ $(1 \leq N < M)$. Translate this algorithm into Python.

## Correction

### Algorithm

**Require:** Two integers $N$ and $M$ such that $1 \leq N < M$
**Ensure:** Sum of all odd numbers in the range $[N, M]$
  Read $N$ and $M$
  **if** $N \geq M$ **then**
    Output "Error: N must be less than M"
    **return**
  **end if**
  Initialize sum_odds $= 0$
  **for** $i = N$ to $M$ **do**
    **if** $i \mod 2 = 1$ **then**
      sum_odds $=$ sum_odds $+ i$
    **end if**
  **end for**
  Output sum_odds

### Python Implementation

```python
def sum_odd_numbers(N, M):
    if N >= M:
        return "Error: N must be less than M"
    total = 0
    for num in range(N, M + 1):
        if num % 2 == 1:
            total += num
    return total

try:
    N = int(input("Enter N (1   N < M): "))
    M = int(input("Enter M (M > N): "))
    result = sum_odd_numbers(N, M)
    print(f"Sum of odd numbers between {N} and {M}: {result}")
except ValueError:
    print("Invalid input. Please enter integers.")

```

Listing 9: Sum of Odd Numbers Between N and M

## Exercise 7:

Write an algorithm that asks the user for the lengths of the sides of a triangle (lengths are positive integers) and determines whether this triangle is a right triangle or not. Translate this algorithm into Python.

**Correction**

### Algorithm

**Require:** Three positive integers $a$, $b$, and $c$
**Ensure:** Boolean result: Triangle is right-angled or not

Read $a$, $b$, $c$
**if** $a \leq 0$ OR $b \leq 0$ OR $c \leq 0$ **then**
  Output "Invalid input: All sides must be positive."
  **return**
**end if**
Sort $a$, $b$, $c$ such that $a \leq b \leq c$
**if** $a + b \leq c$ **then**
  Output "Invalid triangle: Triangle inequality violated."
  **return**
**end if**
**if** $a^2 + b^2 = c^2$ **then**
  Output "Right-angled triangle"
**else**
  Output "Not a right-angled triangle"
**end if**

### Python Implementation

```python
def is_right_triangle():
    try:
        a = int(input("Enter side a: "))
        b = int(input("Enter side b: "))
        c = int(input("Enter side c: "))
    except ValueError:
        print("Invalid input: All sides must be integers.")
        return

    if a <= 0 or b <= 0 or c <= 0:
        print("All sides must be positive integers.")
        return

    sides = sorted([a, b, c])
    a, b, c = sides

    if a + b <= c:
        print("Invalid triangle: Triangle inequality violated.")
        return

    if a**2 + b**2 == c**2:
        print("This is a right-angled triangle.")
    else:
        print("This is not a right-angled triangle.")

is_right_triangle()
```

Listing 10: Right Triangle Checker

# Exercise 8:

Write an algorithm to convert a binary number $N$ to decimal (The values 0 and 1 entered by the user are stored as characters '0' and '1', and the binary word $N$ is stored in a string). Example execution:

```
Binary to decimal conversion
Enter the number of bits: 8
Enter bit number 7: 1
Enter bit number 6: 0
Enter bit number 5: 1
Enter bit number 4: 1
Enter bit number 3: 1
Enter bit number 2: 0
Enter bit number 1: 1
Enter bit number 0: 0
N=10111010 Its decimal value is: 186
```

Write a Python program to convert a binary number $N$ to decimal.

## Correction

### Algorithm

**Require:** Number of bits $N$ and $N$ binary digits from MSB to LSB
**Ensure:** Decimal value of the binary number
  Read $N$
  Initialize decimal $= 0$
  **for** $i = 0$ to $N - 1$ **do**
    Read bit $b_i$ (either '0' or '1')
    decimal $=$ decimal $\times 2 + \text{int}(b_i)$
  **end for**
  Output decimal

### Python Implementation

```python
def binary_to_decimal():
    try:
        N = int(input("Enter the number of bits: "))
        if N <= 0:
            print("Number of bits must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    decimal = 0
    for i in range(N):
        while True:
            bit = input(f"Enter bit number {N - 1 - i}: ")
            if bit in ['0', '1']:
                decimal = decimal * 2 + int(bit)
                break
            else:
                print("Invalid input. Please enter '0' or '1'.")

    print(f"The decimal value is: {decimal}")

binary_to_decimal()
```

Listing 11: Binary to Decimal Conversion

# Exercise 9:

Write an algorithm for the input of an integer $n$, the calculation and display of the $n$-th term of the sequence $(U_n)$ defined as follows:

$$\begin{cases} U_0 = 2 \\ U_1 = 3 \\ U_{n+2} = \frac{2}{3}U_{n+1} - \frac{1}{4}U_n \end{cases}$$

Translate this algorithm into Python.

## Correction

### Algorithm

**Require:** A non-negative integer $n$
**Ensure:** The $n$-th term of the sequence $U_n$

Read $n$
**if** $n = 0$ **then**
  Output 2
**else if** $n = 1$ **then**
  Output 3
**else**
  Initialize $U_0 = 2$, $U_1 = 3$
  **for** $i = 2$ to $n$ **do**
    $U_2 = \frac{2}{3} \cdot U_1 - \frac{1}{4} \cdot U_0$
    Update $U_0 = U_1$, $U_1 = U_2$
  **end for**
  Output $U_2$
**end if**

### Python Implementation

```python
def compute_sequence_term(n):
    if n < 0:
        return "Error: n must be a non-negative integer."
    if n == 0:
        return 2.0
    if n == 1:
        return 3.0

    u0, u1 = 2.0, 3.0
    for _ in range(2, n + 1):
        u2 = (2/3) * u1 - (1/4) * u0
        u0, u1 = u1, u2
    return u2

try:
    n = int(input("Enter a non-negative integer n: "))
    result = compute_sequence_term(n)
    print(f"The {n}-th term of the sequence is: {result:.6f}")
except ValueError:
    print("Invalid input. Please enter an integer.")

```

Listing 12: Sequence Term Calculation

# Exercise 10:

The Syracuse sequence is defined according to a parity condition as follows:

$$u_0 \in \mathbb{N}, u_{n+1} = \begin{cases} \frac{1}{2}u_n & \text{if } u_n \text{ is even} \\ 3u_n + 1 & \text{if } u_n \text{ is odd} \end{cases}$$

The "Czech conjecture" states that for all initial values $u_0 \in \mathbb{N}^*$, there exists a rank $n$ for which $u_n = 1$. For example, if $u_0 = 6$ then $u_8 = 1$.

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $u_n$ | 6 | 3 | 10 | 5 | 16 | 8 | 4 | 2 | 1 | 4 | 2 |

Write an algorithm that asks the user to input the initial value $u_0$ and determines and displays the smallest value of $n$ for which $u_n = 1$. Translate this algorithm into Python.

## Correction

### Algorithm

**Require:** A positive integer $u_0$
**Ensure:** The smallest integer $n$ such that $u_n = 1$
  Read $u_0$
  Initialize $n = 0$
  Set current $= u_0$
  **while** current $\neq 1$ **do**
    **if** current $\mod 2 = 0$ **then**
      current $=$ current$/2$
    **else**
      current $= 3 \times$ current $+ 1$
    **end if**
    $n = n + 1$
  **end while**
  Output $n$

### Python Implementation

```python
def collatz_steps():
    try:
        u0 = int(input("Enter a positive integer (u0): "))
        if u0 <= 0:
            print("Error: u0 must be a positive integer.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    n = 0
    current = u0
    while current != 1:
        if current % 2 == 0:
            current = current // 2
        else:
            current = 3 * current + 1
        n += 1
    print(f"The smallest n for which u_n = 1 is: {n}")

collatz_steps()
```

Listing 13: Syracuse Sequence Checker

# Exercise 11:

To approximate the hypothetical limit of the sequence:

$$\begin{cases} U_0 = a \\ U_{n+1} = \frac{1+U_n}{1+2U_n} \end{cases}$$

Find an algorithm that calculates the value of the sequence $U_n$ such that $|U_n - U_{n-1}| < \varepsilon$ where $\varepsilon = 10^{-6}$, (where $a$ is a strictly positive real number entered from the keyboard). Translate this algorithm into Python.

## Correction

### Algorithm

**Require:** A strictly positive real number $a$, tolerance $\epsilon = 10^{-6}$
**Ensure:** The smallest $n$ such that $|U_n - U_{n-1}| < \epsilon$
  Read $a$
  **if** $a \leq 0$ **then**
    Output "Error: $a$ must be a positive real number."
    **return**
  **end if**
  Initialize $U_{\text{prev}} = a$, $n = 0$
  **repeat**
    Compute $U_{\text{curr}} = 1 + \frac{U_{\text{prev}}}{1+2\cdot U_{\text{prev}}}$
    $n = n + 1$
    Compute diff $= |U_{\text{curr}} - U_{\text{prev}}|$
    Update $U_{\text{prev}} = U_{\text{curr}}$
  **until** diff $< 10^{-6}$
  Output $U_{\text{curr}}$ and $n$

### Python Implementation

```python
def compute_sequence_term():
    try:
        a = float(input("Enter a strictly positive real number (a): "))
        if a <= 0:
            print("Error: a must be a positive real number.")
            return
    except ValueError:
        print("Invalid input. Please enter a valid real number.")
        return

    epsilon = 1e-6
    U_prev = a
    n = 0

    while True:
        U_curr = 1 + U_prev / (1 + 2 * U_prev)
        diff = abs(U_curr - U_prev)
        U_prev = U_curr
        n += 1
        if diff < epsilon:
            break

    print(f"Converged to U_n = {U_curr:.10f} after {n} iterations.")

compute_sequence_term()
```

Listing 14: Sequence Term Calculation with Stopping Condition

# Exercise 12:

Consider the following sequence $(X_n)n \in \mathbb{N}$:

$$\begin{cases} X_0 = A \\ X_{n+1} = \frac{1}{2}\left(X_n + \frac{A}{X_n}\right) \end{cases}$$

where $A$ is a strictly positive real number. Write an algorithm that asks the user to input the value of $A$ and then calculates and displays the value of the sequence $(X_n)$ (The stopping point for iterations is $|X_n - X_{n-1}| < 10^{-9}$). Implement this algorithm in Python. What does this algorithm calculate?

## Correction

### Algorithm

**Require:** A strictly positive real number $A$
**Ensure:** Approximation of $\sqrt{A}$ such that $|X_n - X_{n-1}| < 10^{-9}$
  Read $A$
  **if** $A \leq 0$ **then**
    Output "Error: $A$ must be strictly positive."
    **return**
  **end if**
  Initialize $X_{\text{prev}} = A$
  **repeat**
    Compute $X_{\text{curr}} = 0.5 \times \left(X_{\text{prev}} + \frac{A}{X_{\text{prev}}}\right)$
    Compute diff $= |X_{\text{curr}} - X_{\text{prev}}|$
    Update $X_{\text{prev}} = X_{\text{curr}}$
  **until** diff $< 10^{-9}$
  Output $X_{\text{curr}}$

### Python Implementation

```python
def newton_raphson_sqrt():
    try:
        A = float(input("Enter a strictly positive real number (A): "))
        if A <= 0:
            print("Error: A must be strictly positive.")
            return
    except ValueError:
        print("Invalid input. Please enter a valid real number.")
        return

    epsilon = 1e-9
    X_prev = A
    iterations = 0

    while True:
        X_curr = 0.5 * (X_prev + A / X_prev)
        diff = abs(X_curr - X_prev)
        X_prev = X_curr
        iterations += 1
        if diff < epsilon:
            break

    print(f"Approximated sqrt({A}) = {X_curr:.10f} after {iterations} iterations.")

newton_raphson_sqrt()
```

Listing 15: Newton-Raphson Square Root Approximation

**Explanation:** This algorithm computes the square root of $A$ using the **Newton-Raphson method**, an iterative technique for finding roots of equations. It converges quadratically to $\sqrt{A}$ when starting with a reasonable initial guess (here, $X_0 = A$). The stopping condition ensures high precision (within $10^{-9}$).

# Exercise 13:

Let $T$ be an array of integers of size $N$. Write an algorithm and the corresponding Python program that:

- Asks the user to input the elements of $T$.
- Displays the elements of $T$.
- Calculates and displays the sum of the elements of the array $T$.

## Correction

### Algorithm

**Require:** Size $N$ of array $T$ and $N$ integers as input
**Ensure:** Display array elements and their sum

```
Read N
if N ≤ 0 then
    Output "Error: Array size must be positive."
    return
end if
Initialize empty array T
for i = 0 to N − 1 do
    Read integer x
    Append x to T
end for
Output "Array: T"
Initialize total = 0
for x in T do
    total = total + x
end for
Output "Sum: total"
```

### Python Implementation

```python
def array_operations():
    try:
        N = int(input("Enter the size of the array (positive integer): "))
        if N <= 0:
            print("Error: Array size must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    T = []
    print(f"Enter {N} integers:")
    for i in range(N):
        while True:
            try:
                x = int(input(f"Element {i + 1}: "))
                T.append(x)
                break
            except ValueError:
                print("Invalid input. Please enter an integer.")

    print("Array:", T)
    total = sum(T)
    print("Sum of elements:", total)

array_operations()
```

Listing 16: Array Input

# Exercise 14:

Let $T$ be an array of integers of size $N$. Write an algorithm and the corresponding Python program that:

- Asks the user to input the elements of $T$.
- Displays the elements of $T$.
- Transfers the positive elements of $T$ to the array $TPOS$ and the strictly negative elements to the array $TNEG$.
- Displays the elements of $TPOS$ and $TNEG$.

**Correction**

**Algorithm**

**Require:** Size $N$ of array $T$ and $N$ integers as input
**Ensure:** Display $T$, $T_{\text{POS}}$, and $T_{\text{NEG}}$

```
Read N
if N ≤ 0 then
    Output "Error: Array size must be positive."
    return
end if
Initialize empty array T
for i = 0 to N − 1 do
    Read integer x
    Append x to T
end for
Output "Array: T"
Initialize empty arrays T_POS and T_NEG
for x in T do
    if x > 0 then
        Append x to T_POS
    else if x < 0 then
        Append x to T_NEG
    end if
end for
Output "Positive elements: T_POS"
Output "Negative elements: T_NEG"
```

**Python Implementation**

## Correction

```python
def array_separation():
    try:
        N = int(input("Enter the size of the array (positive integer): "))
        if N <= 0:
            print("Error: Array size must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    T = []
    print(f"Enter {N} integers:")
    for i in range(N):
        while True:
            try:
                x = int(input(f"Element {i + 1}: "))
                T.append(x)
                break
            except ValueError:
                print("Invalid input. Please enter an integer.")

    T_POS = []
    T_NEG = []
    for x in T:
        if x > 0:
            T_POS.append(x)
        elif x < 0:
            T_NEG.append(x)

    print("Original array:", T)
    print("Positive elements:", T_POS)
    print("Negative elements:", T_NEG)

array_separation()
```

Listing 17: Array Separation: Positive vs Negative

# Exercise 15:

Let $T$ be an array of real numbers of size $N$. Write an algorithm and the corresponding Python program that:

- Asks the user to input the elements of $T$.
- Calculates and displays the norm of the vector $T$ given by the following formula: $\text{norm} = \left(\sum_{i=1}^{N} T[i]^2\right)^{\frac{1}{2}}$

## Correction

### Algorithm

**Require:** Size $N$ of array $T$ and $N$ real numbers as input
**Ensure:** Norm of the vector $T$

Read $N$
**if** $N \leq 0$ **then**
  Output "Error: Array size must be positive."
  **return**
**end if**
Initialize empty array $T$
**for** $i = 0$ to $N - 1$ **do**
  Read real number $x$
  Append $x$ to $T$
**end for**
Initialize sum_squares $= 0$
**for** $x$ in $T$ **do**
  sum_squares $=$ sum_squares $+ x^2$
**end for**
Compute norm $= \sqrt{\text{sum\_squares}}$
Output norm

### Python Implementation

```python
import math

def vector_norm():
    try:
        N = int(input("Enter the size of the array (positive integer): "))
        if N <= 0:
            print("Error: Array size must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    T = []
    print(f"Enter {N} real numbers:")
    for i in range(N):
        while True:
            try:
                x = float(input(f"Element {i + 1}: "))
                T.append(x)
                break
            except ValueError:
                print("Invalid input. Please enter a valid real number.")

    sum_squares = sum(x**2 for x in T)
    norm = math.sqrt(sum_squares)
    print(f"The norm of the vector is: {norm:.6f}")

vector_norm()
```

Listing 18: Vector Norm Calculation

# Exercise 16:

Let $v_1$ and $v_2$ be two arrays of real numbers of the same size $N$. Write an algorithm and the corresponding Python program that:

- Asks the user to input the elements of $v_1$ and $v_2$.
- Calculates and displays the distance $d$ between the two vectors given by:

$$d = \frac{1}{N^2} \sum_{i=1}^{N} (v_1[i] - v_2[i])^2$$

- Calculates and displays the dot product of the two vectors $v1$ and $v2$.

**Correction**

**Algorithm**

**Require:** Size $N$ of vectors $v_1$ and $v_2$, and $N$ real numbers for each vector
**Ensure:** Distance $d$ and dot product $p$ between the vectors
  Read $N$
  **if** $N \leq 0$ **then**
    Output "Error: Vector size must be positive."
    **return**
  **end if**
  Initialize empty arrays $v_1$ and $v_2$
  **for** $i = 0$ to $N - 1$ **do**
    Read real number $x$ for $v_1[i]$
    Append $x$ to $v_1$
  **end for**
  **for** $i = 0$ to $N - 1$ **do**
    Read real number $y$ for $v_2[i]$
    Append $y$ to $v_2$
  **end for**
  Initialize $\text{sum\_diff} = 0$ and $\text{sum\_dot} = 0$
  **for** $i = 0$ to $N - 1$ **do**
    $\text{diff} = v_1[i] - v_2[i]$
    $\text{sum\_diff} = \text{sum\_diff} + \text{diff} \times \text{diff}$
    $\text{sum\_dot} = \text{sum\_dot} + v_1[i] \times v_2[i]$
  **end for**
  Compute $d = \frac{\text{sum\_diff}}{N^2}$
  Compute $p = \text{sum\_dot}$
  Output $d$ and $p$

**Python Implementation**

**Correction**

```python
def vector_operations():
    try:
        N = int(input("Enter the size of the vectors (positive integer): "))
        if N <= 0:
            print("Error: Vector size must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    v1 = []
    v2 = []
    print(f"Enter {N} real numbers for vector v1:")
    for i in range(N):
        while True:
            try:
                x = float(input(f"Element {i + 1}: "))
                v1.append(x)
                break
            except ValueError:
                print("Invalid input. Please enter a valid real number.")

    print(f"Enter {N} real numbers for vector v2:")
    for i in range(N):
        while True:
            try:
                y = float(input(f"Element {i + 1}: "))
                v2.append(y)
                break
            except ValueError:
                print("Invalid input. Please enter a valid real number.")

    sum_diff = 0.0
    sum_dot = 0.0
    for i in range(N):
        diff = v1[i] - v2[i]
        sum_diff += diff * diff
        sum_dot += v1[i] * v2[i]

    distance = sum_diff / (N * N)
    dot_product = sum_dot

    print(f"Distance between vectors: {distance:.6f}")
    print(f"Dot product of vectors: {dot_product:.6f}")

vector_operations()

```

Listing 19: Vector Distance and Dot Product Calculation

# Exercise 17:

Let $T$ be an array of integers of size $N$. Write an algorithm and the corresponding Python program that:

- Asks the user to input the elements of $T$.
- Displays the elements of $T$.
- Rearranges the elements of array $T$ in reverse order without using a helper array.
- Displays the resulting array.

**Correction**

**Algorithm**

**Require:** Size $N$ of array $T$ and $N$ integers as input
**Ensure:** Array $T$ reversed in-place
  Read $N$
  **if** $N \leq 0$ **then**
    Output "Error: Array size must be positive."
    **return**
  **end if**
  Initialize empty array $T$
  **for** $i = 0$ to $N - 1$ **do**
    Read integer $x$
    Append $x$ to $T$
  **end for**
  Output "Original array: $T$"
  Initialize $i = 0$, $j = N - 1$
  **while** $i < j$ **do**
    Swap $T[i]$ and $T[j]$
    $i = i + 1$
    $j = j - 1$
  **end while**
  Output "Reversed array: $T$"

**Python Implementation**

## Correction

```python
def reverse_array_in_place():
    try:
        N = int(input("Enter the size of the array (positive integer): "))
        if N <= 0:
            print("Error: Array size must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    T = []
    print(f"Enter {N} integers:")
    for i in range(N):
        while True:
            try:
                x = int(input(f"Element {i + 1}: "))
                T.append(x)
                break
            except ValueError:
                print("Invalid input. Please enter an integer.")

    print("Original array:", T)

    # In-place reversal
    i = 0
    j = N - 1
    while i < j:
        T[i], T[j] = T[j], T[i]
        i += 1
        j -= 1

    print("Reversed array:", T)

reverse_array_in_place()
```

Listing 20: In-Place Array Reversal

# Exercise 18:

Let $T$ be an array of real numbers of size $N$. Write an algorithm and the corresponding Python program that:

- Asks the user to input the elements of $T$.
- Determines the maximum, minimum, the index of the maximum, and the index of the minimum of $T$ (If the array $T$ contains multiple maximums and minimums, the program will display the position of the first occurrence encountered).

**Correction**

**Algorithm**

**Require:** Size $N$ of array $T$ and $N$ real numbers as input
**Ensure:** Maximum value, minimum value, and their indices (first occurrence)

```
Read N
if N ≤ 0 then
    Output "Error: Array size must be positive."
    return
end if
Initialize empty array T
for i = 0 to N − 1 do
    Read real number x
    Append x to T
end for
Output "Array: T"
Initialize max_val = T[0], min_val = T[0]
Initialize max_index = 0, min_index = 0
for i = 1 to N − 1 do
    if T[i] > max_val then
        max_val = T[i]
        max_index = i
    end if
    if T[i] < min_val then
        min_val = T[i]
        min_index = i
    end if
end for
Output "Maximum: max_val at index max_index"
Output "Minimum: min_val at index min_index"
```

**Python Implementation**

**Correction**

```python
def find_max_min_indices():
    try:
        N = int(input("Enter the size of the array (positive integer): "))
        if N <= 0:
            print("Error: Array size must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    T = []
    print(f"Enter {N} real numbers:")
    for i in range(N):
        while True:
            try:
                x = float(input(f"Element {i + 1}: "))
                T.append(x)
                break
            except ValueError:
                print("Invalid input. Please enter a valid real number.")

    print("Array:", T)

    max_val = T[0]
    min_val = T[0]
    max_index = 0
    min_index = 0

    for i in range(1, N):
        if T[i] > max_val:
            max_val = T[i]
            max_index = i
        if T[i] < min_val:
            min_val = T[i]
            min_index = i

    print(f"Maximum: {max_val} at index {max_index}")
    print(f"Minimum: {min_val} at index {min_index}")

find_max_min_indices()
```

Listing 21: Find Max/Min and Their Indices

# Exercise 19:

Write an algorithm and the corresponding Python program to calculate the value of a polynomial of degree $N$.

**Correction**

**Algorithm**

**Require:** Degree $N$ of the polynomial, $N+1$ coefficients $a_0, a_1, ..., a_N$, and a real number $x$
**Ensure:** Value of the polynomial $P(x) = a_0x^N + a_1x^{N-1} + \cdots + a_N$

Read $N$
**if** $N < 0$ **then**
  Output "Error: Degree must be non-negative."
  **return**
**end if**
Initialize array $A$ of size $N+1$
**for** $i = 0$ to $N$ **do**
  Read coefficient $a_i$
  Append $a_i$ to $A$
**end for**
Read real number $x$
Initialize result $= A[0]$
**for** $i = 1$ to $N$ **do**
  result $=$ result $\times\, x + A[i]$
**end for**
Output result

**Python Implementation**

## Correction

```python
def evaluate_polynomial():
    try:
        N = int(input("Enter the degree of the polynomial (non-negative integer): "))
        if N < 0:
            print("Error: Degree must be non-negative.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    coefficients = []
    print(f"Enter {N + 1} coefficients from highest degree to lowest:")
    for i in range(N + 1):
        while True:
            try:
                coeff = float(input(f"Coefficient for x^{N - i}: "))
                coefficients.append(coeff)
                break
            except ValueError:
                print("Invalid input. Please enter a valid real number.")

    try:
        x = float(input("Enter the value of x: "))
    except ValueError:
        print("Invalid input for x. Please enter a real number.")
        return

    result = coefficients[0]
    for i in range(1, N + 1):
        result = result * x + coefficients[i]

    print(f"P({x}) = {result:.6f}")

evaluate_polynomial()
```

Listing 22: Polynomial Evaluation Using Horner's Method

# Exercise 20:

Let $M$ be a matrix of integers of size $L \times C$. Write an algorithm and the corresponding Python program that:

- Asks the user to input the elements of $M$.
- Displays the elements of $M$.
- Calculates and displays the sum of the elements of matrix $M$.
- Calculates and displays the sum of each row of matrix $M$.
- Calculates and displays the sum of each column of matrix $M$.

**Correction**

**Algorithm**

**Require:** Number of rows $L$ and columns $C$ of matrix $M$, followed by $L \times C$ integers
**Ensure:** Display matrix, total sum, row sums, and column sums
  Read $L$
  Read $C$
  **if** $L \leq 0$ OR $C \leq 0$ **then**
    Output "Error: Matrix dimensions must be positive."
    **return**
  **end if**
  Initialize empty matrix $M$
  **for** $i = 0$ to $L - 1$ **do**
    Create empty row $R$
    **for** $j = 0$ to $C - 1$ **do**
      Read integer $x$
      Append $x$ to $R$
    **end for**
    Append $R$ to $M$
  **end for**
  Output "Matrix:"
  **for** $i = 0$ to $L - 1$ **do**
    Output $M[i]$
  **end for**
  Initialize total $= 0$
  **for** $i = 0$ to $L - 1$ **do**
    **for** $j = 0$ to $C - 1$ **do**
      total $=$ total $+ M[i][j]$
    **end for**
  **end for**
  Output "Total sum: total"
  Output "Row sums:"
  **for** $i = 0$ to $L - 1$ **do**
    row_sum $= \sum M[i]$
    Output "Row $i + 1$: row_sum"
  **end for**
  Output "Column sums:"
  **for** $j = 0$ to $C - 1$ **do**
    col_sum $= 0$
    **for** $i = 0$ to $L - 1$ **do**
      col_sum $=$ col_sum $+ M[i][j]$
    **end for**
    Output "Column $j + 1$: col_sum"
  **end for**

**Python Implementation**

## Correction

```python
def matrix_operations():
    try:
        L = int(input("Enter number of rows (positive integer): "))
        C = int(input("Enter number of columns (positive integer):
    "))
        if L <= 0 or C <= 0:
            print("Error: Matrix dimensions must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter integers for dimensions.
    ")
        return

    M = []
    print(f"Enter {L} rows with {C} integers each:")
    for i in range(L):
        row = []
        while len(row) < C:
            try:
                x = int(input(f"Row {i + 1}, Column {len(row) + 1}:
    "))
                row.append(x)
            except ValueError:
                print("Invalid input. Please enter an integer.")
        M.append(row)

    print("\nMatrix:")
    for row in M:
        print(row)

    total = sum(sum(row) for row in M)
    print(f"\nTotal sum of all elements: {total}")

    print("Row sums:")
    for i, row in enumerate(M):
        print(f"Row {i + 1}: {sum(row)}")

    print("Column sums:")
    for j in range(C):
        col_sum = sum(M[i][j] for i in range(L))
        print(f"Column {j + 1}: {col_sum}")

matrix_operations()
```

Listing 23: Matrix Input

# Exercise 21:

Let $A$ be a square matrix of integers of order $N$. Write an algorithm and the corresponding Python program that:

- Declares and initializes matrix $A$.
- Determines and displays the number of non-zero elements of $A$.
- Calculates and displays the trace of $A$.
- Calculates and displays the product of the diagonal elements.
- Determines and displays the transpose of $A$.
- Calculates and displays the matrix $A^2$.

**Correction**

**Algorithm**

**Require:** Order $N$ of square matrix $A$, followed by $N \times N$ integers
**Ensure:** Non-zero count, trace, diagonal product, transpose, and $A^2$

```
Read N
if N ≤ 0 then
    Output "Error: Matrix order must be positive."
    return
end if
Initialize N × N matrix A
for i = 0 to N − 1 do
    for j = 0 to N − 1 do
        Read A[i][j]
    end for
end for
Initialize nonzero_count = 0
Initialize trace = 0
Initialize diag_product = 1
for i = 0 to N − 1 do
    for j = 0 to N − 1 do
        if A[i][j] ≠ 0 then
            nonzero_count = nonzero_count + 1
        end if
    end for
    trace = trace + A[i][i]
    diag_product = diag_product × A[i][i]
end for
Create transpose matrix A^T of size N × N
for i = 0 to N − 1 do
    for j = 0 to N − 1 do
        A^T[i][j] = A[j][i]
    end for
end for
Create matrix A^2 of size N × N
for i = 0 to N − 1 do
    for j = 0 to N − 1 do
        sum = 0
        for k = 0 to N − 1 do
            sum = sum + A[i][k] × A[k][j]
        end for
        A^2[i][j] = sum
    end for
end for
Output nonzero_count, trace, diag_product, A^T, and A^2
```

**Python Implementation**

**Correction**

```python
def matrix_operations():
    try:
        N = int(input("Enter the order of the square matrix (positive integer): "))
        if N <= 0:
            print("Error: Matrix order must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    # Read matrix A
    A = []
    print(f"Enter {N} rows with {N} integers each:")
    for i in range(N):
        row = []
        while len(row) < N:
            try:
                x = int(input(f"Row {i + 1}, Column {len(row) + 1}: "))
                row.append(x)
            except ValueError:
                print("Invalid input. Please enter an integer.")
        A.append(row)

    # Count non-zero elements
    nonzero_count = sum(1 for row in A for val in row if val != 0)

    # Compute trace and diagonal product
    trace = 0
    diag_product = 1
    for i in range(N):
        trace += A[i][i]
        diag_product *= A[i][i]

    # Compute transpose
    A_T = [[A[j][i] for j in range(N)] for i in range(N)]

    # Compute A^2
    A2 = [[0 for _ in range(N)] for _ in range(N)]
    for i in range(N):
        for j in range(N):
            for k in range(N):
                A2[i][j] += A[i][k] * A[k][j]

    # Output results
    print("\nOriginal Matrix A:")
    for row in A:
        print(row)

    print(f"Number of non-zero elements: {nonzero_count}")
    print(f"Trace of A: {trace}")
    print(f"Product of diagonal elements: {diag_product}")

```

**Correction**

```
    print("\nTranspose of A:")
    for row in A_T:
        print(row)

    print("\nMatrix A^2:")
    for row in A2:
        print(row)

matrix_operations()

```

Listing 24: Advanced Matrix Operations

# Exercise 22:

Let $M$ be a matrix of integers of size $L \times C$. Write an algorithm and the corresponding Python program that:

- Asks the user to input the elements of $M$.
- Displays the elements of $M$.
- Transfers the elements of matrix $M$ row by row to a vector $V$.
- Displays the elements of $V$.

**Correction**

**Algorithm**

**Require:** Number of rows $L$ and columns $C$ of matrix $M$, followed by $L \times C$ integers
**Ensure:** Display matrix $M$, vector $V$ (flattened row-wise), and total sum

```
Read L
Read C
if L ≤ 0 OR C ≤ 0 then
    Output "Error: Matrix dimensions must be positive."
    return
end if
Initialize empty matrix M
for i = 0 to L − 1 do
    Create empty row R
    for j = 0 to C − 1 do
        Read integer x
        Append x to R
    end for
    Append R to M
end for
Initialize empty vector V
for i = 0 to L − 1 do
    for j = 0 to C − 1 do
        Append M[i][j] to V
    end for
end for
Output "Matrix:"
for i = 0 to L − 1 do
    Output M[i]
end for
Output "Vector V (row-wise):"
Output V
```

**Python Implementation**

**Correction**

```python
def matrix_to_vector():
    try:
        L = int(input("Enter number of rows (positive integer): "))
        C = int(input("Enter number of columns (positive integer): "))
        if L <= 0 or C <= 0:
            print("Error: Matrix dimensions must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter integers for dimensions.")
        return

    M = []
    print(f"Enter {L} rows with {C} integers each:")
    for i in range(L):
        row = []
        while len(row) < C:
            try:
                x = int(input(f"Row {i + 1}, Column {len(row) + 1}: "))
                row.append(x)
            except ValueError:
                print("Invalid input. Please enter an integer.")
        M.append(row)

    V = []
    for row in M:
        V.extend(row)

    print("\nMatrix:")
    for row in M:
        print(row)
    print("\nVector (row-wise):")
    print(V)

matrix_to_vector()
```

Listing 25: Matrix to Vector Conversion

# Exercise 23:

Write an algorithm and the corresponding Python program that enables:

- Addition of two matrices.
- Multiplication of a matrix by a scalar.
- Multiplication of two matrices.

**Correction**

**Algorithm**

**Require:** Two matrices $A$ and $B$, scalar $k$, and operation type
**Ensure:** Result of selected matrix operation
  Read operation type: "add", "scalar", or "multiply"
  **if** operation is "add" **then**
    Read dimensions $L \times C$ for matrices $A$ and $B$
    Read matrix $A$ of size $L \times C$
    Read matrix $B$ of size $L \times C$
    Initialize matrix $C$ of size $L \times C$
    **for** $i = 0$ to $L - 1$ **do**
      **for** $j = 0$ to $C - 1$ **do**
        $C[i][j] = A[i][j] + B[i][j]$
      **end for**
    **end for**
    Output matrix $C$
  **else if** operation is "scalar" **then**
    Read scalar $k$
    Read dimensions $L \times C$ for matrix $A$
    Read matrix $A$ of size $L \times C$
    Initialize matrix $B$ of size $L \times C$
    **for** $i = 0$ to $L - 1$ **do**
      **for** $j = 0$ to $C - 1$ **do**
        $B[i][j] = k \times A[i][j]$
      **end for**
    **end for**
    Output matrix $B$
  **else if** operation is "multiply" **then**
    Read dimensions $L1 \times C1$ for matrix $A$
    Read dimensions $L2 \times C2$ for matrix $B$
    **if** $C1 \neq L2$ **then**
      Output "Error: Matrices cannot be multiplied."
      **return**
    **end if**
    Read matrix $A$ of size $L1 \times C1$
    Read matrix $B$ of size $L2 \times C2$
    Initialize matrix $C$ of size $L1 \times C2$
    **for** $i = 0$ to $L1 - 1$ **do**
      **for** $j = 0$ to $C2 - 1$ **do**
        $C[i][j] = 0$
        **for** $k = 0$ to $C1 - 1$ **do**
          $C[i][j] = C[i][j] + A[i][k] \times B[k][j]$
        **end for**
      **end for**
    **end for**
    Output matrix $C$
  **end if**

## Correction

### Python Implementation

```
def read_matrix(rows, cols):
    matrix = []
    print(f"Enter {rows} rows with {cols} integers each:")
    for i in range(rows):
        row = []
        while len(row) < cols:
            try:
                x = int(input(f"Row {i + 1}, Column {len(row) + 1}: "))
                row.append(x)
            except ValueError:
                print("Invalid input. Please enter an integer.")
        matrix.append(row)
    return matrix

def matrix_add(A, B):
    return [[A[i][j] + B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

def matrix_scalar_mult(k, A):
    return [[k * A[i][j] for j in range(len(A[0]))] for i in range(len(A))]

def matrix_multiply(A, B):
    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0])
    result = [[0 for _ in range(cols_B)] for _ in range(rows_A)]
    for i in range(rows_A):
        for k in range(cols_A):
            for j in range(cols_B):
                result[i][j] += A[i][k] * B[k][j]
    return result

def main():
    print("Select operation:")
    print("1. Matrix Addition")
    print("2. Scalar Multiplication")
    print("3. Matrix Multiplication")
    choice = input("Enter choice (1/2/3): ")

    if choice == '1':
        try:
            L = int(input("Enter number of rows: "))
            C = int(input("Enter number of columns: "))
            print("Matrix A:")
            A = read_matrix(L, C)
            print("Matrix B:")
            B = read_matrix(L, C)
            result = matrix_add(A, B)
            print("Result (A + B):")

```

**Correction**

```
for row in result:
                print(row)
        except ValueError as e:
            print(f"Error: {e}")

    elif choice == '2':
        try:
            k = float(input("Enter scalar value: "))
            L = int(input("Enter number of rows: "))
            C = int(input("Enter number of columns: "))
            print("Matrix A:")
            A = read_matrix(L, C)
            result = matrix_scalar_mult(k, A)
            print(f"Result (A * {k}):")
            for row in result:
                print(row)
        except ValueError as e:
            print(f"Error: {e}")

    elif choice == '3':
        try:
            print("Matrix A dimensions:")
            L1 = int(input("Rows: "))
            C1 = int(input("Columns: "))
            print("Matrix B dimensions:")
            L2 = int(input("Rows: "))
            C2 = int(input("Columns: "))
            if C1 != L2:
                print("Error: Matrices cannot be multiplied.")
                return
            print("Matrix A:")
            A = read_matrix(L1, C1)
            print("Matrix B:")
            B = read_matrix(L2, C2)
            result = matrix_multiply(A, B)
            print("Result (A * B):")
            for row in result:
                print(row)
        except ValueError as e:
            print(f"Error: {e}")

    else:
        print("Invalid choice.")

main()

```

Listing 26: Matrix Addition