Assessor Feedback:

LO1. Define basic algorithms to carry out an operation and outline the process of programming an applicati

Pass, Merit & Distinction Descripts **P1** ☐ **P2** ☐ **M1** ☐ **D1** ☐

LO2. Explain the characteristics of procedural, object-orientated and event-driven programming, conduct an Integrated Development Environment (IDE).

Pass, Merit & Distinction Descripts **P3** ☐ **M2** ☐ **D2** ☐

LO3. Implement basic algorithms in code using an IDE.

Pass, Merit & Distinction Descripts **P4** ☐ **M3** ☐ **D3** ☐

LO4. Determine the debugging process and explain the importance of a coding standard.

Pass, Merit & Distinction Descripts **P5** ☐ **P6** ☐ **M4** ☐ **D4** ☐

* Please note that grade decisions are provisional. They are only confirmed once

Higher Nationals - Summative Assignment Feedback Form

Student Name/ID	M.M.M AASHIK / E230667		
Unit Title	Unit 01 – Programming		
Assignment Number	1	**Assessor**	
Submission Date		**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	

internal and external moderation has taken place and grades decisions have been agreed at the assessment board.

Assessor Feedback:		
Grade:	**Assessor Signature:**	**Date:**
Resubmission Feedback: • Please note resubmis sion feedback is focussed only on the resubmitted work		
Grade:	**Assessor Signature:**	**Date:**
Internal Verifier's Comments:		
Signature & Date:		

• Please note that grade decisions are provisional. They are only confirmed once internal and external moderation has taken place and grades decisions have been agreed at the assessment board.

Important Points:

1. It is strictly prohibited to use textboxes to add texts in the assignments, except for the compulsory information. eg: Figures, tables of comparison etc. Adding text boxes in the body except for the before mentioned compulsory information will result in rejection of your work.
2. Avoid using page borders in your assignment body.

3. Carefully check the hand in date and the instructions given in the assignment. Late submissions will not be accepted.
4. Ensure that you give yourself enough time to complete the assignment by the due date.
5. Excuses of any nature will not be accepted for failure to hand in the work on time.
6. You must take responsibility for managing your own time effectively.
7. If you are unable to hand in your assignment on time and have valid reasons such as illness, you may apply (in writing) for an extension.
8. Failure to achieve at least PASS criteria will result in a REFERRAL grade.
9. Non-submission of work without valid reasons will lead to an automatic RE FERRAL. You will then be asked to complete an alternative assignment.
10. If you use other people's work or ideas in your assignment, reference them properly using HARVARD referencing system to avoid plagiarism. You have to provide both intext citation and a reference list.
11. If you are proven to be guilty of plagiarism or any academic misconduct, your grade could be reduced to A REFERRAL or at worst you could be expelled from the course
12. Use word processing application spell check and grammar check function to help editing your assignment.
13. Use **footer function in the word processor to insert Your Name, Subject, Assignment No, and Page Number on each pag**e. This is useful if individual sheets become detached for any reason.

STUDENT ASSESSMENT SUBMISSION AND DECLARATION

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

<table>
<tr><td colspan="2">Student name:
M.M.M AASHIK</td><td>Assessor name:</td></tr>
<tr><td>Issue date:
17.06.2024</td><td>Submission date:</td><td>Submitted on:
27.10.2024</td></tr>
<tr><td colspan="3">Programme: Pearson BTEC HND in Computing</td></tr>
<tr><td colspan="3">Unit: Unit 01 – Programming</td></tr>
<tr><td colspan="3">Assignment number and title:
Building a Leave Management System for Grifindo Lanka Toys (Pvt) Ltd.</td></tr>
</table>

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded

from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Guidelines for incorporating AI-generated content into assignments: The use of AI-generated tools to enhance intellectual development is permitted; nevertheless, submitted work must be original. It is not acceptable to pass off AIgenerated work as your own.

Student Declaration

Student declaration I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. Student signature: Date:

Unit 01: PROGRAMMING

Assignment Brief

Student Name/ID Number	M.M.M AASHIK / E230667
Unit Number and Title	**Unit 01 – Programming**
Academic Year	2024/2025
Unit Tutor	
Assignment Title	**Building a Leave Management System for Grifindo Lanka Toys (Pvt) Ltd.**
Issue Date	17.06.2024
Submission Date	27.10.2024
Submission Format	
The assignment submission is in the form of the following. • The submission should be in the form of an individual written report written in a concise, formal technical style using single spacing and font size 12. • Student has to submit the complete GUI System with the database.	

Unit Learning Outcomes

LO1. Define basic algorithms to carry out an operation and outline the process of programming an application.

LO2. Explain the characteristics of procedural, object-orientated and event-driven programming.

LO3. Implement basic algorithms in code using an IDE.

LO4. Determine the debugging process and explain the importance of a coding standard

Transferable skills and competencies developed

Computing-related cognitive skills :

- **Demonstrate knowledge and understanding of essential facts, concepts, principles and theories relating to computing and computer applications**
- **Use such knowledge and understanding in the modelling and design of computerbased systems for the purposes of comprehension, communication, prediction and the understanding of trade-offs**
- **Recognise and analyse criteria and specifications appropriate to specific problems, and plan strategies for their solutions**
- **Critical evaluation and testing: analyse the extent to which a computer-based system meets the criteria defined for its current use and future development**
- **Methods and tools: deploy appropriate theory, practices and tools for the design, implementation and evaluation of computer-based systems.**

Computing-related practical skills :

- **The ability to specify, design and construct reliable, secure and usable computerbased systems**
- **The ability to evaluate systems in terms of quality attributes and possible trade-offs presented within the given problem**
- **The ability to deploy effectively the tools used for the construction and documentation of computer applications, with particular emphasis on understanding the whole process involved in the effective deployment of computers to solve practical problems**
- **The ability to critically evaluate and analyse complex problems, including those with incomplete information, and devise appropriate solutions, within the constraints of a budget.**

Generic skills for employability:

- **Intellectual skills: critical thinking; making a case; numeracy and literacy**
- **Self-management: self-awareness and reflection; goal setting and action planning**
- **Independence and adaptability; acting on initiative; innovation and creativity**
- **Contextual awareness, e.g. the ability to understand and meet the needs of individuals,**

business and the community, and to understand how workplaces and organisations are governed.

Assignment Brief and Guidance:	

Activity 1

A. The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation.

$$F_n = F_{n-1} + F_{n-2}$$

B. Factorial of a non-negative integer, is multiplication of all integers smaller than or equal to n. For example, factorial of 6 is 6*5*4*3*2*1 which is 720.

$$n! = n * (n - 1) * \ldots\ldots 1$$

Define what an algorithm is and outline the characteristics of a good algorithm. Write the algorithms to display the Fibonacci series and the factorial value for a given number using Pseudo code. Determine the steps involved in the process of writing and executing a program and carry out an analysis of writing the code phase by discussing the potential challenges faced.

Take a sample number and dry run the above two algorithms. Show the outputs at the end of each iteration and the final output. Examine what Big-O notation is and explain its role in evaluating efficiencies of algorithms. Write the Python program code for the above two algorithms and critically evaluate their efficiencies using Big-O notation.

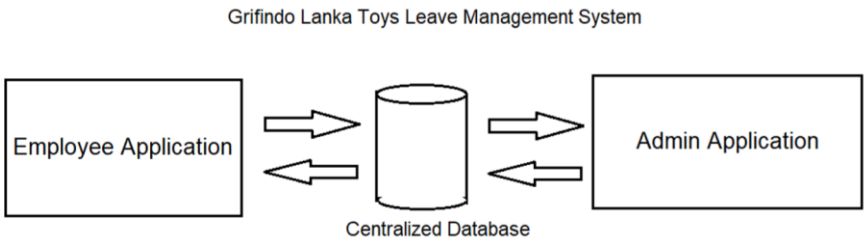
Activity 2

Compare and discuss what is meant by a Programming Paradigm and the main characteristics of Procedural, Object oriented and Event-driven paradigms and the relationships among them. Write small snippets of code as example for the above three programming paradigms using a suitable programming language(s) and critically evaluate the code samples that you have given above in relation to their structure and the unique characteristics.

Activity 3 and Activity 4 are based on the following Scenario.

Grifindo Lanka Toys is a small-scale Toy building company which is in Sri Lanka and currently they have 20 employees working at this company. Grifindo Lanka Toys is the Sri Lankan branch of Grifindo Toys Company which is in UK and previously they have hired your freelance company to develop their payroll system and now they are hiring you again to develop their leave management system for the Sri Lankan branch, Grifindo Lanka Toys Company.

Specifications for the leave management system as follows,



This leave management system has two user roles, and they are Employee and Admin, and they are having two separate desktop applications to interact with the centralized database. Developer can consider the centralized database as a local database for the development purposes.

Functional requirements for the Employee and Admin as follow.

1. Employee Application.
    - Employee should be able to login to the system with the employee number and the given password.
    - Employee should be able to apply for a leave through the system.

      Note:- Normally an employee has 14 annual leaves, 7 casual leaves for a year and 2 shorts leaves per a month. Employee can apply casual leaves as they want before their defined roaster starts. Annual leaves can be applied before 7 days prior to the leave date. Short leave duration is 1 hour and 30 minutes and can be applied for up coming time slots.
    - Employee can view the status of applied leaves (Whether applied leaves are approved or not).
    - Employee can delete applied leaves.
    - Employee can view their remaining leaves and history of applied leaves.
2. Admin Application
    - Admin should be able to login to the leave management system.
    - Admin should be able to register new employees to the system.
    - Admin should be able to define the number of annual leaves, casual leaves for a year and 2 short leaves per every month for every employee. (Newly joined employees will less number of leaves compared to permanent employees)
    - Admin should be able to define the roaster starting time and end time for each employee.

- Admin should be able to approve or reject applied leaves of each employee.
- Admin should be able to view leave reports of,
    - ✦ Individual employee leave history for a given date range.
    - ✦ All employees leave history for a given date range.

Activity 3

3.1. Write the complete pseudocode for the employee apply leave function of the above system. Use the visual studio IDE (using C#.net) to implement the above two applications. Ideally there should be two separate classes for the above two applications and the developer can decide the methods which need to include in those classes. Design the suitable database structure for keeping the data of the above system.

3.2. Analyze the features of an Integrated Development Environment (IDE) and explain how those features help in application development. Evaluate the use of the Visual StudioIDE for your application development contrasted with not using an IDE.

Activity 4

4.1 Design and build two small GUI systems for the above scenario and those two applications should be complete functional systems with all the functions which has described in the above scenario with the database structure which has designed in activity 3.

4.2 Examine debugging process and the features available in Visual studio IDE for debugging your code more easily. Evaluate how you used the debugging process to develop more secure, robust application with examples.

4.3 Explain and outline the coding standards you have used in your application development. Critically evaluate why a coding standard is necessary for the team as well as for the individual.

Reading:

Aho, A. V. et al. (1987) Data Structures and Algorithms. 1st Ed. Addison-Wesley. Hunt, A. et al. (2000) The Pragmatic Programmer: From Journeyman to Master. 1st Ed.

Addison-Wesley. McConnell, S. (2004) Code Complete: A Practical Handbook of Software Construction. 2nd Ed. Microsoft Press.

HN Global:

HN Global HN Global (2021) Reading Lists. Available at: https://hnglobal.highernationals.com/learning-zone/reading-lists

HN Global (2021) Student Resource Library. Available at: https://hnglobal.highernationals.com/subjects/resource-libraries

HN Global (2021) Textbooks. Available at: https://hnglobal.highernationals.com/textbooks

Learning Outcomes and Assessment Criteria

Pass	Merit	Distinction
LO1 Define basic algorithms to carry out an operation and outline the process of programming an application		
P1 Define an algorithm and outline the process in building an application. **P2** Determine the steps taken from writing code to execution.	**M1** Analyse the process of writing code, including the potential challenges faced.	**D1** Evaluate the implementation of an algorithm in a suitable language and the relationship between the written algorithm and the code variant.
LO2 Explain the characteristics of procedural, object-orientated and event-driven programming		
P3 Discuss what procedural, object-orientated and event-driven paradigms are; their characteristics and the relationship between them.	**M2** Compare the procedural, object-orientated and event-driven paradigms used in given source code of an application.	**D2** Critically evaluate the source code of an application that implements the procedural, object-orientated and event-driven paradigms, in terms of the code structure and characteristics.
LO3 Implement basic algorithms in code using an IDE		
P4 Write a program that implements an algorithm using an IDE.	**M3** Enhance the algorithm written, using the features of the IDE to manage the development process.	**D3** Evaluate the use of an IDE for development of applications contrasted with not using an IDE.
LO4 Determine the debugging process and explain the importance of a coding standard		
P5 Explain the debugging process and the debugging facilities available in the IDE. **P6** Explain the coding standard you have used in your code.	**M4** Examine how the debugging process can be used to help develop more secure, robust applications.	**D4** Evaluate the role and purpose of a coding standard and why it is necessary in a team as well as for the individual.

Contents

Big-O-Notation

Use of Big-O-Notation

Properties of Big-O-Notation

Evaluating the Pseudo code using Big-O-Notation

Evaluating the Python using Big-O-Notation

Challengers faced during the creation of Algorithm in Activity 1

ACTIVITY 02

Programming paradigm

Procedural Programming

Characteristics of Procedural Programming

Advantages and disadvantages of Procedural Programming

Object-Oriented Programming (OOP)

Characteristics of OOP

Advantages and disadvantages of OOP

Event-Driven Programming

Characteristics of Event-Driven Programming

Advantages and Disadvantages of Event Driven Programming

ACTIVITY 03

Pseudo code for Employee Apply Leave Function

Database Structure of Grifindo Lanka Toy Company

Database

What is Structured Query Language (SQL)?

CONECTING SQL SERVER AND THE VISUAL STUDIO

Integrated Development Environment (IDE)

Visual Studio IDE

ACTIVITY 04

Create a simple graphical user interface (GUI) for Grifindo Lanka Toys

Developing the System

Debugging Process

Coding Standards

Acknowledgement

First of all I would like to express my sincere gratitude to the following individuals and

Entities who have played a significant role in completion of the assignment. Ms.Vindya karunarathna, for providing guidance and feedback throughout the entire process. Your expertise and support have been instrumental in shaping the quality of the assignment. And also I like to thank my parents for supporting and assigning my to HND in Esoft to succeed in life by educating me to become a better person. Finally I would like to thank my friends to be supportive as best they can to be productive and improve. I appreciate all your contributions to my growth and development.

Thank you,

You're sincerely,

M.M.M Aashik.

Introduction

My name is Aashik. As an undergrad at ESoft Metro Campus in Kandy, I am

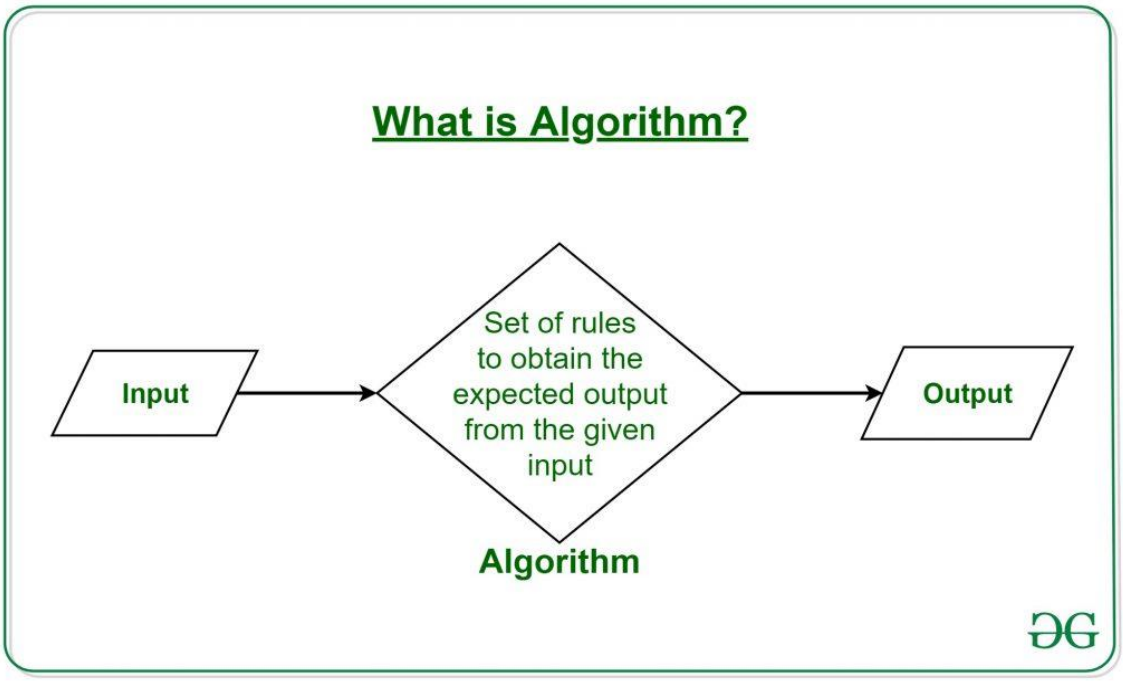
Currently working on developing a fully functional and productive system for a semester one programming assignment. This document explains how the system operates, what it

needs to run, its various functions, the security system that was put in place, and how the entire system will be maintained in the future. It also explains how the end user learns how to use the system correctly.

ACTIVITY 01

1.1 What is an algorithm?

An algorithm is a set of instructions that a computer must follow to perform a calculation or other problem-solving task. By formal definition, an algorithm is a finite set of instructions that are executed in a specific order to complete a particular task. It is not a complete program or code, but rather an informal description of the simple logic of the problem in the form of a flowchart or pseudo code. An algorithm is a step-by-step technique for solving a problem. (Upadhyay, 2024)



1.2Characteristics of an algorithm

An algorithm is a systematic process used to solve a task or problem. Several important factors affect the effectiveness of an algorithm: (Upadhyay, 2024)

1. Finiteness

An algorithm should always complete a finite number of steps before terminating. It should have a defined endpoint or terminate when the operations are complete and should not enter an infinite loop.

2. Clarity

An algorithm must have each step precisely defined. Clear, easy-to-understand instructions make each step easy to understand and execute.

3. Inputs

An algorithm requires one or more inputs. The values that are initially fed into an algorithm before being processed are called inputs. These inputs are taken from a given range of acceptable values.

4. Outputs

An algorithm must produce one or more outputs. The outputs are the results of the algorithm after each step is completed. The relationship between the inputs and the results must be clear.

5. Effectiveness

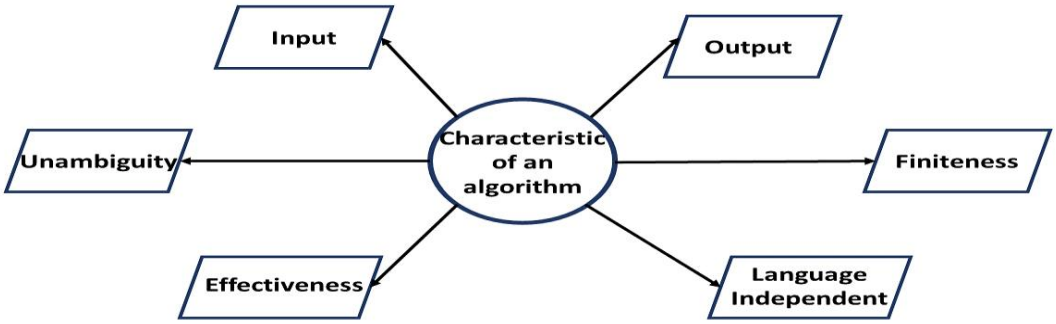
The steps of an algorithm should be simple enough to complete basic operations within a limited time. All operations of the algorithm should be feasible and practical given the available resources.

6. Generality

An algorithm should not be limited to a single case, but should be able to solve a wide range of problems. It should provide a general solution for managing a variety of inputs within a particular area or domain.

7. Language independent

The algorithm must be written as clear, concise instructions.



Use of Algorithm

Brute Force Algorithm: The Brute Force Algorithm stands as the most rudimentary and straightforward type of algorithm. It represents the initial and instinctive approach to a problem—essentially, the first method that comes to mind when encountering the issue. Technically, it involves systematically exploring every potential option or possibility to address and resolve the problem at hand.

Example:

If there is a lock of 4-digit PIN. The digits to be chosen from 0-9 then the brute force will be trying all possible combinations one by one like 0001, 0002, 0003, 0004, and so on until we get the right PIN. In the worst case, it will take 10,000 tries to find the right combination.

Recursive Algorithm: This algorithmic type relies on recursion, where a problem is resolved by dividing it into similar sub problems and repeatedly calling itself until reaching a solution, often guided by a base condition. Recursive algorithms handle various problems like computing factorials, generating Fibonacci sequences, solving the Tower of Hanoi puzzle, implementing Depth-First Search for graphs, among others.

Randomized Algorithm: In randomized algorithms, the use of random numbers plays a key role in determining anticipated outcomes. The selection of these random numbers is strategic, aiming to provide immediate advantages.

Randomized algorithms tackle various problems, like Quicksort, where random numbers assist in choosing the pivot for sorting.

Sorting Algorithm: The sorting algorithm is utilized to arrange data either in ascending or descending order, aiming for an organized and optimized data arrangement. It efficiently arranges data for easier processing. Various problems find solutions through sorting algorithms, such as Bubble Sort, Insertion Sort, Merge Sort, Selection Sort, and Quick Sort, which represent different methods within this algorithmic category.

Searching Algorithm: The searching algorithm is designed to locate a specific key within data, whether sorted or unsorted. This algorithm tackles various problems, with examples like Binary Search or Linear Search, showcasing its application in identifying elements within datasets.

1.3Steps to design a program

There are five basic steps to follow when building software:

I. Define the Problem

- Start with a clear understanding of the problem or task the software needs to solve.
- Identify the program objectives by gathering end-user or stakeholder requirements.

II. Designing the Solution

- Develop a comprehensive project plan, including a general schedule and resource allocation.
- Design the software architecture and explain how the various components interact.

III. Develop the Program

- Create the actual code for the software based on the specification and design.

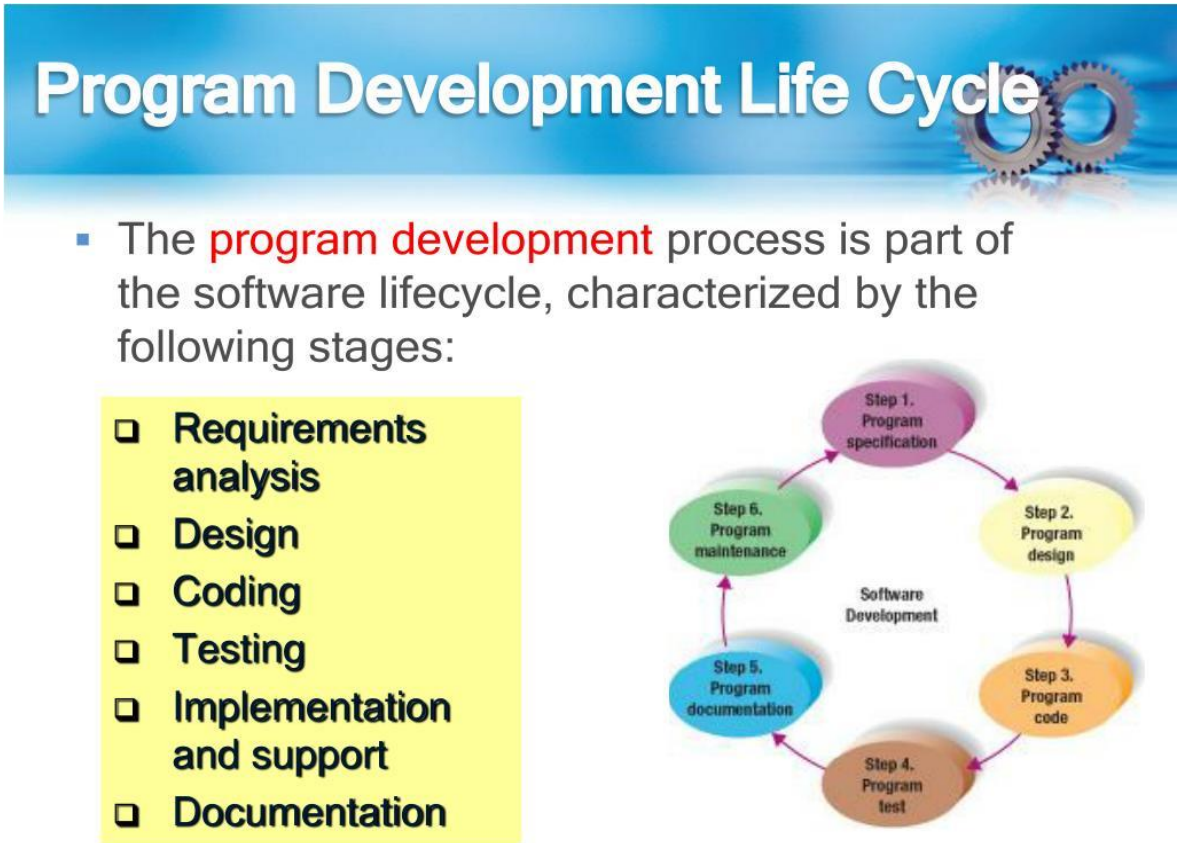
- Use best practices and coding standards to create clean, maintainable code.
- Use version control tools to track changes and collaborate with team members (if applicable).

IV. Testing the program

- Create and run test cases to ensure that the software works as intended.
- Use testing and debugging to find and resolve errors and other problems.
- Ensure that the program meets its intended specifications and functionality.

V. Document and Maintain

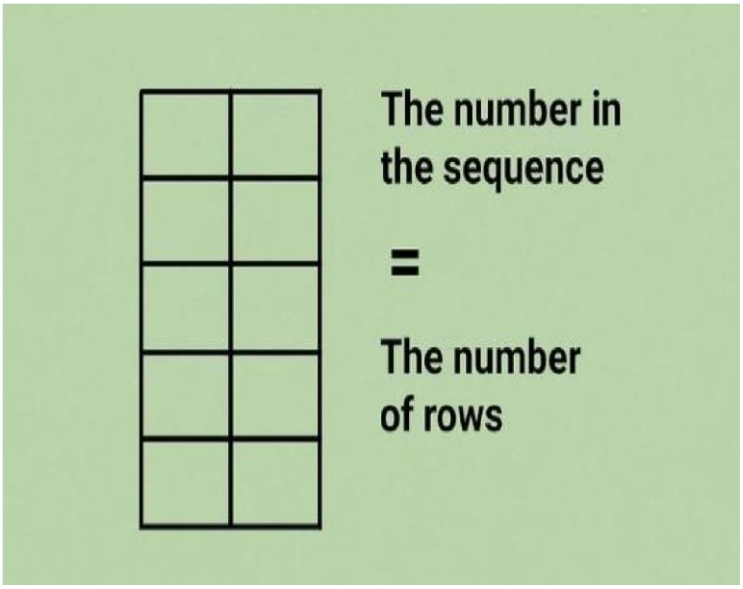
- Set up the software for installation in the desired environment.
- Make the software available to users throughout the deployment.
- Provide ongoing maintenance and support, including upgrades and bug fixes as needed.



Fibonacci sequence

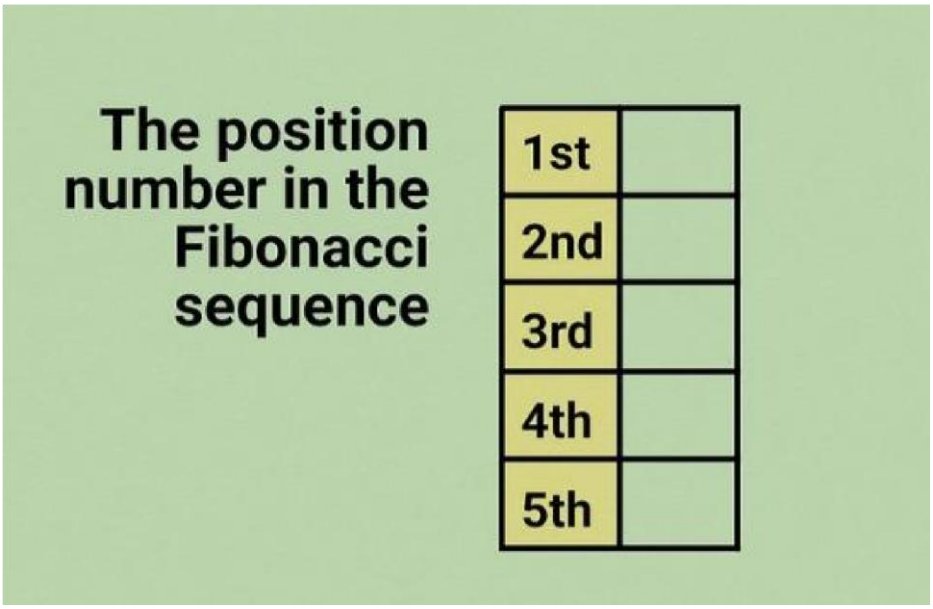
The Fibonacci sequence is a series of gradually increasing integers (the Fibonacci numbers) that starts at zero, then one, then another one, and finally, each number in the sequence is equal to the sum of the two numbers before it. (Sheldon, 2022)

How to calculate Fibonacci sequence

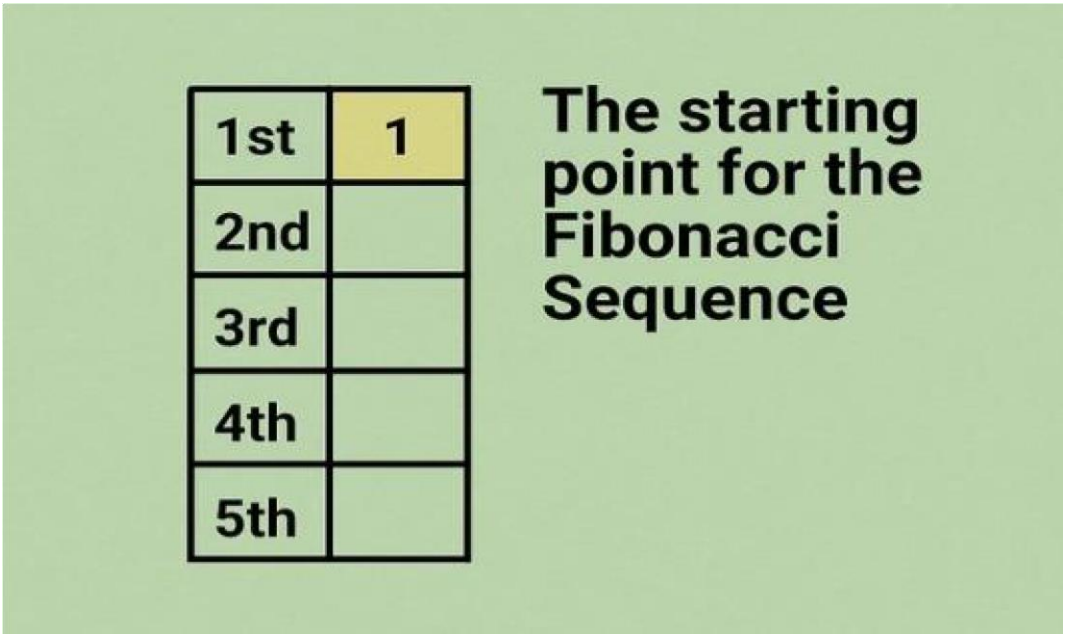


1 **Set up a table with two columns.** The number of rows will depend on how many numbers in the Fibonacci sequence you want to calculate.[2]
For example, if you want to find the fifth number in the sequence, your table will have five rows.

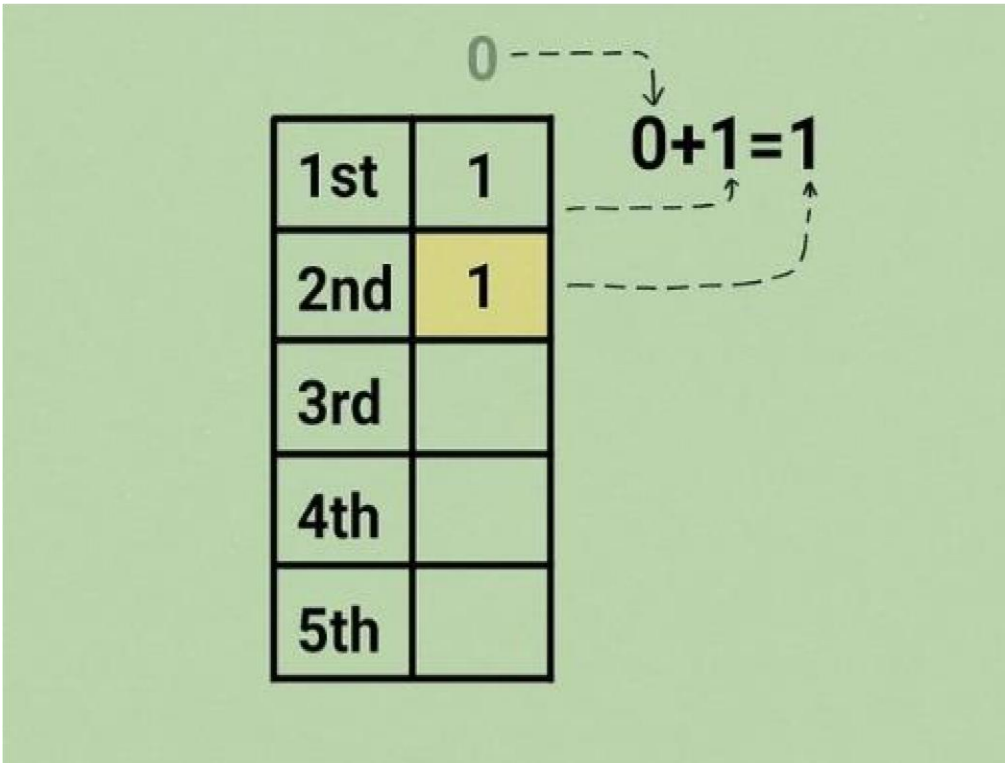
> When using the table method, you cannot find a random number farther down in the sequence without calculating all the number before it. For example, if you want to find the 100th number in the sequence, you have to calculate the 1st through 99th numbers first. This is why the table method only works well for numbers early in the sequence.



2 **Enter the sequence of terms in the left column.** This means just entering Sequence of sequential ordinal numbers, beginning with "1st." The term refers to the position number in the Fibonacci sequence. For example, if you want to figure out the fifth number in the sequence, you will write 1st, 2nd, 3rd, 4th, 5th down the left column. This will show you what the first through fifth terms in the sequence are.



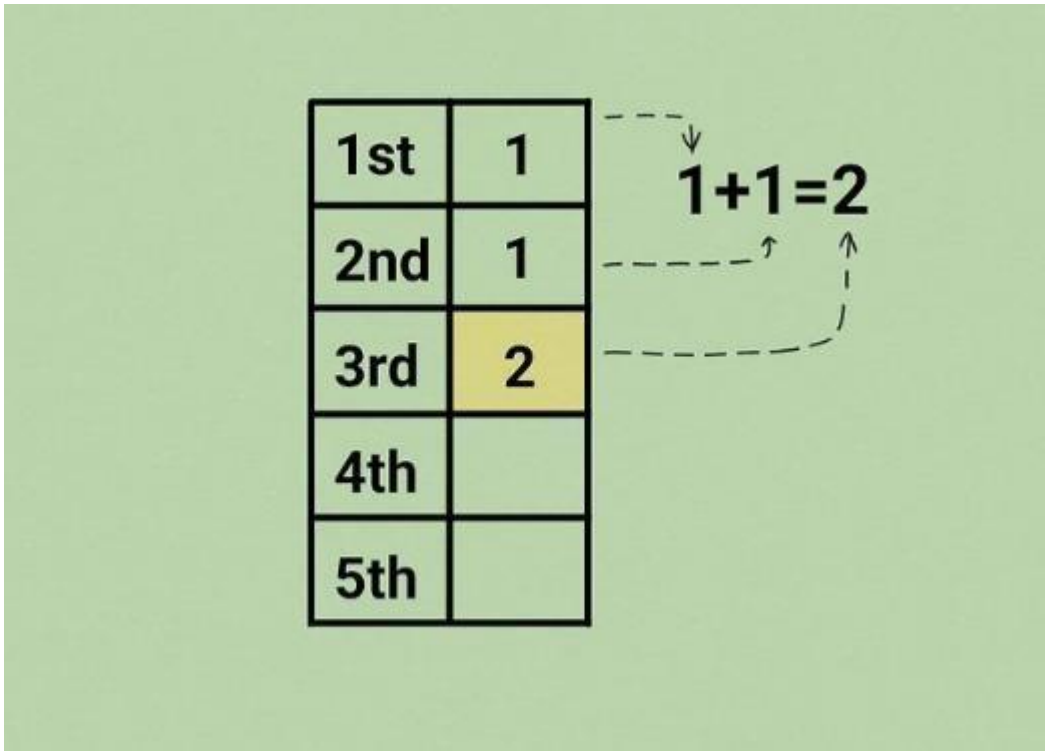
3 **Enter 1 in the first row of the right-hand column.** This is the starting point for the Fibonacci sequence. In other words, the first term in the sequence is 1. The correct Fibonacci sequence always starts on 1. If you begin with a different number, you are not finding the proper pattern of the Fibonacci sequence.



4 **Add the first term (1) and 0.** This will give you the second number in the sequence.

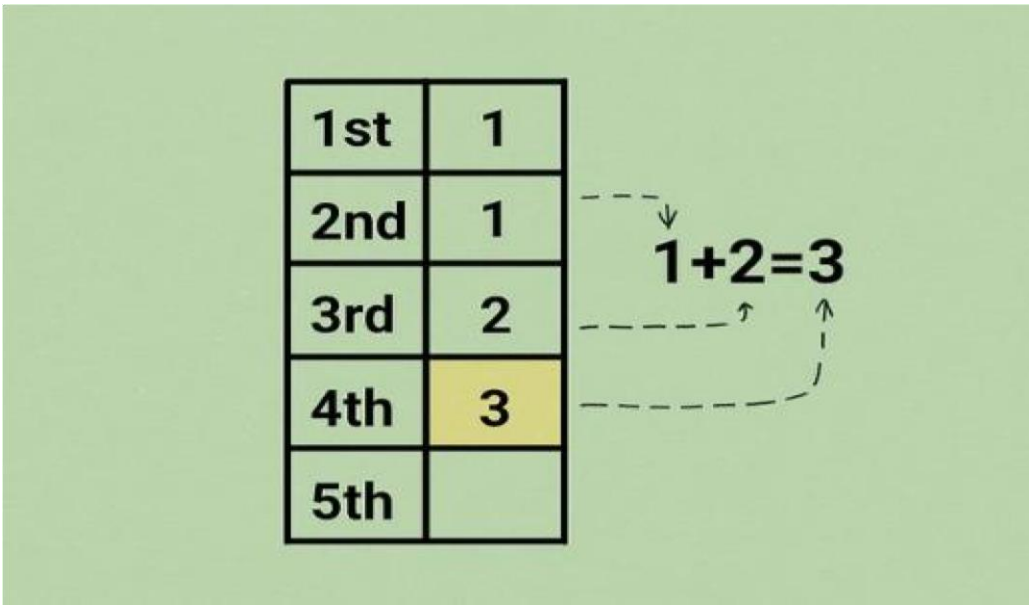
Remember, to find any given number in the Fibonacci sequence, you simply add the two previous numbers in the sequence.

To create the sequence, you should think of 0 coming before 1 (the first term), so 1 + 0 = 1.



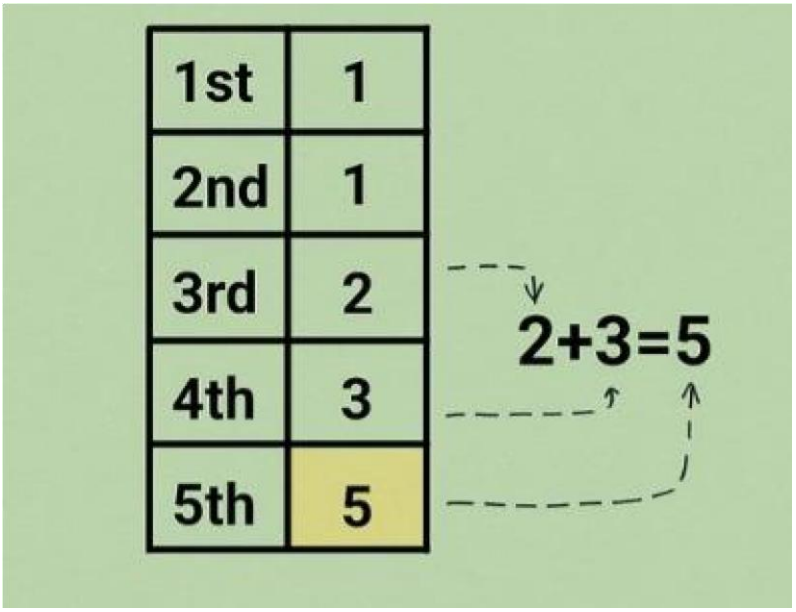
5 **Add the first term (1) and the second term (1).** This will give you the third number in the sequence.[3]

1 + 1 = 2. The third term is 2.



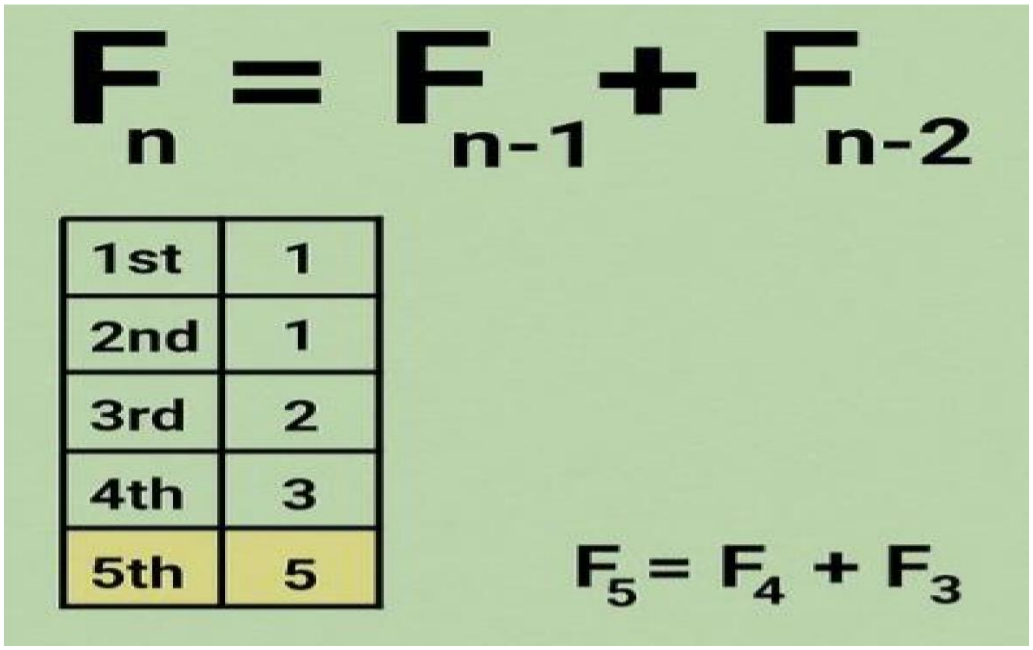
6 **Add the second term (1) and the third term (2) to get the fourth number in the sequence.**

1 + 2 = 3. The fourth term is 3.



7 **Add the third term (2) and the fourth term (3).** This will give you the fifth number in the sequence.[4]

2 + 3 = 5. The fifth term is 5.



8 Sum the previous two numbers to find any given number in the Fibonacci sequence. When you use this method, you are using the formula **Fn = Fn-1 + Fn-2**.[5] Since this is not a closed formula, however, you cannot use it to calculate any given term in the sequence without calculating all the previous numbers.[6]

Pseudo Code

Pseudo code serves as an informal method of programming explanation, free from strict syntax or specific technology constraints. It acts as a blueprint or initial draft for a program, outlining its structure without delving into specific coding nuances. This approach captures the program's flow without intricate technical specifics. Designers employ pseudo code to communicate project requirements to programmers, ensuring alignment between the software's needs and the actual code implementation.

1.4.1 Pseudo Code for Fibonacci sequence

```
start

Define a,b,n,c
Input n
a=o
b=1

if n<0:
     Display"Please enter a positive integer"

        else if n=1
            Display "Fibonacci sequence upto",n
            Display a

                else
                        Display "Fibonacci sequence upto",n
                        Display a
                        print b
                                        for i=2 to n
                                            c=a+b
                                            a=b
                                            b=c
                                            display c

End
```

1.4.2 Python Code for Fibonacci sequence

```
# Function for nth Fibonacci number
def fib(n):
    a = 0
    b = 1

    if n<=0:
        print("Please enter a positive integer")
    elif n == 1:
        print("Fibonacci sequence upto",n,":")
        print(a)
    else:
        print("Fibonacci sequence upto",n,":")
        print(a)
        print(b)
        for i in range(2,n):
            c = a + b
            a = b
            b = c
            print(c)

# Main Program

n=int(input("how many terms ? \n"))
fib(n)
```

1.4.3 Dry run and Output for the above pseudo Code and the Python Code

- **Input(n) = 0**

n	n<0	n=1	i	i<n	c	a	b	output
0	T					0	1	Please enter a positive integer

```
>>>
============= RESTART: E:\Esoft\Sem 01\Programing\Assessment\fib.py
how many terms ?
0
Please enter a positive integer
>>>
```

- **Input(n) = 1**

n	n<0	n=1	i	i<n	c	a	b	output
1	F	T				0	1	0

```
>>>
============= RESTART: E:\Esoft\Sem 01\Programing\Assessment\fib.py ============
how many terms ?
1
Fibonacci sequence upto 1 :
0
>>>
```

- **Input(n) = 4**

n	n<0	n=1	i	i<n	c	a	b	output
4	F	F				0	1	0,1
			2	T	1	1	1	1
			3	T	2	1	2	2
			4	F				

```
>>>
============= RESTART: E:\Esoft\Sem 01\Programing\Assessment\fib.py ============
how many terms ?
4
Fibonacci sequence upto 4 :
0
1
1
2
```

- **Input(n) = 6**

n	n<0	n=1	i	i<n	c	a	b	output
6	F	F				0	1	0,1
			2	T	1	1	1	1
			3	T	2	1	2	2
			4	T	3	2	3	3
			5	T	5	3	5	5
			6	F				

```
>>>
========== RESTART: E:\Esoft\Sem 01\Programing\Assessment\fib.py ==========
how many terms ?
6
Fibonacci sequence upto 6 :
0
1
1
2
3
5
```

1.5 Factorial

The Factorial of a whole number 'n' is defined as the product of that number with every whole number less than or equal to 'n' till 1. For example, the factorial of 4 is 4 * 3 * 2 * 1, which is equal to 24. The factorial of a whole number is the function that multiplies the number by every natural number below it. (Anon., 2022)

Example:
The factorial of 10, represented as 10! is calculated as:
10! = 10 × 9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1 = 3,628,800

n Factorial	n (n - 1) (n - 2) 1	n! = n × (n - 1)!	Result
1 Factorial	1	1	1
2 Factorial	2 × 1	= 2 × 1!	= 2
3 Factorial	3 × 2 × 1	= 3 × 2!	= 6
4 Factorial	4 × 3 × 2 × 1	= 4 × 3!	= 24
5 Factorial	5 × 4 × 3 × 2 × 1	= 5 × 4!	= 120
6 Factorial	6 × 5 × 4 × 3 × 2 × 1	= 6 × 5!	= 720
7 Factorial	7 × 6 × 5 × 4 × 3 × 2 × 1	= 7 × 6!	= 5040
8 Factorial	8 × 7 × 6 × 5 × 4 × 3 × 2 × 1	= 8 × 7!	= 40320
9 Factorial	9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1	= 9 × 8!	= 362880
10 Factorial	10 × 9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1	= 10 × 9!	= 3628800

1.5.1 Pseudo Code for Factorial

```
start

Define n, fac
Input n
fac=1

if n<0:
     Display"Sorry, Factorial cannot be calculated"

      else if n=0
         Display "Factorial of the number is 1"
            else
                  Display "Factorial of the given number is: "
                  for i=1 to n
                  fac=fac*i
            Display fac
End
```

1.5.2 Python Code for Factorial

```
def factorial(n):
    fac=1

    if n<0:
        print("Sorry, Factorial cannot be calculated")

    elif n==0:
        print("Factorial of the number is 1")

    else:
        print ("Factorial of the given number is: ")
        for i in range(1,n+1):
            fac=fac*i
        print (fac)

#main program
n=int(input("Enter a non negative integer number\n"))
factorial(n)
```

1.5.3 Dry run and Output for the above pseudo Code and the Python Code

- **Input(n) = 4**

n	n<0	n=0	i	i<=0=n	fac	Output
4	F	F	1	T	1	
			2	T	2	
			3	T	6	
			4	T	24	
			5	F		24

```
============= RESTART: E:\Esoft\Sem 01\Programing\Assessment\fac.py ============
Enter a non negative integer number
4
Factorial of the given number is:
24
```

- **Input(n) = 6**

n	n<0	n=0	i	i<=0=n	fac	Output
6	F	F	1	T	1	
			2	T	2	
			3	T	6	
			4	T	24	
			5	T	120	
			6	T	720	
			7	F		720

```
============= RESTART: E:\Esoft\Sem 01\Programing\Assessment\fac.py ============
Enter a non negative integer number
6
Factorial of the given number is:
720
>>>
```

- **Input(n) = 0**

n	n<0	n=0	i	i<=0=n	fac	Output
0		T				Factorial of the number is 1

```
============= RESTART: E:\Esoft\Sem 01\Programing\Assessment\fac.py ============
Enter a non negative integer number
0
Factorial of the number is 1
```

- **Input(n) = (-6)**

n	n<0	n=0	i	i<=0=n	fac	Output
-6	T					Sorry, Factorial cannot be calculated

```
============= RESTART: E:\Esoft\Sem 01\Programing\Assessment\fac.py ============
Enter a non negative integer number
-6
Sorry, Factorial cannot be calculated
```

Big-O-Notation

Big O notation, stemming from a group of symbols formulated by German mathematicians like Paul Bachmann and Edmund Landau, elucidates the behaviour of a function nearing a specific value or infinity. This symbol, also known as Bachmann–Landau notation or asymptotic notation, uses the letter O, chosen by Bachmann to represent the order of approximation.

In computer science, Big O notation sorts algorithms based on how their runtime or space requirements scale concerning the input size. Additionally, within analytic number theory, it often sets a limit on the variance between an arithmetic function and a more understandable approximation, such as the remainder term within the prime number theorem. This notation extends beyond mathematics and computer science, offering analogous estimations and applications in diverse domains.

1.6.1. Use of Big-O-Notation

Big O notation finds primary applications in two domains:
In mathematics, it commonly characterizes how effectively a finite series represents a given function, notably in scenarios like truncated Taylor series or asymptotic expansions.

In computer science, it aids in analysing algorithms.
In both these spheres, the function g(x) within O (·) is typically chosen to be as simple as feasible, excluding constant factors and lower-order terms.

There exist two closely related but discernibly distinct usages of this notation:

- Infinite asymptotic.
- Infinitesimal asymptotic.

However, this divergence pertains solely to application, not to underlying principles. The formal definition for "big O" remains consistent across both scenarios, differing only in the limits applied to the function argument.

1.6.2. Properties of Big-O-Notation

The Big O algorithm in data structures encompasses several crucial properties that are mandatory. These essential properties are outlined below:

Summation Function:

For a function f(n) = f1(n) + f2(n) + — + fm(n), where fi(n) ≤ fi+1(n) for all i = 1, 2, – , m, the notation O(f(n)) equates to O(max(f1(n), f2(n), –, fm(n))).

Logarithmic Function:

Given functions f(n) = logan and g(n) = logbn, their respective Big O notations, O(f(n)) and O(g(n)), are equivalent.

Constant Multiplication:

When f(n) = c.g(n), with c representing a nonzero constant, then the Big O notation of f(n) equals the Big O notation of g(n), denoted as O(g(n)).

Polynomial Function:

For a function f(n) = a0 + a1.n + a2.n2 + — + am.nm, the Big O notation of f(n) simplifies to O(nm).

Evaluating the Pseudo code using Big-O-Notation

Factorial

The given factorial pseudo code utilizes a loop that ranges from 2 to 'n', computing the factorial by progressively multiplying 'result' by 'i'. This exhibits a linear time complexity of O(n) in Big O notation. As 'n' increases, the loop's iterations grow linearly, resulting in a proportional increase in computation time. Its effectiveness allows it to handle larger 'n' values without facing exponential or factorial time escalation, maintaining a consistent linear complexity pattern as 'n' expands.

Fibonacci

The given Fibonacci pseudo code employs a loop to compute Fibonacci numbers from 2 to 'n' by adding the preceding two numbers in each step. It showcases a time complexity of O(n) in Big O notation, indicating that as 'n' grows, the loop iterations expand proportionally, resulting in a

linear rise in computation time. Its efficiency permits handling larger 'n' values without encountering exponential or factorial time escalation, maintaining a linear growth pattern in complexity as 'n' increases.

Evaluating the Python using Big-O-Notation

Factorial

The Python code given to calculate the factorial involves a loop that runs 'n' times from 2 to 'n', multiplying 'result' by 'i' in each iteration to find the factorial. This approach exhibits a linear time complexity of O(n) in Big O notation. As 'n' grows, the loop operates 'n' times proportionally, resulting in an incremental rise in computation time. Its effectiveness allows the algorithm to manage larger 'n' values without experiencing exponential or factorial increases in time, maintaining a steady linear growth in complexity as 'n' expands.

Fibonacci

The given Python code for the Fibonacci sequence utilizes a loop that iterates 'n' times from 2 to 'n', computing Fibonacci numbers by adding 'a' and 'b' in each cycle. This algorithm showcases a linear time complexity of O(n) according to Big O notation. As 'n' grows, the loop performs 'n' iterations in direct proportion, resulting in a linear increase in computational time. Its efficiency allows it to manage larger 'n' values without experiencing exponential or factorial time surges, preserving a steady linear growth in complexity as 'n' increases.

Challengers faced during the creation of Algorithm in Activity 1

The intricacies of problem complexity can hinder the creation of straightforward and effective algorithms.

Designing an algorithm that precisely aligns with the problem's criteria poses a challenge, demanding both creativity and an in-depth comprehension of the issue at hand.

Juggling efficiency, encompassing time and space complexity, is a common dilemma. Striking the perfect balance often necessitates trade-offs between speed and memory usage. Addressing diverse scenarios and edge cases presents a hurdle, demanding rigorous testing to ensure the algorithm performs accurately in all conditions.

Fine-tuning algorithms for optimal performance, especially on extensive datasets or in real-time scenarios, requires ongoing evaluation, meticulous profiling, and continuous refinement. Formally analysing algorithms for correctness and efficiency is a rigorous task, demanding expertise and a deep understanding of the subject matter.

Ensuring that an algorithm remains maintainable, readable, and adaptable to evolving requirements or added features is vital yet challenging.
Translating the algorithm design into code may introduce errors, particularly when handling intricate logic or complex data structures.

ACTIVITY 02

2.1 Programming paradigm

A paradigm is often referred to as a method of problem solving or accomplishing a task. A programming paradigm is a method for solving a problem using a programming language. Alternatively, it can be defined as a method of problem solving that employs a certain strategy and the use of available tools and procedures. Many well-known programming languages require specific implementation strategies, which are referred to as paradigms. Aside from diverse programming languages, there are numerous paradigms that cater to every requirement. (bhumika_rani, 2024)

2.2 Procedural Programming

Procedural programming evolved from structured programming and can be defined

As programming approach based on the concept of procedure invocation. A process, also known as a routine, subprogram, or function, is merely a series of computational steps to be completed. During programme execution, any procedure can be called at any time, either from another procedure or from itself. (bhumika_rani, 2024)

Characteristics of Procedural Programming

- **Predefined Functions**

In procedural programming languages, predefined functions are functions that are included in a library of available functions. These functions allow the programmer to perform common operations without writing the necessary code. For the programmer, this reduces production time.

- **Local Variables**

A local variable is a programming term that has a limited scope. This means that the variable only works within the function for which it was designed. Local variables can only be used in this way. Therefore, if a developer or user tries to use them in a way that does not correspond to their intended purpose, the code may crash and the task may remain unfinished.

- **Global Variables**

Global variables provide extended functionality when local variables are not sufficient. Developers can use global variables in almost every function. Variables declared globally are accessible to all methods and functions in the code, allowing programmers to access critical information throughout the entire program operation.

- **Parameter Transfer**

Data values passed between functions in a code sequence are called parameters. Developers pass input parameters to a module or device and receive output parameters. Programmers can use parameters to perform calculations on data by assigning it to variables, equations, and statements in a function.

- **Modularity**

Developers can use modular design frameworks to break down the functionality of their code into more manageable blocks. These blocks are sometimes called methods or functions in procedural programming languages, and programmers access them through this code. Repeatability of key functions makes setup code more efficient. Previously, programmers had to use the same code over and over again to complete tasks that needed to be performed multiple times.

- **Top-down Approach**.

Procedural programming uses a top-down design and creation method. Using this strategy, the programmer determines the main goal of the program and analyzes the elements required to achieve that goal. He may decide to split some of the components of smaller targets into individual units. The developer can then start developing the modules to generate the operating code for the software. (Team, 2024)

```
#global variable
pi = 3.14

#calculate area of circle
def area_base(radius):
    area=pi*radius*radius
    return area

#calculate the cylinder volume
def cylinder_volume(height,radius):
    base_area=area_base(radius)
    volume=height*base_area
    return volume

#calculating n cylinders volumes
total_volume=0
volume=0
n=int(input("How many Cylinders? "))
if n==1:
    height = int(input("cylinder height? "))
    radius = int(input("radius of the cylinder?"))
    volume=cylinder_volume(height,radius)
    print("The volume of the cylinder ",n," is:",round(volume,2))
else:
    for i in range (1,n+1):
        height = int(input("cylinder height? "))
        radius = int(input("radius of the cylinder?"))
        total_volume+=volume
        volume=cylinder_volume(height,radius)
        print("The volume of theth cylinder ",i,"is:",round(volume,2))
    print("The totalvolume of the cylinders is:",round(total_volume,2))
```

- Predefined functions: The code does not use any predefined functions, but it defines two custom functions, area base () and cylinder volume (), to perform specific tasks.

- Local variables: The code uses local variables like area and base area inside the functions area base () and cylinder volume (), respectively. These variables are only accessible within their respective functions.

- Global variables: The code defines a global variable pi, which can be accessed from anywhere in the program.

- Parameter passing: The code passes parameters to functions like radius, height, and n to specify the necessary values needed for computation.

- Modularity: The code follows the principle of modularity by defining separate functions for calculating the area of the base of a cylinder and

the volume of the cylinder. This enables code reuse and makes the program more organized.

- Top-down approach: The code follows a top-down approach, where the main program calls the custom functions area base () and cylinder volume () to perform specific tasks. This approach simplifies the program and makes it easier to understand and debug.

Advantages and disadvantages of Procedural Programming

Procedural programming is a programming technique that focuses on functions or procedures. One of its primary advantages is its ease of learning and maintenance, as its structured design allows for the detection and correction of faults. Furthermore, it is better optimized for speed and efficiency of code execution than other programming styles. Procedural code can also be divided down into reusable functions, making it easier to maintain and extend. However, there are several disadvantages to procedural programming, such as Limited abstraction and challenges when working with huge programmes. It has concurrency difficulties and is not appropriate for GUI-based applications. It provides modularity but lacks the flexibility of other paradigms like object-oriented or functional programming, which might limit code reuse.

2.3 Object-Oriented Programming (OOP)

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well suited for software that is large, complex and actively updated or maintained. This includes programs for manufacturing and design, as well as mobile applications.

Example

An object can range from physical entities, such as a human being who is

Described by properties like name and address, to small computer programs, such as widgets. Once an object is known, class of objects that defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. Objects can communicate with well-defined interfaces

Characteristics of OOP

- **Classes**

  Classes are groups of objects that have similar data-part attributes. A class is a blueprint for an object that contains all of its characteristics, methods, and other features.

- **Objects**

  An object is a physical thing with certain qualities and actions. Memory is allocated when an object is instantiated at that moment. Each item may interact with each other without knowing the specifics of their information or code.

- **Encapsulation**

  The encapsulation principle states that all important information is contained inside an object and only select information is exposed. The implementation and state of each object are privately held inside a defined class. Other objects do not have access to this class or the authority to make changes. They are only able to call a list of public functions or methods. This characteristic of data hiding provides greater program security and avoids unintended data corruption. (Gillis, 2024)

- **Abstraction.**

  A developer must include abstraction into software or a product for a better user experience. In essence, abstraction keeps all problems hidden and only communicates what is absolutely necessary to the outside world.

- **Inheritance.**

  Inheritance refers to taking on the characteristics of another individual. Data and methods can be passed down from one class to another, much to how you pick up traits and habits from your parents. The capability of oops to combine and inherit objects and their attributes is one of its most important and fundamental features.

- **Polymorphism.**

  Objects are designed to share behaviors, and they can take on more than one form. The program determines which meaning or usage is necessary for each execution of that object from a parent class, reducing the need to duplicate code. A child class is then created, which extends the functionality of the parent class. Polymorphism enables different types of objects to pass through the same interface.

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def get_info(self):
        return f"This car is a {self.brand} {self.model}"

my_car = Car("Rolls Royce", "Boat Tail")
print(my_car.get_info())
```

Result

```
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:/Users/NEXEN Technologies/AppData/Local/Programs/Python/Python3
.py
This car is a Rolls Royce Boat Tail
```

Advantages and disadvantages of OOP

Instead of writing code from scratch, you can build programs from standard working modules that communicate with each other, saving development time and increasing productivity. OOP languages allow you to break programs into smaller problems that are easier to solve (one object at a time). This new technology aims to increase programming productivity, improve software quality, and reduce maintenance costs. OOP systems are easily upgradeable from small to large systems. Multiple instances of an object can coexist without interference. The division of responsibilities in the project is very simple, based on objects. Objects in the problem domain can be mapped to objects in the program. The principle of data hiding helps the programmer to write secure programs where the code of other parts of the program cannot be intruded into. Inheritance eliminates redundant code and extends the use of existing classes. Message passing techniques are used for communication between objects, making it very easy to create interfaces to external systems. A data-centric design approach allows you to capture the details of your model in an implementable format. Using OOP languages results in significantly longer programmes compared to procedural approaches. Larger programmes take longer to complete and run slower. OOP is not a universal language, hence it does not apply everywhere. It is only used when needed. It is not appropriate for every type of situation. OOP is tough to employ, thus a programmer must have strong design and programming skills, as well as solid planning. It takes time to adjust to OOP. Some people may struggle with the thinking process required for object-oriented programming. In OOP, everything is considered an object. As a result, you must have a solid understanding of things before using it.

2.4 Event-Driven Programming

Event-driven programming focuses on specific events. Ultimately, programme flow is dependent on occurrences. So far, we've explored sequential and parallel execution models, but an asynchronous model is one that integrates event-driven programming techniques. Event-driven programming is built on an event loop that is always checking for new occurrences. Event-driven programming operates on the basis of events. When events are executed in a loop, they control what happens and in what order. (Anon., 2024)

Characteristics of Event-Driven Programming

- ✓ **Event handlers**

An event handler is code that is designed to run in response to a specific event. Let's imagine you want to show a message when a button is pressed. In this scenario, clicking the button is both the event and the code that causes the message to be displayed.

- ✓ **Event loop**

A central loop that waits for events and forwards them to the appropriate event handlers. This loop runs continuously, waiting for an event to occur.

- ✓ **Event**

An event is a behavior that might prompt a reaction. Events include things like keystrokes, mouse clicks, sensor data input, communications from other applications, and so on.

- ✓ **Asynchronous Execution**

Event-driven programming often requires asynchronous processing, where events are handled independently from the main program flow, allowing multiple events to be handled quickly and efficiently.

- ✓ **Service Orient Processing**

Service Oriented Processing refers to background services that are triggered by events on a system.

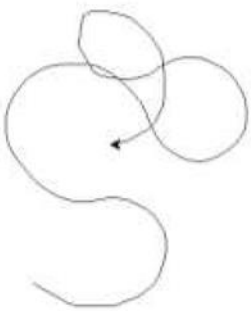
- ✓ **Time Management**

If a system responds to time, then it is controlled by time. This is best explained using an example such as an alarm clock.

- ✓ **Trigger Function.**

A trigger function acts as the glue that connects an event and an event handler together.

```
import turtle

def move_forward():
    turtle.forward(10)

def move_backward():
    turtle.backward(10)

def turn_left():
    turtle.left(10)

def turn_right():
    turtle.right(10)


# Bind events to functions
turtle.onkeypress(move_forward, "Up")
turtle.onkeypress(move_backward, "Down")
turtle.onkeypress(turn_left, "Left")
turtle.onkeypress(turn_right, "Right")
turtle.listen()

# Main event loop
while True:
    turtle.forward(0)
```



- ✓ Use an event handler to respond to user input. In this case, use the onkeypress method to bind the arrow key events to the corresponding functions.
- ✓ Use an event loop to continuously wait for events. In this case, use the listen method to start listening for events, and use a while loop to continuously move the turtle forward one unit at a time.
- ✓ Interrupt the main event loop to handle user events. When the user presses an arrow key, the corresponding event handler function is called, interrupting the main event loop.

- ✓ Multiple events can occur simultaneously. In this case, the user can press more than one arrow key at the same time to move the turtle in different directions.

- ✓ It has a non-blocking event loop. The event loop does not block the execution of other code while the program is waiting for an event to occur.

Advantages and Disadvantages of Event Driven Programming

Event-driven programming is a programming paradigm that is widely used in various software applications. This method allows a program to respond quickly to events or messages triggered by the user, such as mouse clicks or keystrokes. One of the main advantages of this method is responsiveness, allowing a program to respond immediately to user actions. Furthermore, event-driven programming promotes modularity, extensibility, and customizability of code, making it suitable for large, complex applications. It also promotes solid code organization, making the code easier to understand and maintain. However, event-driven programming also has disadvantages, including: B. Debugging challenges due to unclear control flow, complexity that can arise from handling a large number of events, and overhead that can lead to slower execution. Finally, event-driven programming can be difficult to understand and learn, especially for novice programmers, due to its complex concepts and terminology.

ACTIVITY 03

3.1 Pseudo code for Employee Apply Leave Function

```
FUNCTION Apply Leave (employee ID, leave Type, leave Date, leave Duration)
  IF NOT IsLoggedIn (employee number) THEN
    RETURN "Please login"
  END IF

  // Check remaining leaves based on leave Type
  IF leave Type = "Annual Leave" THEN
    IF remainingAnnualLeaves (employee number) = 0 THEN
      RETURN "No Annual Leaves Remaining"
    END IF
    IF leave Date < Current Date + 7 days THEN
```

```
        RETURN "Annual leaves must be applied 7 days in advance"
      END IF
    ELSE IF leave Type = "Casual Leave" THEN
      IF remainingCasualLeaves (employee number) = 0 THEN
        RETURN "No Casual Leaves Remaining"
      END IF
      IF leave Date < rosterStartDate THEN
        RETURN "Cannot apply casual leave after roster starts"
      END IF
    ELSE IF leave Type = "Short Leave" THEN
      IF remainingShortLeaves (employee number) = 0 THEN
        RETURN "No Short Leaves Remaining"
      END IF
      IF leave Duration > 1 hour 30 minutes THEN
        RETURN "Short leave duration cannot exceed 1 hour 30 minutes"
      END IF
    END IF

    // Insert leave request into the database
    INSERT INTO Leave Requests (employee number, leave Type, leave Date, and leave Duration, status)
    VALUES (employee number, leave Type, leave Date, leave Duration, "Pending Approval")

    RETURN "Leave Request Submitted"
END FUNCTION
```

Database Structure of Grifindo Lanka Toy Company

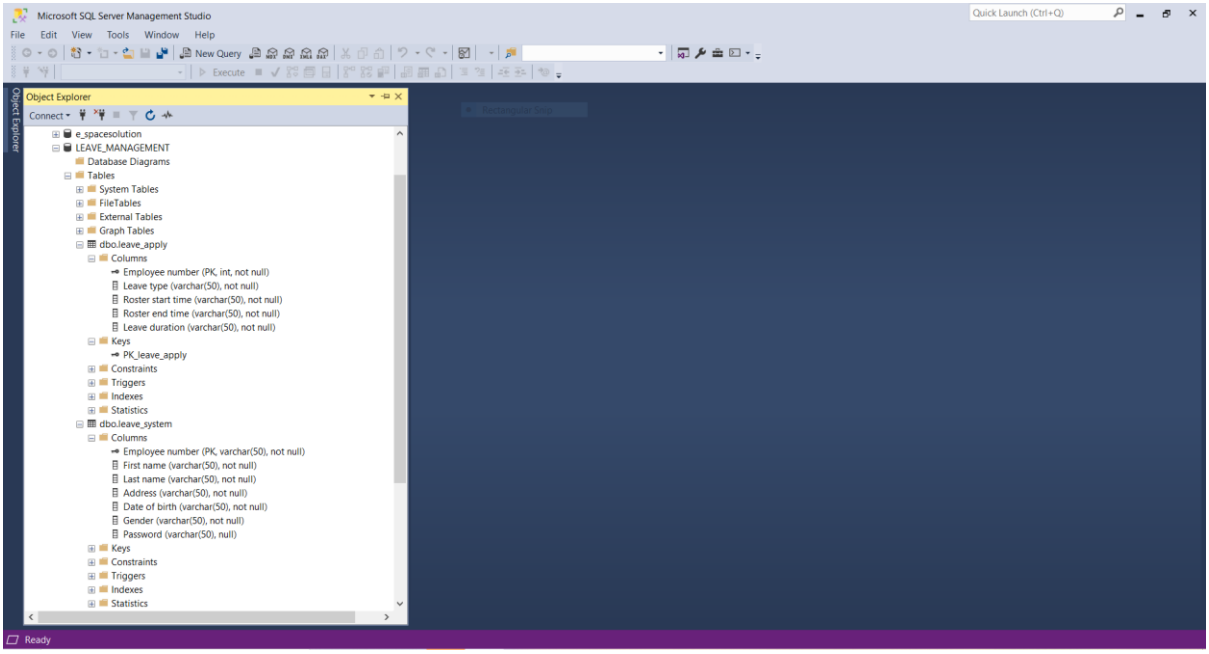
Database

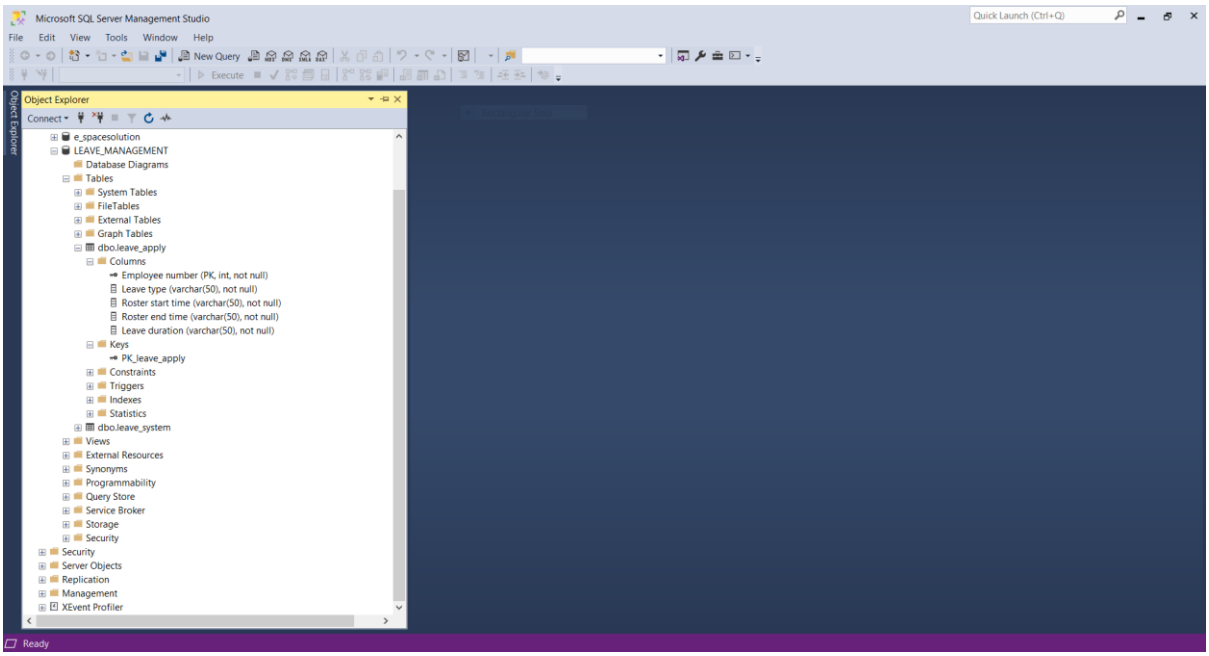
A database represents an organized compilation of structured information or data, usually stored digitally within a computer system. Typically overseen by a Database Management System (DBMS), this amalgamation of data, DBMS, and related applications is commonly known as a database system or simply a database.

Information within prevalent databases today is often arranged in tables featuring rows and columns, facilitating streamlined data processing and querying. This format allows for accessible, controlled, and organized handling of data, enabling efficient management, modification, updating, and organization. A majority of databases rely on Structured Query Language (SQL) for tasks involving data querying and writing.

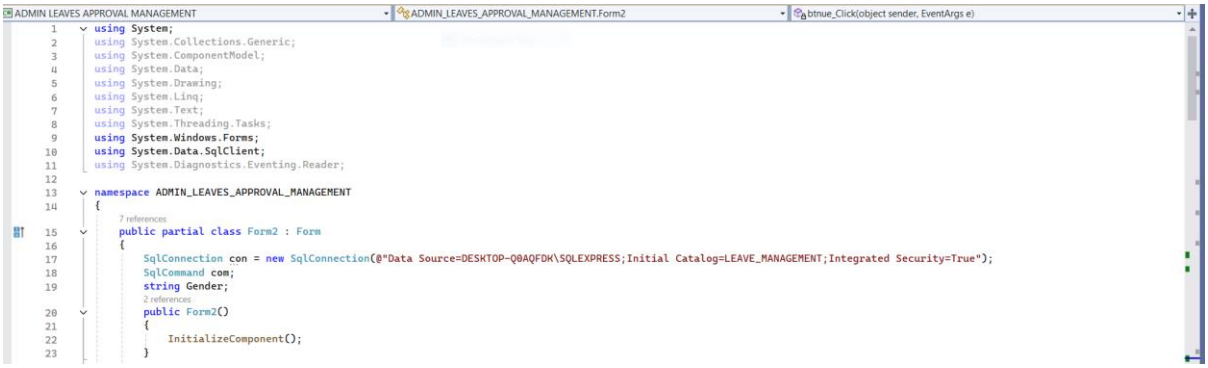
What is Structured Query Language (SQL)?

SQL, a programming language employed across nearly all relational databases, serves to interrogate, modify, and outline data, in addition to managing access control. Originating in the 1970s at IBM and significantly influenced by Oracle, SQL's evolution brought about the establishment of the SQL ANSI standard, fostering numerous extensions by corporations like IBM, Oracle, and Microsoft. Despite its enduring prevalence, emerging programming languages are now starting to emerge.





CONECTING SQL SERVER AND THE VISUAL STUDIO



Integrated Development Environment (IDE)

An Integrated Development Environment (IDE) is a software package that offers programmers with a complete development environment for software creation, testing, and deployment. It usually contains a text editor, a compiler or interpreter, a debugger, and other software development tools. The goal of an IDE is to simplify the development process by combining all of the necessary tools and functionality into a single programme. They frequently feature an easy interface that allows programmers to create and modify code, manage project files, and search the code base. IDEs frequently provide code templates, auto-completion features, and debugging tools to assist developers in producing high-quality code more quickly.

Popular IDEs include Visual Studio, Eclipse, IntelliJ IDEA, PyCharm, and Xcode.
A software program known as an integrated development environment (IDE) offers complete tools for software development to computer programmers. A few of an IDE's essential attributes are as follows:

- **Code editor:** IDEs typically have a code editor with features such as syntax highlighting, code completion, code folding, and debugging support.
- **Debugging tools:** Debugging tools allow developers to test and troubleshoot code more efficiently by setting breakpoints, analyzing variables, and stepping through code.
- Compiler and Build Tools: To generate executable programs from source code, an IDE typically includes a compiler and build tools.

- **Integration with version control systems:** Developers can manage code changes and collaborate more effectively by integrating their IDEs with version control systems such as Git, SVN, and Mercurial.
- **Project management:** IDEs aid in project management in software development by providing team collaboration tools, file organization, and project templates.
- **Code refactoring:** IDEs can automate code refactoring operations such as renaming variables and extracting methods to improve code quality and maintainability.
- **Built-in documentation:** Developers can create and manage code documentation within the IDE using the IDE's built-in documentation tools.

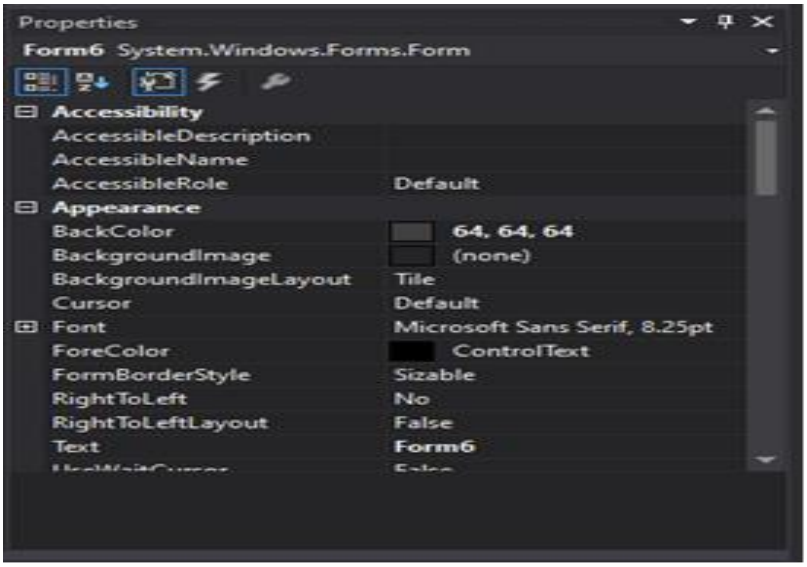
- **Support for multiple programming languages:** Modern IDEs support multiple programming languages, allowing developers to work with different languages in the same environment.
- **Plugins and extensions:** Plugins and extensions are available for IDEs that provide new features and interfaces to third-party tools and services.
- **User interface customization:** Most IDEs offer customizable user interfaces that allow developers to customize the layout, fonts, and colors to their liking

Visual Studio IDE

Visual Studio is a powerful developer tool that allows you to finish the entire programming process in one spot. This full integrated development environment (IDE) enables you to write, change, debug, build, and publish code. Visual Studio features a graphical designer, a compiler, code completion tools, source code control, extensibility, and a variety of tools to help you not just edit and debug code, but also with every step of the software development process.

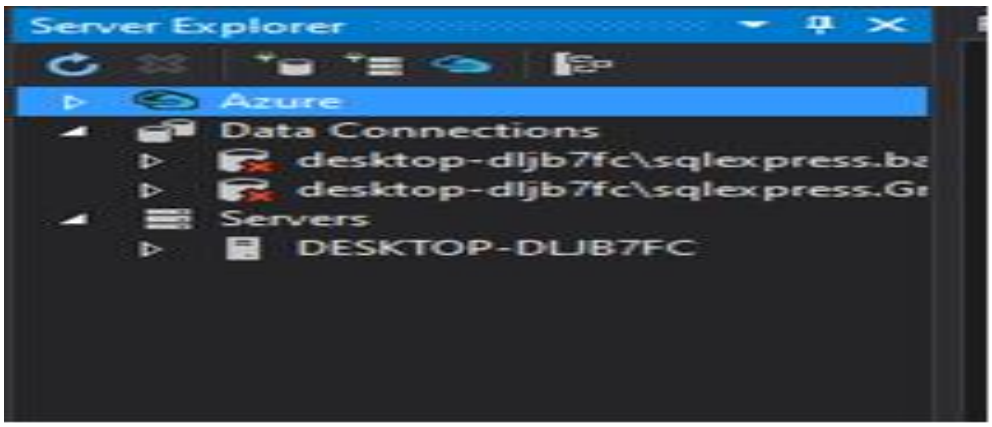
- **Properties**

In editors and designers, the Property pane is used to analyse and modify the design-time properties and events of specific objects. In addition, the Properties box allows you to modify and view file, project, and solution properties. The View menu allows you to view the Properties Window. Pressing F4 or typing "Properties" in the search bar will also send you there. The Properties window provides a variety of editing fields, each adapted to the requirements of a certain property. Edit boxes, drop-down lists, and connections to custom editor dialogue boxes are examples of editable fields.



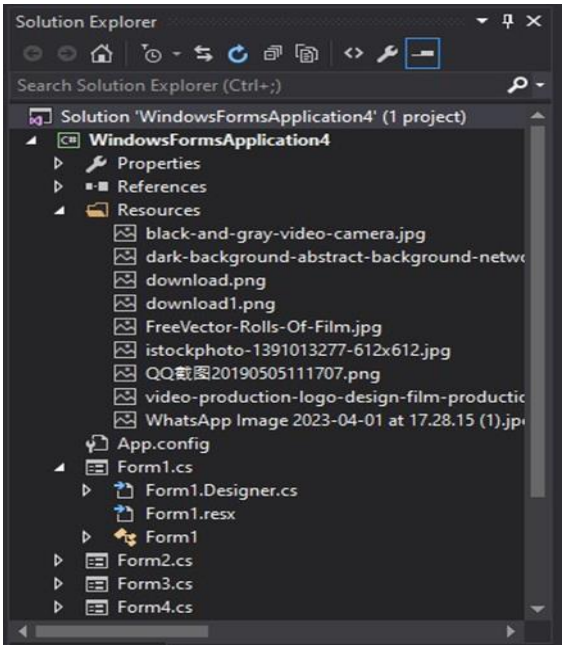
- **Server Explorer**

Server Explorer is a shortcut that allows you to access servers that are either installed on the system or linked to it. These are often database servers like SQL Server. By visiting the server, you have access to all of the databases on that specific server, from which you can then establish the connections required within your software.



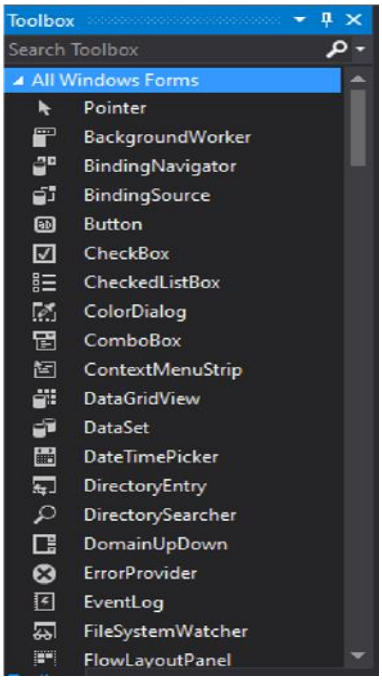
- **Solution Explorer**

The Solution Explorer window displays a list of all the projects and files they include. The window is used to navigate to files, include/exclude files from the project, and manage project references and properties.



- **Toolbox**

A Windows control is a graphical item that allows the user to interact with the computer. The controls are as diverse as the demands and aims. Because there are so many controls for varied purposes, their incorporation into an application and setting are left to the computer programmer. The Toolbox is the accessory that contains the majority of the controls used in an application:



ACTIVITY 04

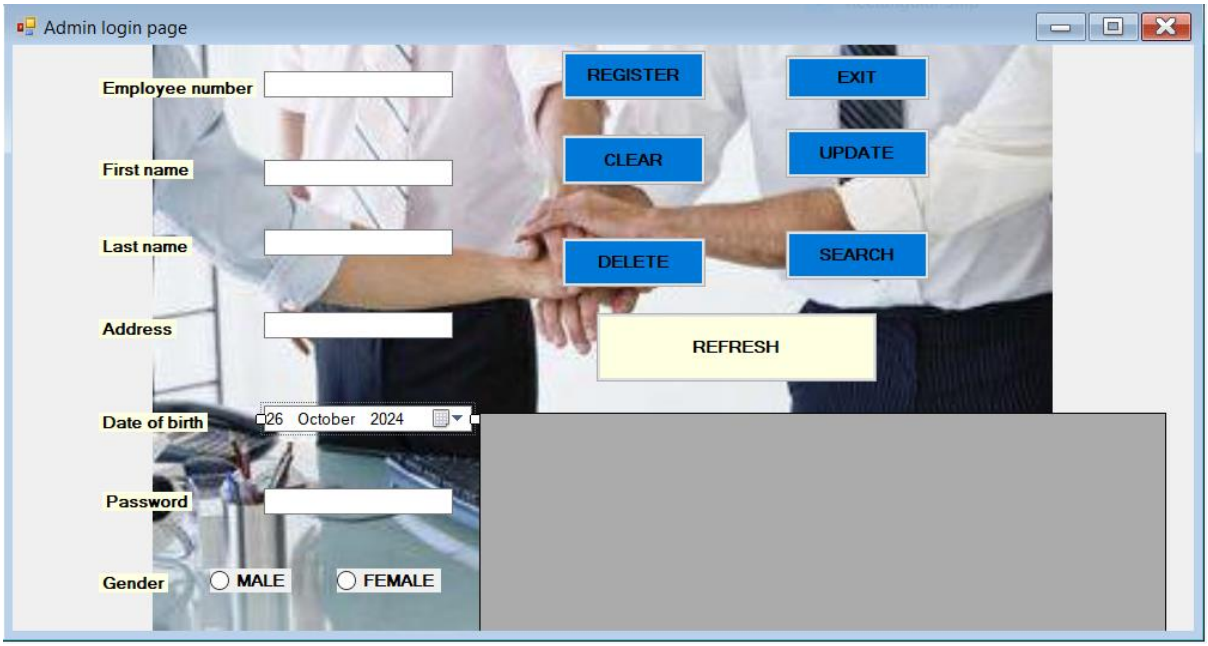
4.0 Create a simple graphical user interface (GUI) for Grifindo Lanka Toys

The Windows Forms framework in Visual Studio 2022 is widely used when developing interactive Graphical User Interfaces (GUIs) for desktop applications. A sophisticated integrated development environment (IDE), Visual Studio 2022 offers a number of tools and features to aid in the planning, creation, and testing of applications. To rapidly and easily construct Windows Forms apps, developers can use Visual Studio 2022. The pre-built controls and drag-and-drop interface enable rapid development, while the debugging and testing tools ensure the program's dependability

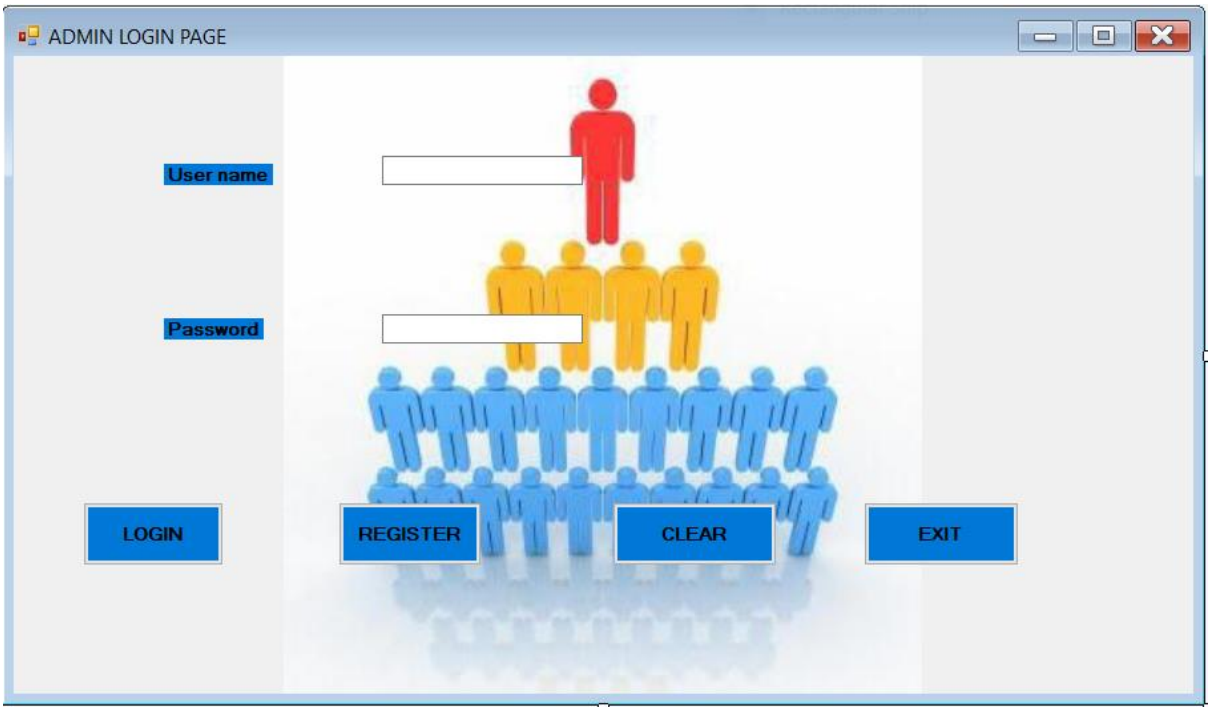
4. 1Developing the System

INTERFACES

Employee registration page with user report.



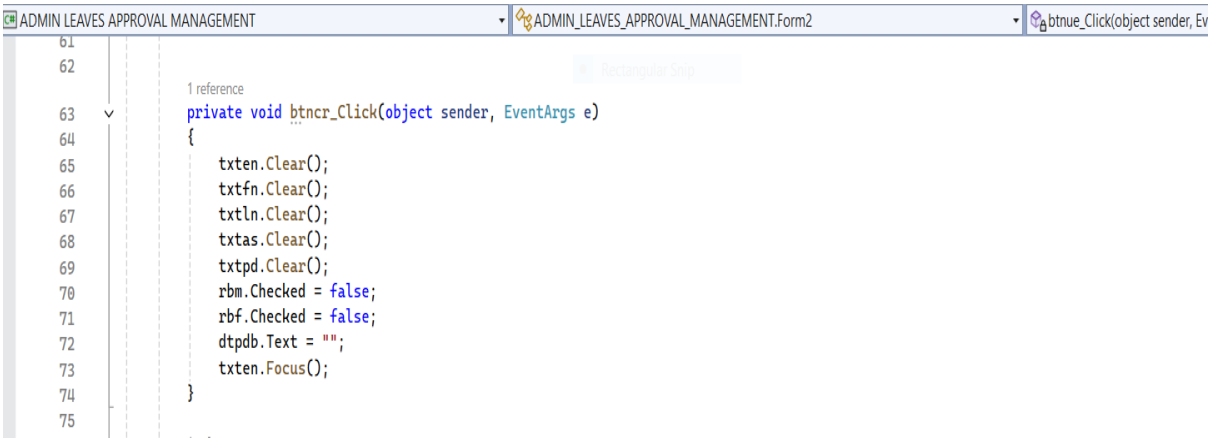
Admin login page



Apply button coding

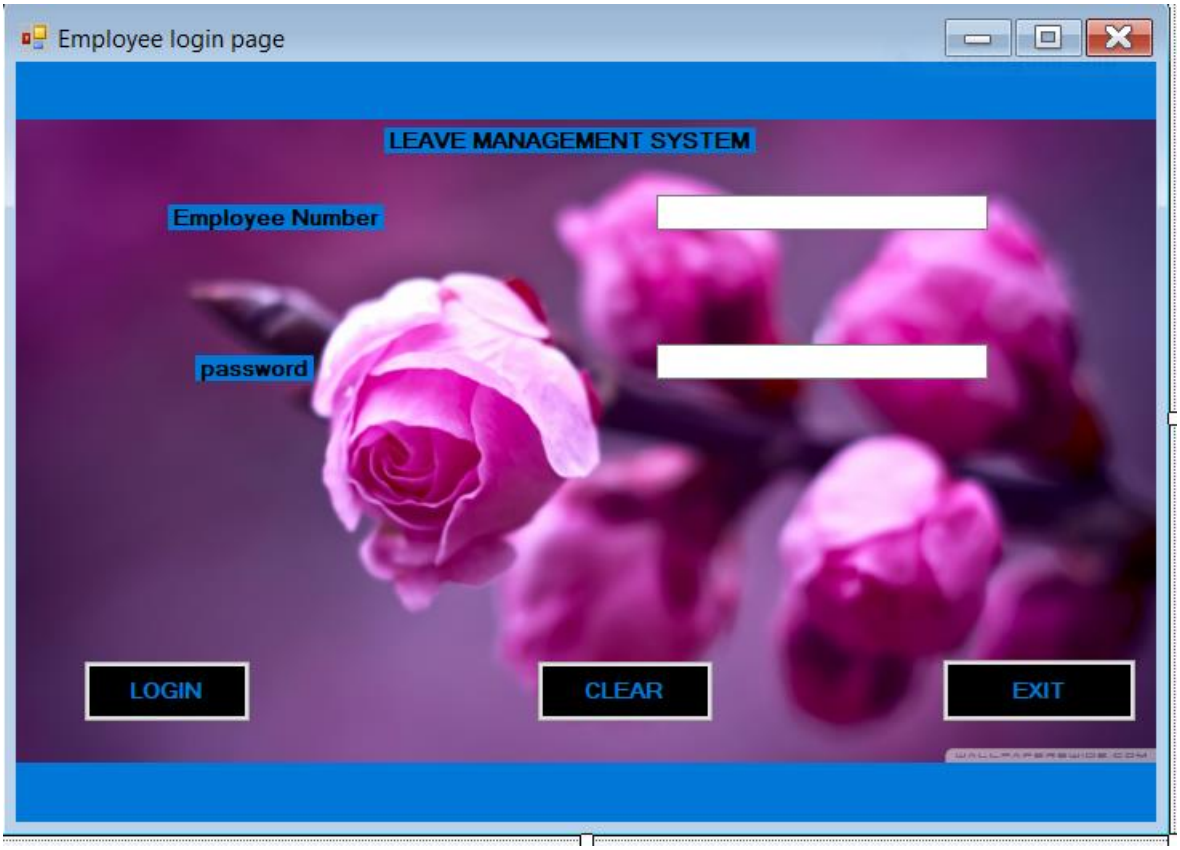
```
leaves management system 1 | leaves_management_system_1.Form3 | button1_Click(object sender, EventArgs e)

33      private void button1_Click(object sender, EventArgs e)
34      {
35          string en = txten.Text;
36          string leaveType = cmbLt.SelectedItem.ToString();
37          DateTime Rosterstarttime= dtprst.Value;
38          DateTime Rosterendtime = dtpret.Value;
39          string duration = txtld.Text;
40
41          con.Open();
42          string INSERT= "INSERT INTO leave_apply  VALUES ('"+txten.Text+"','" + cmbLt.Text + "','" + dtprst.Text + "','"+dtpret.Text+"','" + txtld.Text + "')";
43          com = new SqlCommand(INSERT, con);
44           com.ExecuteNonQuery();
45
46          con.Close();
47
48          MessageBox.Show($"Leave applied","MESSAGE",MessageBoxButtons.OK,MessageBoxIcon.Information);
49          Form4 obj = new Form4();
50          obj.Show();
51          this.Hide();
52      }
53
```

Clear button coding



Delete button coding

Employee login page

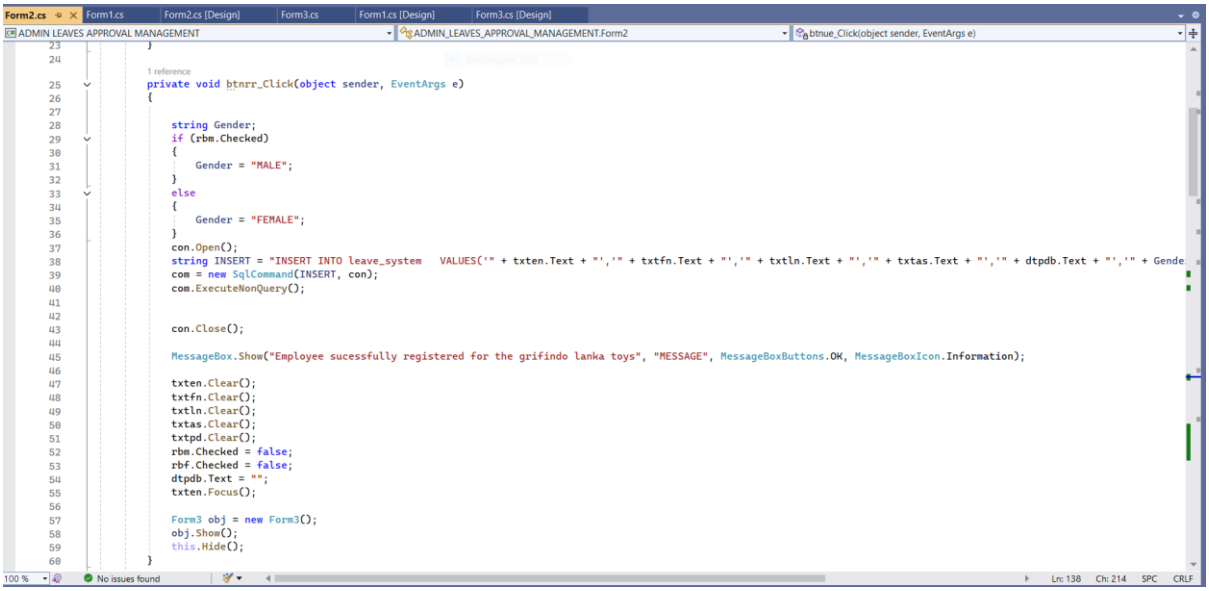


Exit button coding

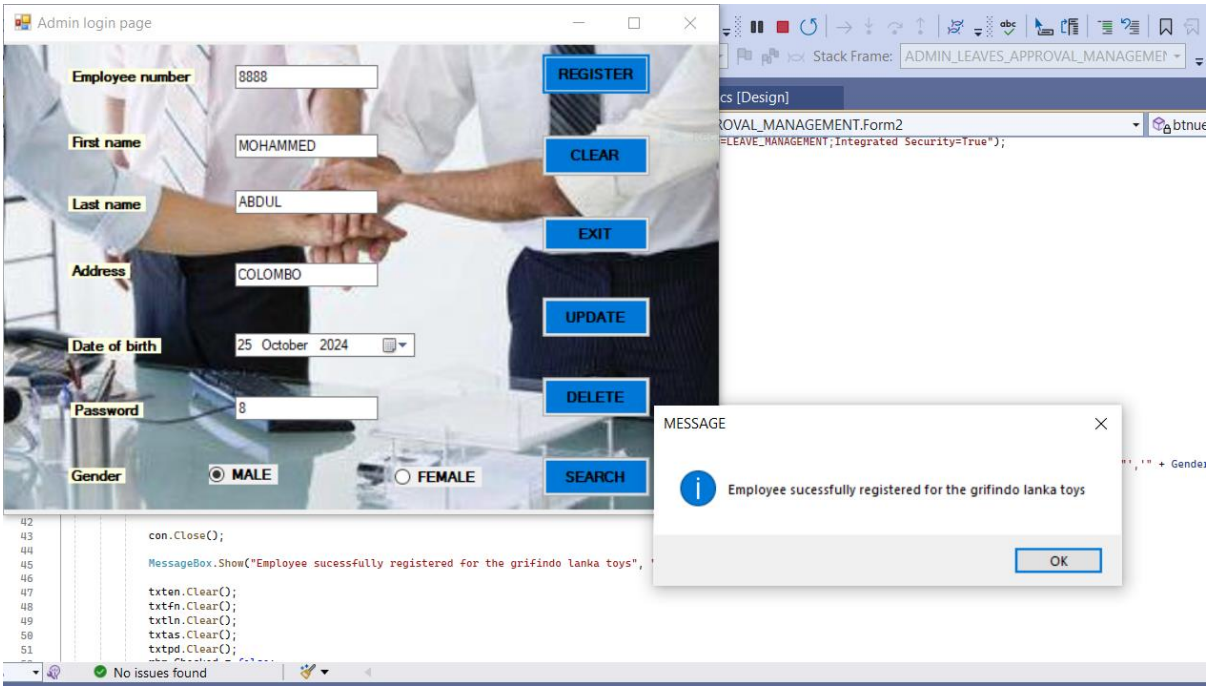
```
1 reference
private void btnet_Click(object sender, EventArgs e)
{
    var result = MessageBox.Show("Do you want to exit?", "MESSAGE", MessageBoxButtons.YesNo);
    if (result == DialogResult.Yes)
    {
        Form1 obj1 = new Form1();
        obj1.Show();
        this.Hide();
    }
    // No action needed if "No" is selected

}
```

Leave apply page

Register button coding



Registering an employee



Search button coding

```
1 reference
private void btnsh_Click(object sender, EventArgs e)
{
    try
    {
        con.Open();
       string Search= "Select * from leave_systen where Employee number ='"+txten.Text+"'";
        com = new SqlCommand(Search, con);
        SqlDataReader reader;
        reader = com.ExecuteReader();
        if (reader.Read())
        {
            txtfn.Text = reader["First name"].ToString();
            txtln.Text = reader["Last name"].ToString();
            txtas.Text = reader["Address"].ToString();
            dtpdb.Text = reader["Date of birth"].ToString();
            txtpd.Text = reader["Password"].ToString();
            Gender = reader["Gender"].ToString();
            if (Gender == "MALE")
            {
                rbm.Checked = true;
            }
            else
            {
                rbf.Checked = true;
            }
        }
        else
        {
            MessageBox.Show("There is no registration details in this Employee number", "error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }



    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);

    }
    finally
    {
        con.Close();
    }
}
```

Update button coding

```
134
        1 reference
135     private void btnue_Click(object sender, EventArgs e)
136     {
137         con.Open();
138         string update = "update leave_system set First name='" + txtfn.Text + "',Last name='" + txtln.Text + "',Address= '" + txtas.Text + "', Date of birth = '" + dtpdb.Text
139
140         com = new SqlCommand(update, con);
141         com.ExecuteNonQuery();
142
143         MessageBox.Show("The record was updated", "UPDATE", MessageBoxButtons.OK, MessageBoxIcon.Information);
144         con.Close();
145         txten.Clear();
146         txtfn.Clear();
147         txtln.Clear();
148         txtas.Clear();
149         txtpd.Clear();
150         rbm.Checked = false;
151         rbf.Checked = false;
152         dtpdb.Text = "";
153         txten.Focus();
154     }
155
```

4.2 Debugging Process

Debugging is the process of locating and correcting mistakes or bugs in the source code of any software. When software fails to perform as planned, computer programmers examine the code to find the source of the problem. They employ debugging tools to run the software in a controlled environment, inspect the code step by step, and evaluate and repair the problem.

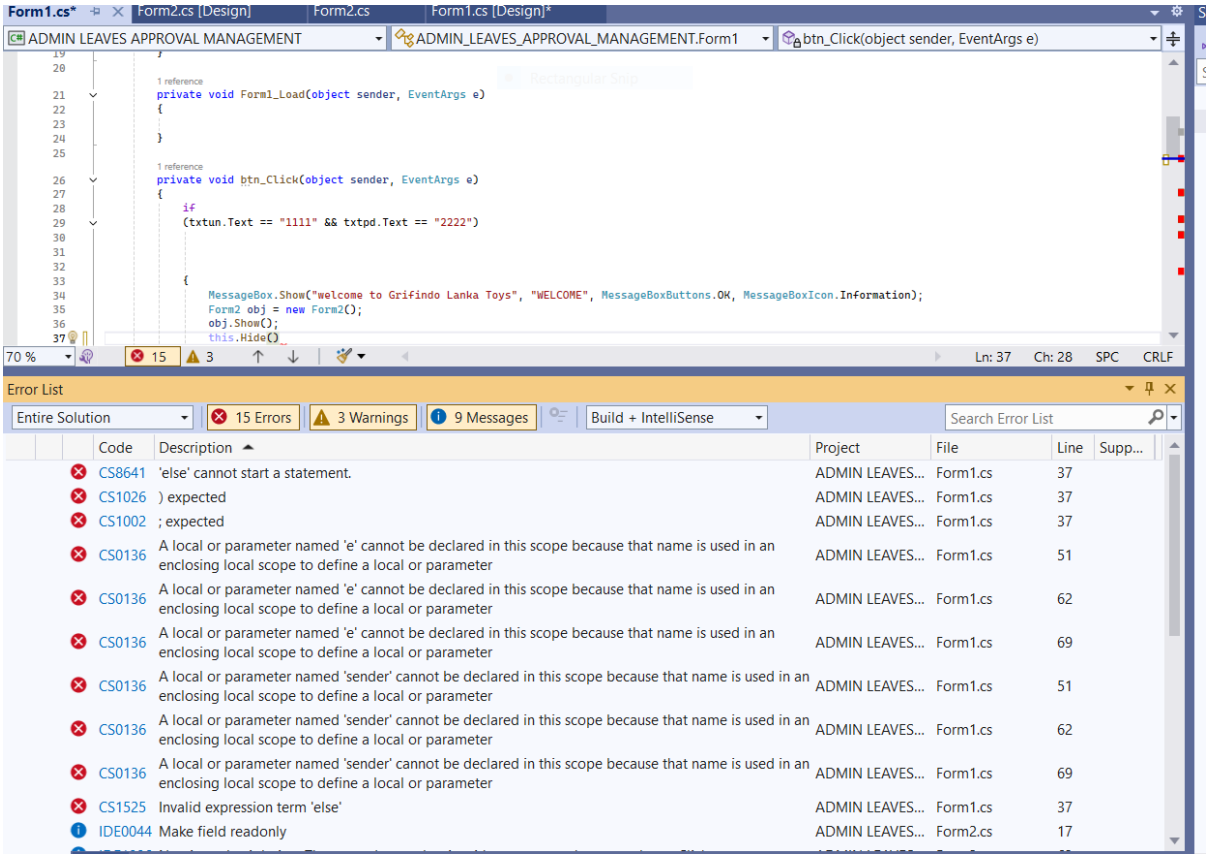
Visual Studio is a robust integrated development environment (IDE) that offers a variety of debugging options to assist developers in identifying and correcting issues in their code.

Visual Studio includes the following major debugging features:

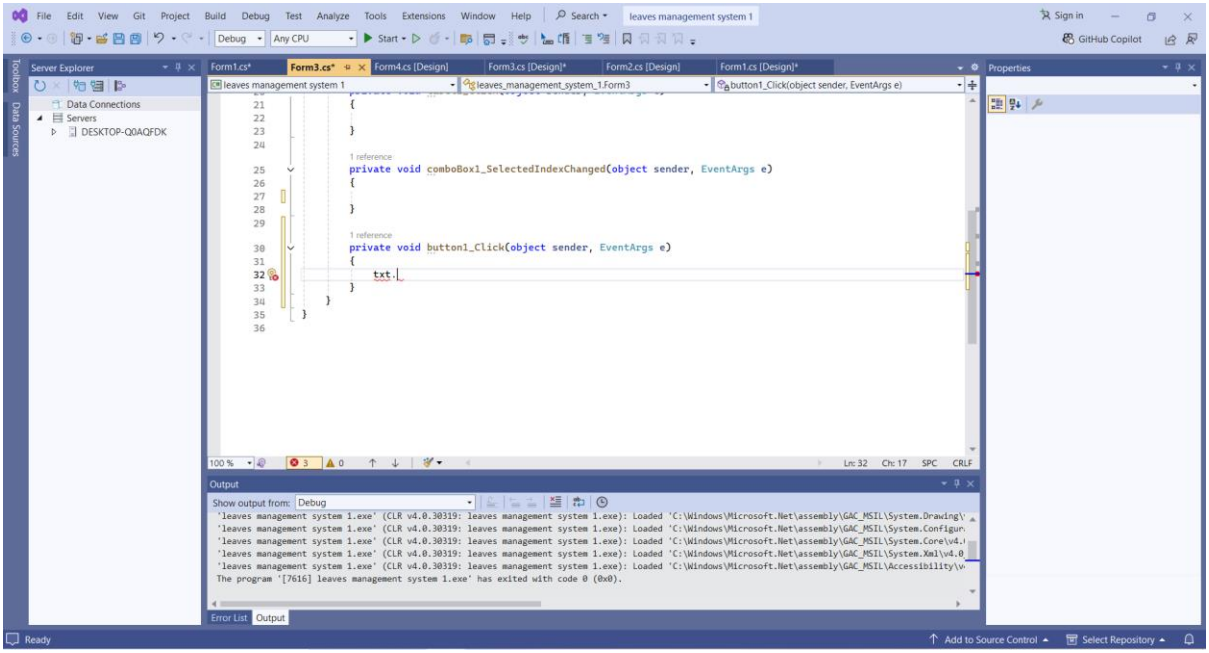
- Debugging windows
- Error list
- Breakpoints
- Try-Catch Method

Error list

Visual Studio's Error List pane displays any build, compilation, or runtime issues that occur when debugging the application. The error messages can assist developers in determining where the issue happened, what caused it, and what code needs to be updated in order to repair the error. In Visual Studio, the Error List window may be opened via the View menu.

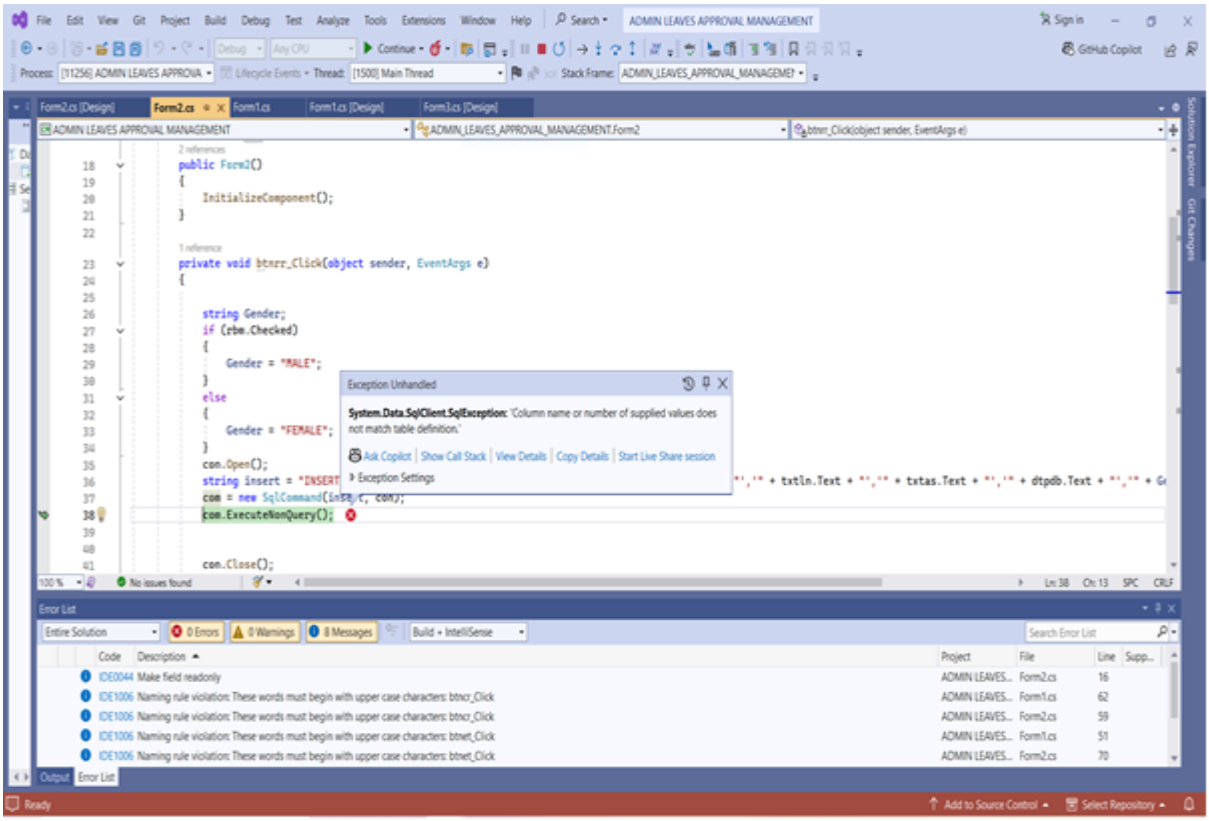


The example shown provides a list of form errors with 15 issues noted at a particular time. The error list displays the type of error and the specific code line that caused it. Developers may identify the precise location of any issue by clicking on it and then take the necessary action to fix it. This error list feature is a helpful tool that could aid programmers in swiftly identifying and fixing problems, enabling them to create efficient software.



Breakpoints

The most fundamental and essential component of reliable debugging is breakpoints. A breakpoint instructs Visual Studio to interrupt your running code so that you can check variable values, memory behavior, or the status of a branch of code.



Breakpoints also provide the added advantage of enabling code editing during debugging without requiring us to restart code execution. While debugging, we might additionally keep track of and modify variable values. whereas debugging the code, Add Watch enables us to continuously monitor the value of a variable whereas Quick Watch only shows the variable's current value.

Try-Catch Method

This is a programming construct that allows programmers to discover and address mistakes during code execution. This function may be used by developers to build code that attempts to do a certain operation and, if it fails, gently manages the issue rather than crashing the program. Visual Studio developers may utilize the try-catch technique to collect and manage errors that occur while debugging.

```
91          try
92          {
93              con.Open();
94              string Search= "SELECT * FROM leave_system WHERE Employee number ='"+txten.Text+"'";
95              com = new SqlCommand(Search, con);
96              SqlDataReader dr;
97              dr = com.ExecuteReader();
98              if (dr.Read())
99              {
100                 txtfn.Text = dr["First name"].ToString();
101                 txtln.Text = dr["Last name"].ToString();
102                 txtas.Text = dr["Address"].ToString();
103                 dtpdb.Text = dr["Date of birth"].ToString();
104                 txtpd.Text = dr["Password"].ToString();
105                 Gender = dr["Gender"].ToString();
106                 if (Gender == "MALE")
107                 {
108                     rbm.Checked = true;
109                 }
110                 else
111                 {
112                     rbf.Checked = true;
113                 }
114             }
115             else
116             {
117                 MessageBox.Show("There is no registration details in this Employee number", "error", Messa
118             }
119
120
121
122
123         }
124         catch(Exception ex)
125         {
126             MessageBox.Show(ex.Message);
127
128         }
```

The program that needs to execute in the try block. If an exception arises while this code is running, the program will jump to the "catch" block, where you may deal with the problem. You can write code to notify the user of an error or report an exception in the "catch" block.

4.3.Coding Standards

Coding standards are guidelines and best practices for writing reliable, consistent code. Consider coding standards as guidelines, tactics, and best practices for creating code that is clearer, easier to read, and free of errors. They give programmers a consistent framework with which to write intricate, highly useful code.

The Goal of Coding Standards

- **Improve Code Quality**: Coding standards guarantee that code is written consistently, readably, and in a way that can be maintained. This makes the code easier for engineers to comprehend and work with, resulting in higher-quality software.
- **Improve Efficiency**: By adhering to code standards, developers may save time by avoiding frequent errors and using tried-and-true solutions.
- **Facilitate Collaboration**: It establishes a standard language that all developers can understand, allowing teams to successfully cooperate, exchange code, and communicate.
- **Ensure Compatibility**: It guarantees that the code is interoperable with many platforms, browsers, and OS-device combinations.
- **Reduce Maintenance Costs**: By adhering to established standards, developers may minimize the introduction of new defects and perform code changes more quickly and easily.

Standards for Coding

- **Global Data Types**: These standards define which data types can and cannot be declared global.
- **Standard headers for several modules**: Each module's header should follow a consistent structure and content for better comprehension and code maintenance. The header format must have the following items, which are often used by businesses.

  - The Module's Name
  - The Module's Creation Date
  - Module Modification History Author
  - Module synopsis describing what the module performs
  - The module supports several functions, as well as their input and output parameters.
  - The module accesses or modifies global variables.
  -

- **Indentation**: Proper indentation is essential for making the code more understandable. For intelligible code, programmers should make good use of white space. Here are some instances of spacing rules.

  - A space is necessary after a comma separates two function arguments.
  - Each nested block must be properly spaced and indented.
  - Each program block should have suitable indentation at the beginning and finish.
  - All brackets should start on a new line, as should the code that follows them.

- **Avoid using GOTO statements**: The GOTO statement causes the program to be unstructured, which reduces program understanding and makes debugging more difficult.

- **The code should be correct. Documents**: Clear comments should be included in the code to make it easy to understand. When there are comments between the statements, the code is simpler to understand.

Advantages of coding standards

1. Reduced Failure Rate
2. Optimize development time
3. Easier Debugging
4. Affordable
5. Conclusion

The importance of coding standards for a person

Individual coding standards contribute to the program's efficiency and production. It aids in the development of code in less time and at the same time. We can create programs that are effective.

The importance of coding standards for a group

Without good code standards, readability and maintainability suffer. Programmers are unable to create effective and efficient systems. The importance of coding standards for a group

Was a team of individuals is needed to create a system, including a developer, tester, designer, modifiers, and managers. These individuals collaborate with the same people to design a system, therefore if we maintain correct coding standards, they can also grasp cods, and if there are flaws or difficulties, anybody can address them. Following code standards is thus more beneficial.

Gant Chart

Gant Chart												
Month	July				August				September			
Week	1	2	3	4	1	2	3	4	1	2	3	4
Activity 1												
Activity 2												
Activity 3												
Activity 4												

References

Anon., 2022. *factorial.* [Online] Available at: https://www.cuemath.com/numbers/factorial/ [Accessed 09 07 2024].

Anon., 2024. *Event-Driven Programming.* [Online] Available at: https://www.tutorialspoint.com/concurrency_in_python/concurrency_in_python_eventdriven_programming.htm [Accessed 12 07 2024].

bhumika_rani, 2024. *Programming Paradigms.* [Online] Available at: https://www.geeksforgeeks.org/user/bhumika_rani/contributions/?itm_source=geeksforgeeks&itm_medium=article_author&itm_campaign=auth_user [Accessed 11 7 2024].

Gillis, A. S., 2024. *object-oriented programming (OOP).* [Online] Available at: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP [Accessed 11 07 2024].

Sheldon, R., 2022. *Fibonacci sequence.* [Online] Available at: https://www.techtarget.com/whatis/definition/Fibonacci-sequence [Accessed 09 07 2024].

Team, I. E., 2024. *Top down approach.* [Online] Available at: https://uk.indeed.com/career-advice/career-development/top-down-approach [Accessed 11 07 2024].

Upadhyay, S., 2024. *An Algorithm.* [Online] Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-an-algorithm#what_is_an_algorithm [Accessed 08 07 2024].

Upadhyay, S., 2024. *An Algorithm.* [Online] Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-an-algorithm#what_is_an_algorithm [Accessed 07 07 2024].

Upadhyay, S., 2024. *An Algorithm.* [Online] Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-an-algorithm#what_is_an_algorithm [Accessed 08 07 2024].