**Higher Nationals**

Internal verification of assessment decisions – BTEC (RQF)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **INTERNAL VERIFICATION – ASSESSMENT DECISIONS** | | | | | | |
| **Programme title** | Higher National Diploma in Computing | | | | | |
| **Assessor** |  | | **Internal Verifier** | |  | |
| **Unit(s)** |  | | | | | |
| **Assignment title** |  | | | | | |
| **Student’s name** |  | | | | | |
| **List which assessment criteria the Assessor has awarded.** | **Pass** | | **Merit** | **Distinction** | | |
|  | |  |  | | |
| **INTERNAL VERIFIER CHECKLIST** | | | | | | |
| **Do the assessment criteria awarded match those shown in the assignment brief?** | | Y/N |  | | | |
| **Is the Pass/Merit/Distinction grade awarded justified by the assessor’s comments on the student work?** | | Y/N |  | | | |
| **Has the work been assessed accurately?** | | Y/N |  | | | |
| **Is the feedback to the student:**  Give details:   * Constructive? * Linked to relevant assessment criteria? * Identifying opportunities for improved performance? * Agreeing actions? | | Y/N Y/N  Y/N  Y/N |  | | | |
| **Does the assessment decision need amending?** | | Y/N |  | | | |
| **Assessor signature** | |  | | | **Date** |  |
| **Internal Verifier signature** | |  | | | **Date** |  |
| **Programme Leader signature** (if required) | |  | | | **Date** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Confirm action completed** | | | |
| **Remedial action taken**  Give details: |  | | |
| **Assessor signature** |  | **Date** |  |
| **Internal Verifier signature** |  | **Date** |  |
| **Programme Leader signature** (if required) |  | **Date** |  |

Higher Nationals - Summative Assignment Feedback Form

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name/ID** |  | | |
| **Unit Title** |  | | |
| **Assignment Number** |  | **Assessor** |  |
| **Submission Date** |  | **Date Received 1st submission** |  |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Assessor Feedback:**   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **LO1. Define basic algorithms to carry out an operation and outline the process of programming an application.** | | | | | | | | **Pass, Merit & Distinction Descripts** | **P1** | **M1** | D1 |  |  |  | | **LO2. Explain the characteristics of procedural, object-orientated and event-driven programming, conduct an analysis of a suitable Integrated Development Environment (IDE).** | | | | | | | | **Pass, Merit & Distinction Descripts** | **P2** | **M2** | **D2** |  |  |  | | **LO3. Implement basic algorithms in code using an IDE.** | | | | | | | | **Pass, Merit & Distinction Descripts** | **P3** | **M3** | **D3** |  |  |  | | **LO4. Determine the debugging process and explain the importance of a coding standard.** | | | | | | | | **Pass, Merit & Distinction Descripts** | **P4** | **P5** | **M4** | **D4** |  |  | |  | | | | |
| **Grade:** | **Assessor Signature:** | | **Date:** |
| **Resubmission Feedback:** | | | |
| **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.

**Assignment Feedback**

|  |  |  |  |
| --- | --- | --- | --- |
| **Formative Feedback: Assessor to Student** | | | |
| **Action Plan** | | | |
| **Summative feedback** | | | |
| **Feedback: Student to Assessor** | | | |
| **Assessor signature** |  | **Date** |  |
| **Student signature** |  | **Date** |  |

**Pearson Higher Nationals in**

**Computing**

Unit 01: Programming

Assignment 01

**General Guidelines**

1. A Cover page or title page – You should always attach a title page to your assignment. Use previous page as your cover sheet and make sure all the details are accurately filled.
2. Attach this brief as the first section of your assignment.
3. All the assignments should be prepared using a word processing software.
4. All the assignments should be printed on A4 sized papers. Use single side printing.
5. Allow 1” for top, bottom , right margins and 1.25” for the left margin of each page.

**Word Processing Rules**

1. The font size should be **12** point, and should be in the style of **Time New Roman**.
2. **Use 1.5 line spacing**. Left justify all paragraphs.
3. Ensure that all the headings are consistent in terms of the font size and font style.
4. 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.
5. Use word processing application spell check and grammar check function to help editing your assignment.

**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. Carefully check the hand in date and the instructions given in the assignment. Late submissions will not be accepted.
3. Ensure that you give yourself enough time to complete the assignment by the due date.
4. Excuses of any nature will not be accepted for failure to hand in the work on time.
5. You must take responsibility for managing your own time effectively.
6. 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.
7. Failure to achieve at least PASS criteria will result in a REFERRAL grade .
8. Non-submission of work without valid reasons will lead to an automatic RE FERRAL. You will then be asked to complete an alternative assignment.
9. 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 in-text citation and a reference list.
10. 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

**Student Declaration**

I hereby, declare that I know what plagiarism entails, namely to use another’s work and to present it as my own without attributing the sources in the correct way. I further understand what it means to copy another’s work.

1. I know that plagiarism is a punishable offence because it constitutes theft.
2. I understand the plagiarism and copying policy of the Edexcel UK.
3. I know what the consequences will be if I plagiaries or copy another’s work in any of the assignments for this program.
4. I declare therefore that all work presented by me for every aspects of my program, will be my own, and where I have made use of another’s work, I will attribute the source in the correct way.
5. I acknowledge that the attachment of this document signed or not, constitutes a binding agreement between myself and Edexcel UK.
6. I understand that my assignment will not be considered as submitted if this document is not attached to the attached.

**Student’s Signature: Date:**

**(*Provide E-mail ID*) (*Provide Submission Date*)**

**Higher National Diploma in Computing**

Assignment Brief

|  |  |  |
| --- | --- | --- |
| Student Name /ID Number | |  |
| **Unit Number and Title** | | **Unit 01: Programming** |
| Academic Year | | 2021/22 |
| Unit Tutor | |  |
| **Assignment Title** | | **Design &Implement a GUI based system using a suitable Integrated Development Environment** |
| Issue Date | |  |
| Submission Date | |  |
| IV Name & Date | |  |
| **Submission Format** | | |
| **This submission will have 3 components**   1. Written Report   This submission is in the form of an individual written report. This should be written in a concise, formal business style using single spacing and font size 12. You are required to make use of headings, paragraphs and subsections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system. Please also provide a bibliography using the Harvard referencing system. (**The recommended word count is 1,500–2,000 words for the report excluding annexures)**   1. Implemented System (Software)   The student should submit a GUI based system developed using an IDE. The system should connect with a backend database and should have at least 5 different forms and suitable functionality including insert, edit and delete of main entities and transaction processing.   1. Presentation   With the submitted system student should do a presentation to demonstrate the system that was developed. Time allocated is 10 to 15 min. Student may use 5 to 10 PowerPoint slides while doing the presentation, but live demonstration of the system is required. Evaluator will also check the ability to modify and debug the system using the IDE. | | |
| **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, conduct an analysis of a suitable Integrated Development Environment (IDE).  **LO3.** Implement basic algorithms in code using an IDE.  **LO4.** Determine the debugging process and explain the importance of a coding standard | |

**Grading Rubric**

|  |  |  |
| --- | --- | --- |
| **Grading Criteria** | **Achieved** | **Feedback** |
| **LO1 Define basic algorithms to carry out an operation and outline the process of programming an application.** |  |  |
| **P1**Provide a definition of what an algorithm is and outline the process in building an application. |  |  |
| **M1**Determine the steps taken from writing code to execution. |  |  |
| **D1** Evaluate the implementation of an algorithm in a suitable language. Evaluate the relationship between the written algorithm and the code variant |  |  |
| **LO2 Explain the characteristics of procedural, objectorientated and event-driven programming, conduct an analysis of a suitable Integrated Development Environment (IDE)** |  |  |
| **P2**Give explanations of what procedural, objectorientated, and eventdriven paradigms are; their characteristics and the relationship between them. |  |  |
| **M2** Compare and contrast 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 which implements the programming paradigms, in terms of the code structure and characteristics. |  |  |
| **LO3Implement basic algorithms in code using an IDE.** |  |  |
| **P3**Write a program that implements an algorithm using an IDE. |  |  |
| **M3**Use the IDE to manage the development process of the program. |  |  |
| **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** |  |  |
| **P4**Explain the debugging process and explain the debugging facilities available in the IDE. |  |  |
| **P5**Outline the coding standard you have used in your code. |  |  |
| **M4**Evaluate how the debugging process can be used to help develop more secure, robust applications. |  |  |
| **D4** Critically evaluate why a coding standard is necessary in a team as well as for the individual. |  |  |

**Table of Content**

[**Acknowledgement** 17](#_Toc122938694)

[**Introduction** 18](#_Toc122938695)

[1. **Define basic algorithms to carry out an operation and outline the process of programming an application**. 19](#_Toc122938696)

[1.1 Define what an algorithm 19](#_Toc122938697)

[1.2 The characteristics of a good algorithm 20](#_Toc122938698)

[1.3 Pseudocode for Fibonacci series 21](#_Toc122938699)

[1.4 Dry run of Fibonacci series 23](#_Toc122938700)

[1.5 Python program for Fibonacci series 24](#_Toc122938701)

[1.6 Pseudocode for Factorial value 25](#_Toc122938702)

[1.7 Dry run of factorial number 26](#_Toc122938703)

[1.8 Python program for factorial number 27](#_Toc122938704)

[1.9 Big-O notation 27](#_Toc122938705)

[1.10 Big-O notation's role in evaluating efficiencies of algorithms 28](#_Toc122938706)

[1.11 Evaluate the efficiency 28](#_Toc122938707)

[2. **Explain the characteristics of procedural, object-orientated and event-driven programming, conduct an analysis of a suitable Integrated Development Environment (IDE)**. 30](#_Toc122938708)

[2.1 Define Programming Paradigm 30](#_Toc122938709)

[2.2 Main characteristics of Procedural Programming Paradigm 31](#_Toc122938710)

[2.3 Main characteristics of Object-oriented Programming Paradigm 31](#_Toc122938711)

[2.4 Main characteristics of Event-driven Programming Paradigm 32](#_Toc122938712)

[2.5 Code snippet for procedural programming 33](#_Toc122938713)

[2.6 Code snippet for object-oriented programming 33](#_Toc122938714)

[2.7 Code snippet for event-driven programming 34](#_Toc122938715)

[2.8 Evaluate based on structure and the unique characteristics programming paradigms 34](#_Toc122938716)

[3. **Implement basic algorithms in code using an IDE**. 36](#_Toc122938717)

[3.1 Write suable algorithms for vehicle tariff calculation for rents and hires 36](#_Toc122938718)

[3.2 Use the visual studio IDE (using C#.net) to Implement the above algorithms and design the suitable database structure 39](#_Toc122938719)

[3.3 Implementation of Database Structure 43](#_Toc122938720)

[3.4 Analysis on the feature of an integrated development environment (IDE) 45](#_Toc122938721)

[3.5 Evaluate the use of the Visual Studio IDE for your application development contrasted with not using an IDE 47](#_Toc122938722)

[4. **Determine the debugging process and explain the importance of a coding standard** 49](#_Toc122938723)

[4.1 Design and build a small system to calculate vehicle hire amounts and record them in a database for customer billing and management reporting for Ayubo drive. 49](#_Toc122938724)

[4.2 Debugging process 53](#_Toc122938725)

[4.3 Outline the coding standards you have used in your application development 58](#_Toc122938726)

[**References** 61](#_Toc122938727)

**List of Figures**

[Fig.1.1 Pseudocode of Fibonacci series 23](#_Toc122938829)

[Fig. 1.2 Pseudocode of Factorial numbers 26](#_Toc122938830)

[Fig. 3.1 Rent calculation flow chart 37](#_Toc122938831)

[Fig. 3.2 Day tour – hire calculation flow chart 38](#_Toc122938832)

[Fig. 3.3 Long tour – hire calculation flow chart 40](#_Toc122938833)

[Fig. 3.4 Rent Calculation GUI 41](#_Toc122938834)

[Fig. 3.5 Day tour – hire calculation GUI 42](#_Toc122938835)

[Fig. 3.6 Long Tour – hire calculation GUI 44](#_Toc122938836)

[Fig. 3.7 Database structure GUI 44](#_Toc122938837)

[Fig. 3.8 Tabular format view of Day tour from database 45](#_Toc122938838)

[Fig. 3.9 Tabular format view of vehicle registration from database 45](#_Toc122938839)

[Fig. 3.10 Visual Studio IDE features 47](#_Toc122938840)

[Fig. 4.1 Login view GUI 51](#_Toc122938841)

[Fig. 4.2 Main menu view GUI 52](#_Toc122938842)

[Fig. 4.3 Vehicle management GUI 54](#_Toc122938843)

[Fig. 4.4 Breakpoint usage 56](#_Toc122938844)

[Fig. 4.5 Debugging process 56](#_Toc122938845)

[Fig. 4.6 Diagnostic tools 57](#_Toc122938846)

[Fig. 4.7 Naming conventions 60](#_Toc122938847)

[Fig. 4.8 Usage of comments 60](#_Toc122938848)

# Acknowledgement

The success and final outcome of this assignment required a lot of guidance and support from many people and I feel fortunate to have had the opportunity to work with such knowledgeable and skilled individuals, especially during the hands-on sessions. I also want to express my appreciation to the ESoft academic staff for helping us throughout this process.

# Introduction

Computer programming involves designing and creating computer programs. It encompasses a wide range of tasks and techniques and is a vast field that includes various ways to earn money, such as selling software, building the next big social media platform, or developing decentralized applications using blockchain technology. This course will cover basic to intermediate programming concepts in a thorough manner to improve understanding.

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

## 1.1 Define what an algorithm

An algorithm is a set of steps or instructions used to solve a problem or perform a task. It can be implemented in hardware or software and is used in a variety of fields, including mathematics, computer science, and data processing. Algorithms can be used for simple tasks like sorting numbers or more complex ones like recommending content on social media. They take an initial input and set of instructions and produce an output when the computation is carried out. They are an essential tool for any programmer because they provide a step-by-step approach for solving problems and can be implemented in software or used to guide the design of hardware. (Ekunwe, 2022)

There are many reasons why algorithms are important:

* Efficiency: Algorithms help to solve problems in a systematic and efficient manner. They can be used to optimize processes and reduce the amount of resources required to solve a problem.
* Accuracy: Algorithms can help to ensure that a problem is solved correctly, consistently, and without errors.
* Reusability: Algorithms can be implemented and used in multiple different contexts, which makes them very useful for solving similar problems in different settings.
* Automation: Algorithms can be used to automate tasks and processes, which can save time and effort.
* Debugging: Algorithms can help to identify and fix problems in software or hardware designs.

Overall, algorithms are a crucial component of computer science and are used to solve a wide variety of problems in many different fields.

## 1.2 The characteristics of a good algorithm

The characteristics of a good algorithm are as follows.

* Correctness: A good algorithm should produce the correct results for all inputs.
* Finiteness: A good algorithm should terminate after a finite number of steps.
* Input-Output clarity: The input and output of a good algorithm should be clearly defined and easy to understand.
* Generality: A good algorithm should be applicable to a wide range of inputs and should not be specific to a particular input type.
* Efficacy: A good algorithm should be efficient in terms of time and space complexity. It should be able to solve the problem within a reasonable amount of time and using a reasonable amount of memory or storage space. (Jawady, 2019)
* Simplicity: A good algorithm should be simple and easy to understand, with a clear and concise structure.
* Robustness: A good algorithm should be robust and able to handle unexpected or invalid input gracefully.
* Scalability: A good algorithm should be able to handle large input sizes without a significant increase in computational time or space requirements.

## 1.3 Pseudocode for Fibonacci series

*procedure fibonacci : fib\_num*

*IF fib\_num less than 1*

*DISPLAY 0*

*IF fib\_num equals to 1*

*DISPLAY 1*

*IF fib\_num equals to 2*

*DISPLAY 1, 1*

*IF fib\_num greater than 2*

*Pre = 1,*

*Post = 1,*

*DISPLAY Pre, Post*

*FOR 0 to fib\_num-2*

*Fib = Pre + Post*

*DISPLAY Fib*

*Pre = Post*

*Post = Fib*

*END FOR*

*END IF*

*end procedure*

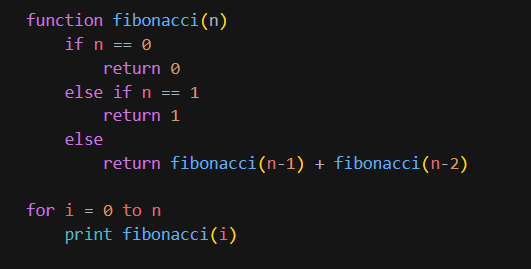


Fig.1.1 Pseudocode of Fibonacci series

This pseudocode defines a function called **fibonacci** that takes a single argument **n** and returns the **n**th number in the Fibonacci series. The Fibonacci series is a sequence of numbers in which each number is the sum of the previous two numbers, starting with 0 and 1. The function uses a recursive approach to compute the **n**th number in the series by calling itself with **n-1** and **n-2** as arguments.

The pseudocode also includes a loop that iterates from 0 to **n** and prints the **i** th number in the Fibonacci series by calling the **fibonacci** function with **i** as the argument. This will generate the first **n** numbers in the Fibonacci series.

## 1.4 Dry run of Fibonacci series

The Fibonacci series is a series of numbers in which each number is the sum of the two preceding ones, usually starting with 0 and 1.

The series for the number 6 would be: 0, 1, 1, 2, 3, 5

• To generate this series, we start by setting the first two numbers to 0 and 1.

• The third number is calculated by adding the first two numbers, which gives us 1.

• The fourth number is calculated by adding the second and third numbers, which gives us 2.

• The fifth number is calculated by adding the third and fourth numbers, which gives us 3.

• The sixth and final number is calculated by adding the fourth and fifth numbers, which gives us 5.

So, the Fibonacci series for the number 6 is: 0, 1, 1, 2, 3, 5.

## 1.5 Python program for Fibonacci series

*def fibonacci(n):*

*# Initialize the first two numbers of the series*

*a, b = 0, 1*

*# Check if the input is valid (must be a positive integer)*

*if n < 0:*

*print("Invalid input")*

*elif n == 0:*

*print("The Fibonacci series for 0 is:")*

*print(a)*

*elif n == 1:*

*print("The Fibonacci series for 1 is:")*

*print(b)*

*else:*

*print("The Fibonacci series for", n, "is:")*

*# Generate the rest of the series*

*for i in range(2, n+1):*

*c = a + b*

*a = b*

*b = c*

*print(c, end=", ")*

*fibonacci(6)*

*fibonacci(10)*

*fibonacci(15)*

## 1.6 Pseudocode for Factorial value

*START*

*Read number*

*Fact = 1*

*i = 1*

*WHILE i<=number*

*Fact=Fact\*i*

*i=i+1*

*ENDWHILE*

*WRITE Fact*

*STOP*

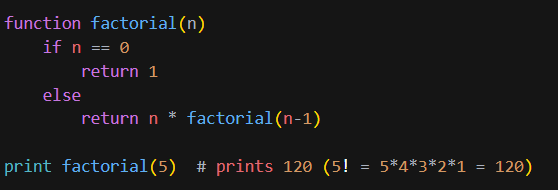


Fig. 1.2 Pseudocode of Factorial numbers

This pseudocode defines a function called factorial that takes a single argument **n** and returns the factorial of **n**. The factorial of a number **n** is the product of all the positive integers from 1 to **n**, and is denoted by **n!**.

The function uses a recursive approach to calculate the factorial by calling itself with **n-1** as the argument. The base case of the recursion is when **n** is 0, in which case the function returns 1. For all other values of **n**, the function returns **n** multiplied by the factorial of

**n-1.**

The pseudocode also includes a call to the factorial function to calculate the factorial of 5 and print the result. This will output the value 120, which is the factorial of 5 (5! = 54321 = 120).

## 1.7 Dry run of factorial number

Here is an example of a dry run of the factorial for the number 4:

* The factorial of a number is the product of that number and all the positive integers that come before it. For example, the factorial of 4 (4!) is 4 \* 3 \* 2 \* 1 = 24.
* To calculate the factorial of 4, we start by setting the initial value to 1.
* We then multiply the initial value by 4 to get 4.
* We then multiply the result by 3 to get 12.
* We then multiply the result by 2 to get 24.
* We then multiply the result by 1 to get the final result of 24.

So, the factorial of 4 is 24.

## 1.8 Python program for factorial number

*def factorial(n):*

*# Check if the input is valid (must be a positive integer)*

*if n < 0:*

*print("Invalid input")*

*elif n == 0:*

*return 1*

*else:*

*# Calculate the factorial by multiplying the number by the factorial of the number before it*

*result = n \* factorial(n-1)*

*return result*

*# Test the function with different input values*

*print(factorial(4))*

*print(factorial(5))*

*print(factorial(6))*

## 1.9 Big-O notation

Big-O notation is a way of expressing the asymptotic complexity of an algorithm, which describes how the running time of the algorithm grows as the input size increases. It gives a rough estimate of the upper bound on the running time of an algorithm, ignoring constants and lower-order terms. (Huang, n.d.)

In Big-O notation, the complexity of an algorithm is expressed as a function of the size of the input, denoted by n. For example, an algorithm with a complexity of O(n) means that the running time of the algorithm grows linearly with the size of the input. An algorithm with a complexity of O(n^2) means that the running time grows as the square of the input size.

Big-O notation is commonly used to compare the relative efficiency of different algorithms. For example, if one algorithm has a complexity of O(n) and another algorithm has a complexity of O(n^2), the first algorithm will be faster for large inputs because the running time grows more slowly as the input size increases.

There are many other complexities that can be expressed using Big-O notation, including O(log n), O(n log n), and O(n!). The specific complexity of an algorithm depends on the details of the implementation and the input data.

## 1.10 Big-O notation's role in evaluating efficiencies of algorithms

The role of Big-O notation in evaluating the efficiency of algorithms is to provide a way to compare the performance of different algorithms. It allows us to determine which algorithm is the most efficient for a given problem by comparing the growth of their runtime or number of operations as the input size increases.

For example, if we have two algorithms that both solve the same problem, but one has a runtime that grows linearly with the input size (O(n)), while the other has a runtime that grows exponentially with the input size (O(2^n)), we can conclude that the first algorithm is more efficient because its runtime grows at a slower rate as the input size increases.

Big-O notation is often used to describe the worst-case performance of an algorithm, but it can also be used to describe the average-case or best-case performance. It is a widely used tool in the field of computer science for analyzing the efficiency of algorithms and helping to choose the most efficient one for a given problem.

## 1.11 Evaluate the efficiency

The efficiency of the Python code for the Fibonacci series can be evaluated using Big-O notation as follows:

The Fibonacci series code has a for loop that iterates from 2 to n+1, where n is the input size. Therefore, the runtime of the code is O(n), which means that the number of operations grows linearly with the input size.

In terms of space complexity, the code uses a constant amount of memory to store the variables a, b, and c, regardless of the input size. Therefore, the space complexity of the code is O(1), which means that the amount of memory used is independent of the input size.

The efficiency of the Python code for the factorial function can also be evaluated using Big-O notation as follows:

The factorial function code has a recursive structure, where the function calls itself with an input size that is one less than the current input size. Therefore, the runtime of the code is O(n), which means that the number of operations grows linearly with the input size.

In terms of space complexity, the code uses a constant amount of memory to store the variable result, regardless of the input size. However, each time the function calls itself, it creates a new stack frame, which uses additional memory. Therefore, the space complexity of the code is O(n), which means that the amount of memory used grows linearly with the input size.

Overall, both the Fibonacci series code and the factorial function code have a linear time complexity, which means that they are efficient for input sizes that are not too large. However, the factorial function has a higher space complexity due to its recursive structure, which means that it may not be suitable for very large input sizes that require a lot of memory.

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

## 2.1 Define Programming Paradigm

A programming paradigm is a style or approach to programming that defines how a program is structured and written. Different programming paradigms have different characteristics and ways of organizing code, and can be used to solve different types of problems. (Geeks, 2022)

Some common programming paradigms include:

* Imperative programming: This paradigm focuses on describing how a program should execute, using statements that change the program's state. Examples of imperative languages include C, C++, and Java.
* Functional programming: This paradigm emphasizes the use of functions to transform data and avoid changing state. Functional languages include Haskell, Lisp, and ML.
* Object-oriented programming: This paradigm is based on the idea of encapsulating data and behavior in objects, which can interact with each other through methods. Object-oriented languages include Java, C#, and Python.
* Logic programming: This paradigm is based on representing knowledge as logical statements and using a reasoning engine to deduce conclusions. Logic programming languages include Prolog and Mercury.
* Concurrent programming: This paradigm is concerned with writing programs that can execute multiple tasks concurrently. Concurrent programming languages include Erlang and Go.

There are many other programming paradigms in addition to these, and many programming languages support multiple paradigms.

## 2.2 Main characteristics of Procedural Programming Paradigm

Procedural programming is a programming paradigm that focuses on organizing code into procedures or functions that perform specific tasks. Some main characteristics of procedural programming include:

* Modularity: Procedural programming emphasizes the use of procedures or functions to break down a problem into smaller, more manageable pieces. This makes it easier to understand and maintain the code.
* Top-down design: Procedural programming often follows a top-down design approach, in which the problem is decomposed into smaller subproblems that are solved in a specific order.
* Sequential execution: Procedures or functions in procedural programming are typically executed in a specific sequence, following the order in which they are defined.
* Data manipulation: Procedural programming often involves manipulating data using variables and arrays.
* Imperative style: Procedural programming often uses an imperative style, in which the program specifies how to execute a task using statements that change the program's state.

Procedural programming is a widely used paradigm and is supported by many programming languages, including C, Pascal, and BASIC. It is particularly well-suited for problems that can be divided into discrete steps and is often used for systems programming and developing algorithms.

## 2.3 Main characteristics of Object-oriented Programming Paradigm

Object-oriented programming (OOP) is a programming paradigm that is based on the idea of encapsulating data and behavior in objects. Some main characteristics of OOP include:

* Abstraction: In object-oriented programming (OOP), abstraction refers to the process of representing essential features of an object without including the implementation details. Abstraction allows objects to be defined in terms of their capabilities and characteristics, rather than their specific implementation.
* Encapsulation: OOP emphasizes the use of encapsulation, which means that the data and behavior of an object are packaged together and are not directly accessible to external entities. This helps to protect the internal state of an object and reduces the risk of unintended side effects.
* Inheritance: OOP supports inheritance, which allows objects to inherit characteristics and behavior from a parent object. This allows objects to be extended and customized without having to rewrite the entire codebase.
* Polymorphism: OOP supports polymorphism, which means that an object can take on multiple forms depending on the context in which it is used. This allows objects to be used in a flexible and reusable way.
* Message passing: In OOP, objects communicate with each other by sending and receiving messages. This allows objects to interact and collaborate to solve a problem.

OOP is a widely used paradigm and is supported by many programming languages, including Java, C#, and Python. It is particularly well-suited for developing large, complex systems and is often used for software development in business and industry.

## 2.4 Main characteristics of Event-driven Programming Paradigm

Event-driven programming is a programming paradigm in which the flow of the program is determined by events that are triggered by the user or by the system. Some main characteristics of event-driven programming include:

* Asynchronous execution: In event-driven programming, the program does not follow a predetermined flow, but rather executes in response to events that are triggered by the user or the system. This allows the program to be more responsive and to handle multiple tasks concurrently.
* Event handling: Event-driven programming involves defining event handlers, which are functions that are executed in response to specific events. Event handlers are typically registered with the system or user interface and are called when the corresponding event occurs.
* Callback functions: Event-driven programming often involves the use of callback functions, which are functions that are passed as arguments to other functions and are called when a specific event or condition occurs. This allows the program to be more flexible and responsive.
* User-driven execution: Event-driven programming is often used in user interfaces, where the program executes in response to user input or other events that are triggered by the user.

Event-driven programming is a widely used paradigm and is supported by many programming languages, including JavaScript, Python, and C#. It is particularly well-suited for developing interactive programs and user interfaces, and is often used for web development and desktop applications.

## 2.5 Code snippet for procedural programming

*def add(x, y):*

*result = x + y*

*return result*

*print(add(3, 4))*

## 2.6 Code snippet for object-oriented programming

A simple class that defines a point in two-dimensional space:

*class Point:*

*def \_\_init\_\_(self, x, y):*

*self.x = x*

*self.y = y*

*point1 = Point(3, 4)*

*point2 = Point(5, 6)*

*print(point1.x, point1.y)*

*print(point2.x, point2.y)*

## 2.7 Code snippet for event-driven programming

A simple program that responds to button clicks:

*from tkinter import \**

*def button\_click():*

*print("Button clicked")*

*root = Tk()*

*button = Button(root, text="Click me", command=button\_click)*

*button.pack()*

*root.mainloop()*

## 2.8 Evaluate based on structure and the unique characteristics programming paradigms

1. Procedural programming:

* Focus on breaking a program down into small, reusable chunks of code called procedures or functions
* Modularity and separation of concerns
* Logical order and sequence of operations
* Reusable code that can be called multiple times to solve different problems

1. Object-Oriented programming:

* Focus on encapsulating data and behavior in objects
* Objects interact with each other through methods
* Inheritance, polymorphism, and encapsulation
* Objects can be created, modified, and deleted as needed
* Modeling real-world systems or processes

1. Event-Driven programming:

* Focus on responding to events or triggers that occur in the program or in the environment
* Program waits for a specific event or trigger to occur, and then executes a predetermined set of instructions in response
* Often used in programs with a graphical user interface (GUI) and user interaction
* Used to create interactive programs that respond to user input or external events.

# 3. Implement basic algorithms in code using an IDE.

## 3.1 Write suable algorithms for vehicle tariff calculation for rents and hires

Rent calculation flow chart



Fig. 3.1 Rent calculation flow chart

Day tour – hire calculation flow chart

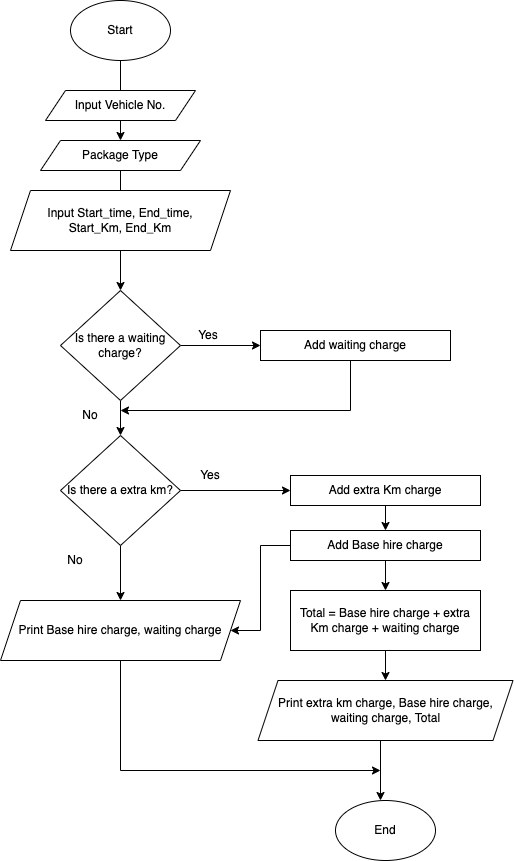


Fig. 3.2 Day tour – hire calculation flow chart

Long tour – hire calculation flow chart

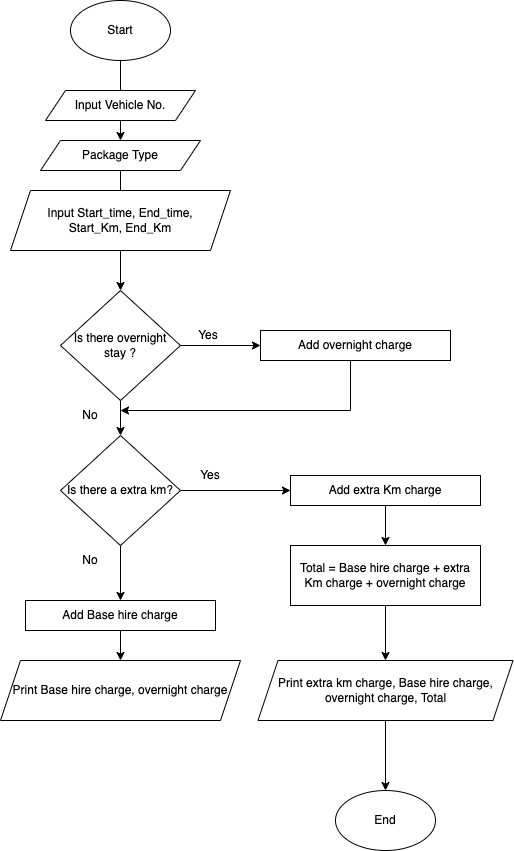


Fig. 3.3 Long tour – hire calculation flow chart

## Use the visual studio IDE (using C#.net) to Implement the above algorithms and design the suitable database structure

***Function 1: Rent calculation***

Text

Description automatically generated

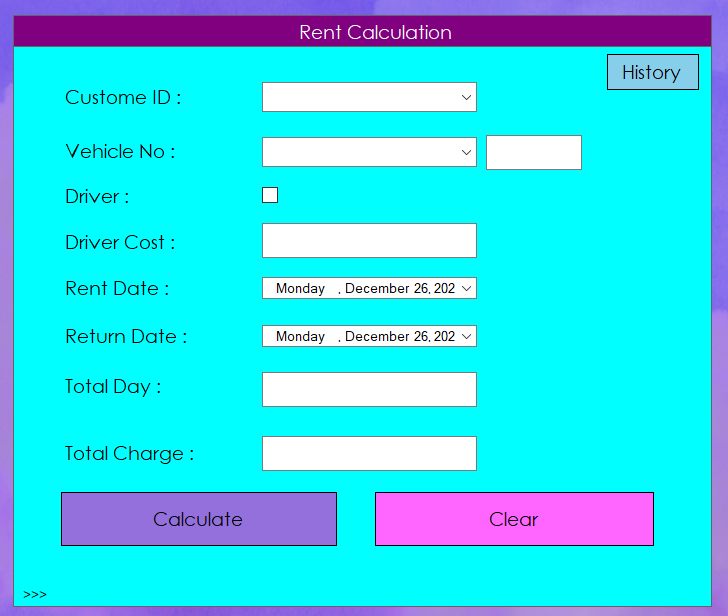


Fig. 3.4 Rent Calculation GUI

***Function 2: Day tour calculation***

Graphical user interface, text

Description automatically generated

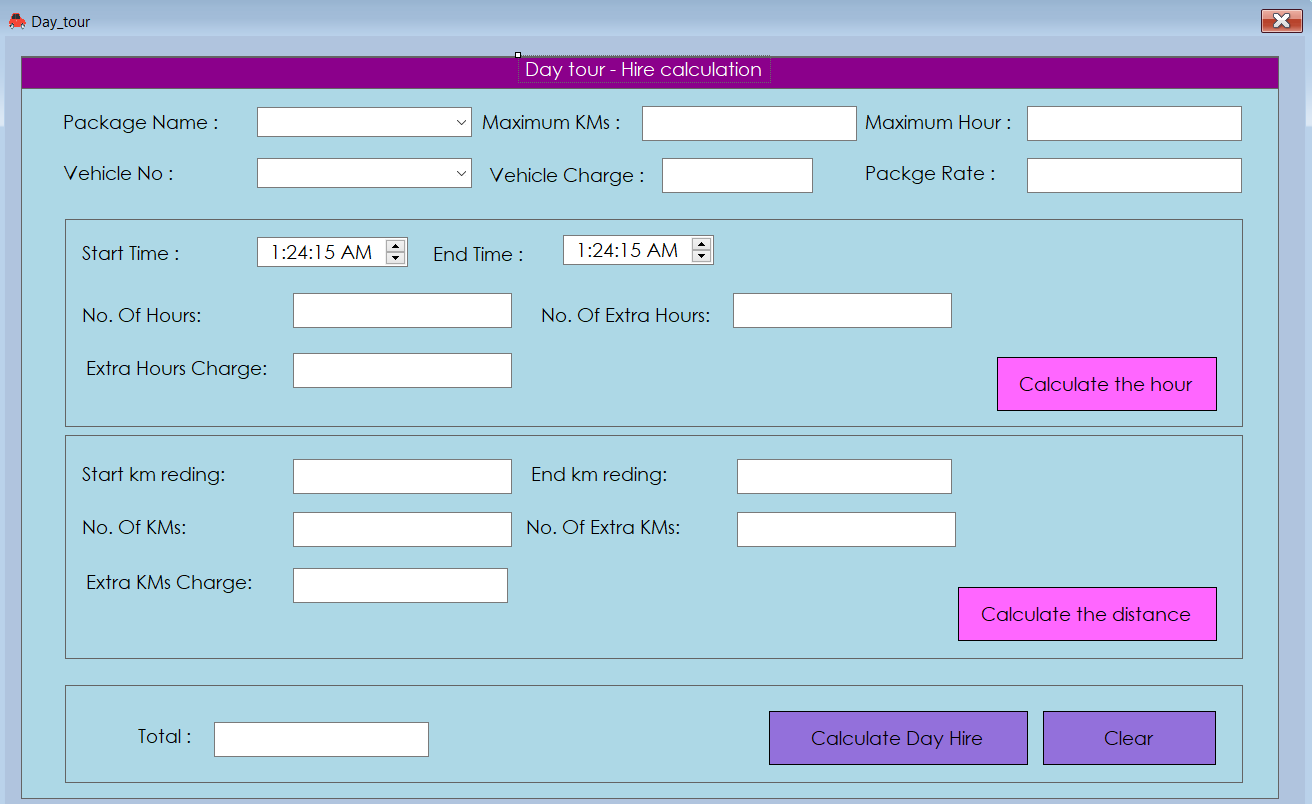


Fig. 3.5 Day tour – hire calculation GUI

***Function 3: Long tour calculation***

A picture containing text

Description automatically generated

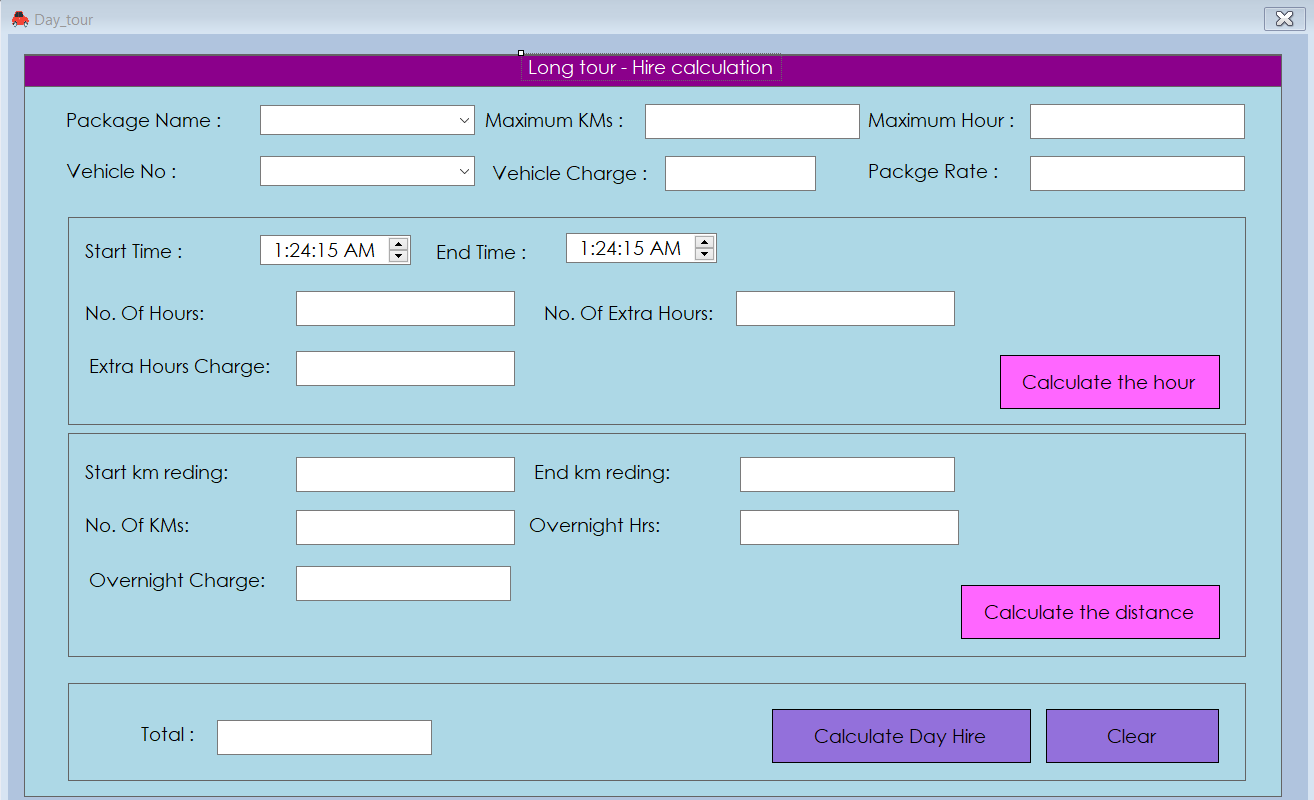


Fig. 3.6 Long Tour – hire calculation GUI

## 3.3 Implementation of Database Structure

This application includes a database with approximately 6 tables to store all the information for rentals, day tours, long tours, package details, vehicle registration, and driver registration. It also allows certain users to modify, add, and delete details through specific functions implemented within the application.

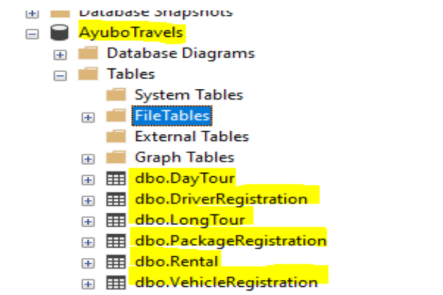


Fig. 3.7 Database structure GUI

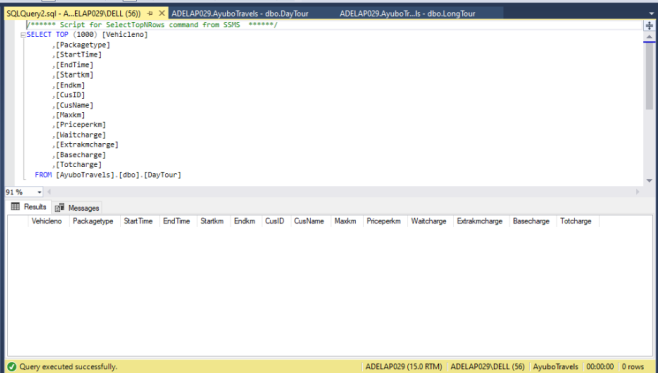


Fig. 3.8 Tabular format view of Day tour from database

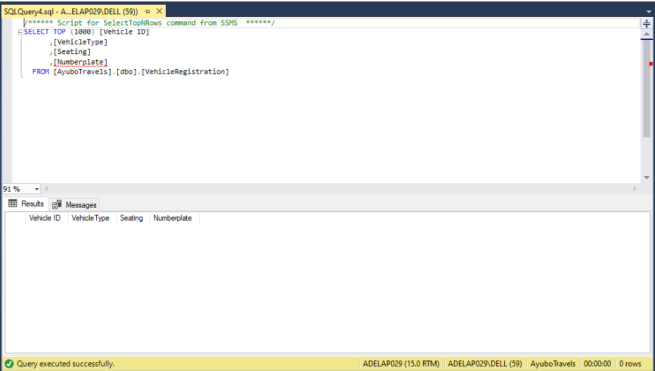


Fig. 3.9 Tabular format view of vehicle registration from database

## 3.4 Analysis on the feature of an integrated development environment (IDE)

A software application that provides a range of tools for developing software, called an Integrated Development Environment (IDE).These tools can include a code editor, compiler, debugger, and other utilities that are necessary for writing, testing, and debugging software. Some common features of IDEs include:

* Code editor: A code editor is a feature that allows you to write and edit source code. It typically includes features like syntax highlighting, code completion, and code formatting to make it easier to write and read code.
* Compiler: A compiler is a feature that converts source code into an executable program. It checks the code for syntax errors and converts it into a form that can be run on a computer.
* Debugger: A debugger is a feature that allows you to find and fix errors in your code. It allows you to set breakpoints in your code, step through the code line by line, and inspect the values of variables at different points in the code.
* Version control: Version control is a feature that allows you to track changes to your code over time and collaborate with other developers. It allows you to save different versions of your code, revert to previous versions if necessary, and merge changes made by multiple developers.
* Integration with other tools: Many IDEs offer integration with other tools and services that are commonly used in software development, such as build systems, testing frameworks, and project management tools. This can make it easier to use these tools within a single application.
* Customization: Many IDEs allow you to customize the layout and appearance of the interface, as well as the behavior of the tools and features. This can make it easier to work with the IDE in a way that is most comfortable and efficient for you.

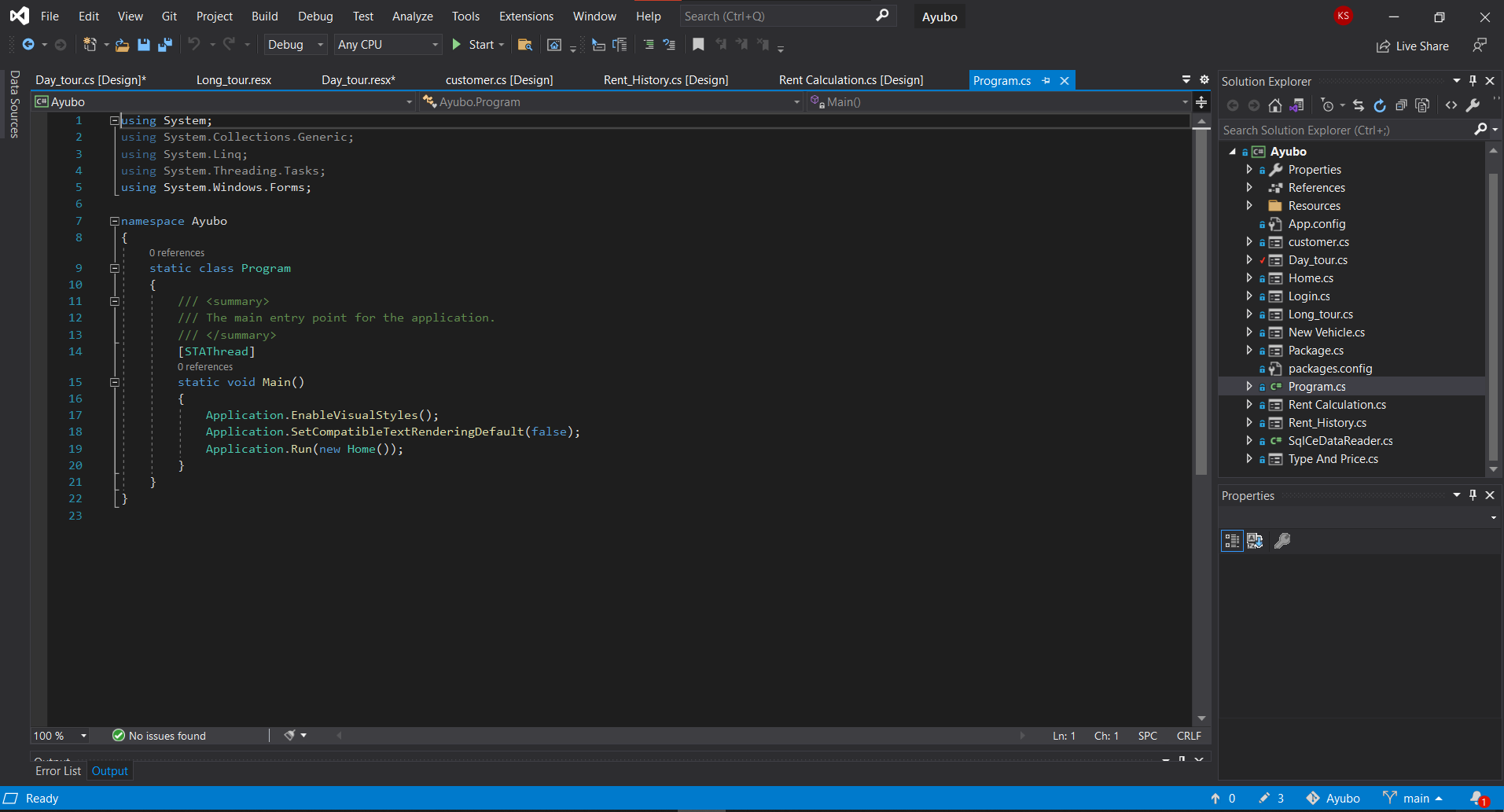


Fig. 3.10 Visual Studio IDE features

Visual Studio is a popular Integrated Development Environment (IDE) developed by Microsoft. It includes a wide range of features and tools for software development, including:

* Code editor: Visual Studio includes a powerful code editor that supports syntax highlighting and code completion for a variety of programming languages. It also includes features like IntelliSense, which can provide suggestions for code completion and help you navigate through your code more easily.
* Debugging: Visual Studio includes a debugger that allows you to find and fix errors in your code. It allows you to set breakpoints, step through code line by line, and inspect the values of variables at different points in the code.
* Integration with other tools: Visual Studio integrates with a variety of tools and services that are commonly used in software development, such as build systems, testing frameworks, and project management tools.
* Customization: Visual Studio allows you to customize the layout and appearance of the interface, as well as the behavior of the tools and features. This can make it easier to work with the IDE in a way that is most comfortable and efficient for you.
* Live Share: Visual Studio includes Live Share, a feature that allows you to collaborate with other developers in real-time. With Live Share, you can edit and debug code together, regardless of whether you are in the same physical location.
* Artificial intelligence: Visual Studio includes artificial intelligence (AI) features that can help you write better code. For example, the IntelliCode feature uses AI to provide code completion suggestions based on patterns learned from millions of lines of code.
* Mobile development: Visual Studio includes tools and features for building mobile applications for Android, iOS, and other platforms. This includes tools for designing user interfaces, debugging, and testing.

These are just a few examples of the many features and tools included in Visual Studio. It is a comprehensive IDE that is suitable for a wide range of software development projects.

## 3.5 Evaluate the use of the Visual Studio IDE for your application development contrasted with not using an IDE

Using the Visual Studio Integrated Development Environment (IDE) can provide several benefits for application development, compared to not using an IDE. Some of the advantages of using an IDE like Visual Studio include:

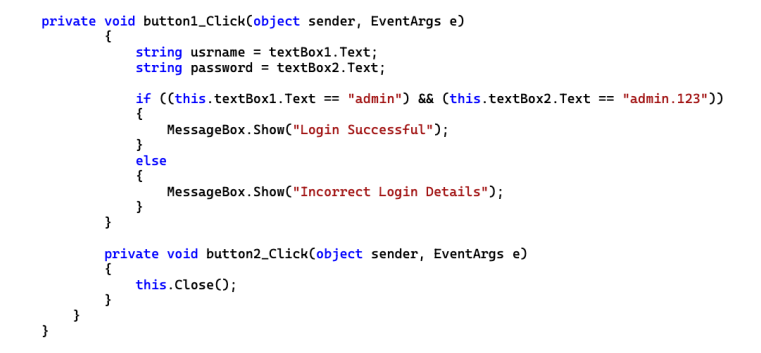
* Productivity: IDEs like Visual Studio include a wide range of tools and features that can make it easier and more efficient to develop applications. For example, the code editor includes features like syntax highlighting, code completion, and code formatting, which can help you write and read code more easily. The debugger allows you to find and fix errors in your code, and the integration with other tools and services can make it easier to use these tools within a single application.
* Collaboration: IDEs like Visual Studio include features that can facilitate collaboration with other developers. For example, the Live Share feature allows you to edit and debug code together in real-time, regardless of whether you are in the same physical location. This can make it easier to work on projects with other people and share knowledge and ideas.
* Customization: IDEs like Visual Studio allow you to customize the layout and appearance of the interface, as well as the behavior of the tools and features. This can make it easier to work with the IDE in a way that is most comfortable and efficient for you.
* Artificial intelligence: IDEs like Visual Studio include artificial intelligence (AI) features that can help you write better code. For example, the IntelliCode feature uses AI to provide code completion suggestions based on patterns learned from millions of lines of code.
* Mobile development: IDEs like Visual Studio include tools and features for building mobile applications for Android, iOS, and other platforms. This includes tools for designing user interfaces, debugging, and testing.

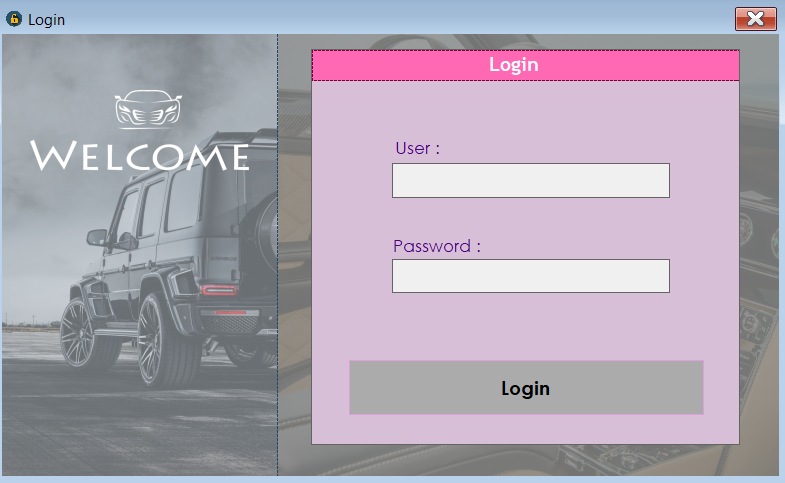
On the other hand, not using an IDE can have some drawbacks. For example, you may need to use multiple tools and services to develop your application, which can be time-consuming and may not offer the same level of integration as an IDE. Additionally, you may not have access to features like code completion, debugging, and collaboration that can make development more efficient.

# Determine the debugging process and explain the importance of a coding standard

## 4.1 Design and build a small system to calculate vehicle hire amounts and record them in a database for customer billing and management reporting for Ayubo drive.

1. Login



  
Fig. 4.1 Login view GUI

1. Main menu

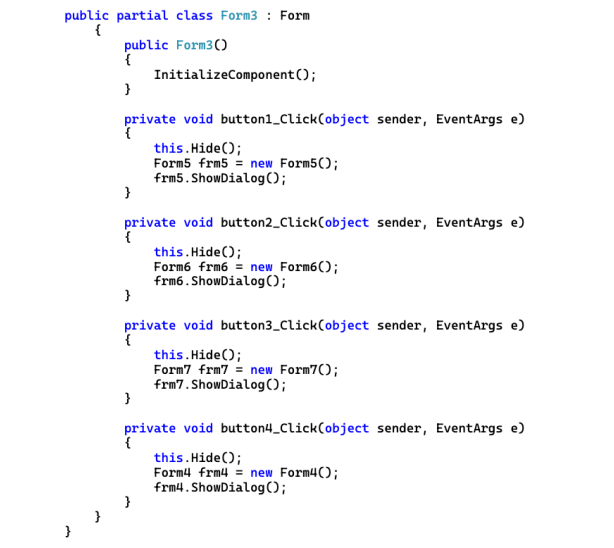
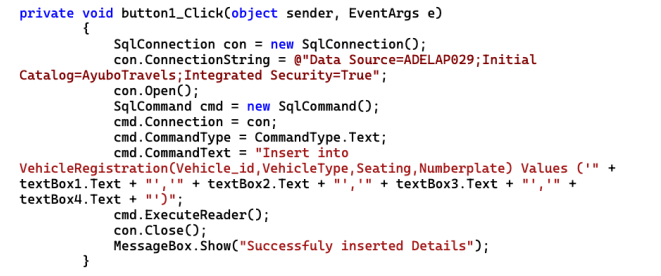




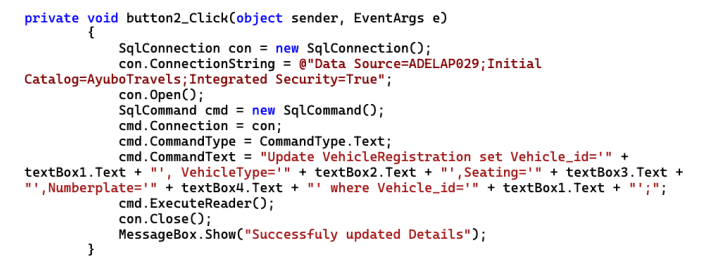
Fig. 4.2 Main menu view GUI

1. Vehicle management

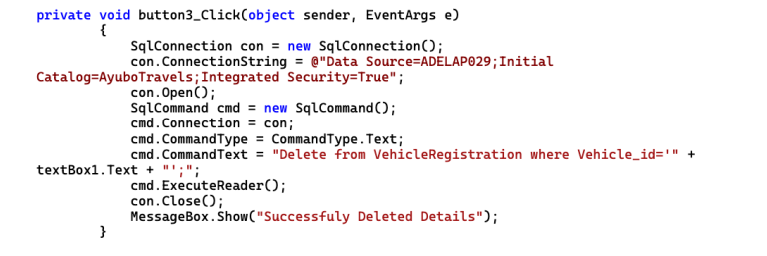
* Add vehicle



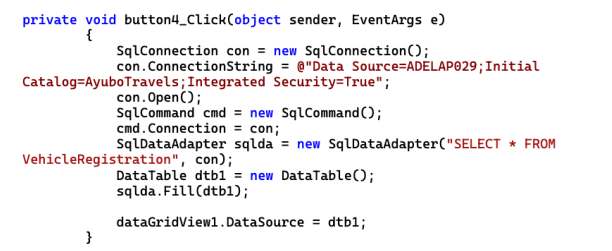
* Update Vehicle



* Delete vehicle



* View vehicle



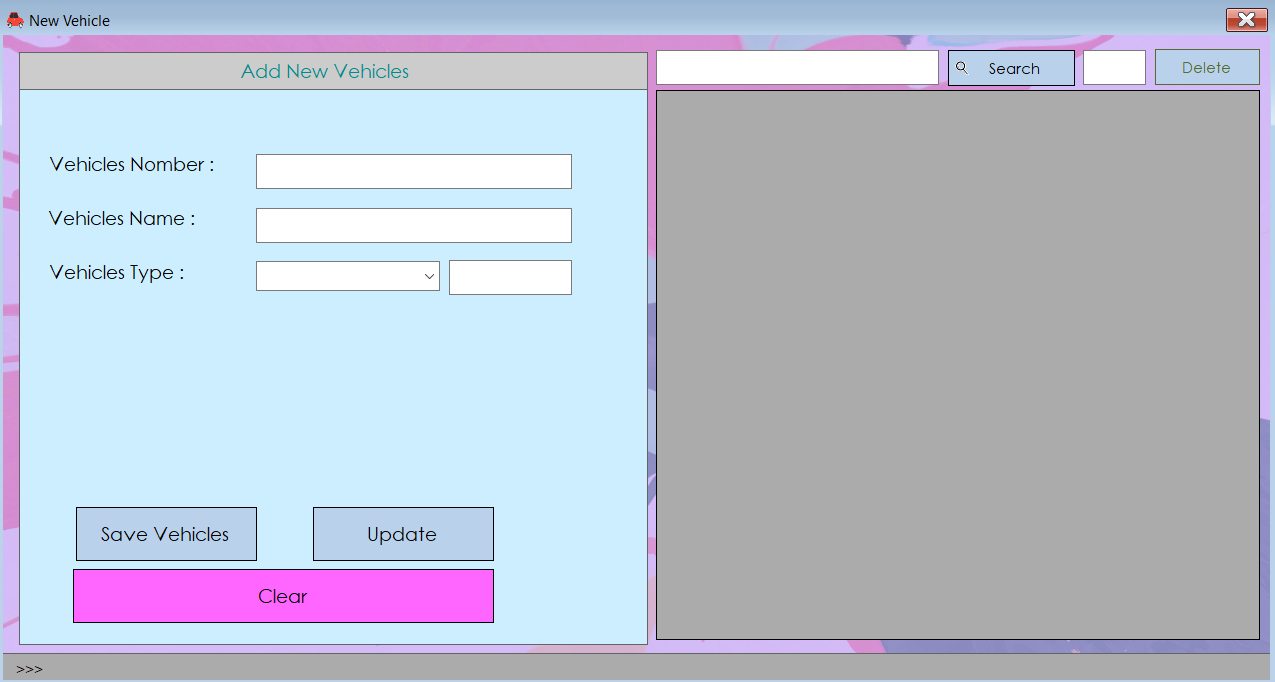


Fig. 4.3 Vehicle management GUI

## 4.2 Debugging process

The debugging process is a methodical approach to finding and fixing errors in software. It is an important part of the software development process and can help you identify and resolve issues in your code. The general steps in the debugging process are:

* Identify the problem: The first step in debugging is to identify the problem or error that you are trying to fix. This may involve reviewing error messages, analyzing the code, or testing the software to determine what is causing the issue.
* Reproduce the problem: Once you have identified the problem, it is important to reproduce the issue in a controlled environment. This can help you narrow down the cause of the problem and make it easier to fix.
* Determine the cause of the problem: Once you have reproduced the problem, you can begin to analyze the code to determine the root cause of the issue. This may involve using tools like a debugger to inspect the values of variables or stepping through the code line by line to see how it is executing.
* Develop a solution: Once you have determined the cause of the problem, you can begin to develop a solution to fix it. This may involve making changes to the code or implementing a workaround to avoid the issue.
* Test the solution: After you have developed a solution, it is important to test it to ensure that it resolves the problem. This may involve running the software again and verifying that the issue has been fixed.
* Document and deploy the fix: After you have successfully tested the solution, it is important to document the changes you made and deploy the fix to the appropriate environment. This can help ensure that the issue is permanently resolved and that others are aware of the changes that were made.

The debugging process can be iterative, meaning that you may need to repeat some or all of these steps multiple times until the problem is fully resolved. It is an important part of the software development process and can help you identify and fix issues in your code efficiently and effectively.

**Debugging process of Visual Studio IDE**

The Visual Studio Integrated Development Environment (IDE) includes a powerful debugger that can be used to identify and fix errors in your code. The general steps in the debugging process using Visual Studio are:

* Set a breakpoint: A breakpoint is a point in your code where the debugger will pause execution and allow you to inspect the values of variables and step through the code line by line. To set a breakpoint in Visual Studio, you can click on the left margin of the code editor or use the "Debug > Toggle Breakpoint" menu option.

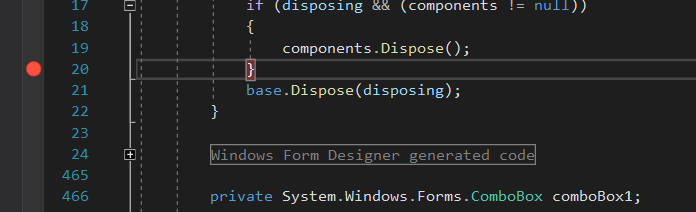


Fig. 4.4 Breakpoint usage

* Start debugging: To start debugging your code in Visual Studio, you can select "Debug > Start Debugging" from the menu or use the F5 key. This will launch the debugger and execute your code until it reaches a breakpoint.

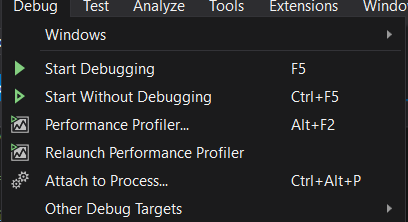


Fig. 4.5 Debugging process

* Inspect variables and step through code: When the debugger reaches a breakpoint, it will pause execution and allow you to inspect the values of variables and step through the code line by line. You can use the "Watch" window to inspect the values of specific variables, or you can use the "Locals" window to see all of the variables that are in scope. You can also use the "Step Over" and "Step Into" buttons to step through the code line by line.

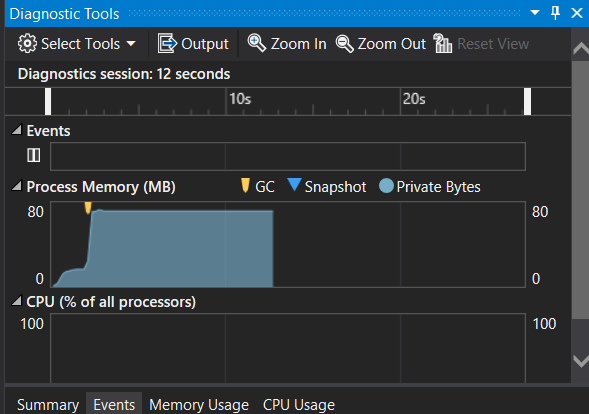


Fig. 4.6 Diagnostic tools

* Resume execution: When you are finished inspecting the code and variables, you can use the "Continue" button to resume execution of the code. The debugger will continue executing the code until it reaches the next breakpoint or the end of the program.
* Fix the problem: Once you have identified the cause of the problem, you can make changes to the code to fix it. You can then use the debugger to verify that the issue has been resolved.

Using the debugger in Visual Studio can be an effective way to identify and fix errors in your code. It allows you to inspect the values of variables, step through the code line by line, and make changes to the code to resolve issues. (Microsoft, n.d.)

Some of the features available in Visual Studio for debugging your code more easily include:

* Breakpoints: Breakpoints are points in your code where the debugger will pause execution and allow you to inspect the values of variables and step through the code line by line. Visual Studio allows you to set breakpoints in the code editor by clicking on the left margin or using the "Debug > Toggle Breakpoint" menu option.
* Debugging windows: Visual Studio includes several debugging windows that allow you to inspect the values of variables, view the call stack, and see the output of the debugger. These windows include the "Watch" window, the "Locals" window, the "Call Stack" window, and the "Output" window.
* Step through code: The debugger in Visual Studio allows you to step through code line by line, either by using the "Step Over" and "Step Into" buttons or by pressing the F11 and F10 keys. This can be useful for identifying the root cause of an issue or for testing specific parts of your code.
* Inspect variables: Visual Studio allows you to inspect the values of variables while debugging your code. You can use the "Watch" window to inspect specific variables, or you can use the "Locals" window to see all of the variables that are in scope.
* Edit and continue: Visual Studio allows you to make changes to your code while debugging and apply those changes without stopping the debugger. This can be useful for quickly testing changes or for fixing issues without having to restart the debugger.
* Exception handling: Visual Studio includes exception handling tools that allow you to identify and troubleshoot exceptions in your code. You can use the "Exception Settings" window to specify which types of exceptions you want to catch and how you want the debugger to handle them.

Here are some examples of how you can use the debugging process to develop more secure and robust applications:

* Identify and fix security vulnerabilities: By using the debugging process to identify and fix errors in your code, you can reduce the risk of security vulnerabilities. For example, you might use the debugger to identify and fix input validation errors that could allow attackers to inject malicious code into your application.
* Improve stability: By using the debugging process to identify and fix errors in your code, you can improve the stability of your application. This can help reduce the risk of crashes or other issues that could disrupt the user experience.
* Enhance reliability: By using the debugging process to identify and fix errors in your code, you can improve the reliability of your application. This can help ensure that your application performs as expected and meets the needs of your users.
* Streamline development: By using the debugging process to identify and fix errors early in the development process, you can streamline the development process and reduce the time and resources required to complete your project.
* Enhance user experience: By using the debugging process to identify and fix errors in your code, you can improve the user experience of your application. This can help increase user satisfaction and loyalty, and may even lead to increased revenue.

Overall, the debugging process is an important part of developing more secure, robust applications. By using the debugging process to identify and fix errors in your code, you can improve the stability, reliability, and security of your application, as well as streamline the development process and enhance the user experience.

## 4.3 Outline the coding standards you have used in your application development

Coding standards are guidelines that define how code should be written and structured in order to ensure consistency, readability, and maintainability. There are many different coding standards that you can use in application development, and the specific standards you use may depend on the programming language, the type of application you are developing, and the preferences of your team. Here are some examples of coding standards that you might use in application development:

* Naming conventions: Naming conventions define how variables, functions, and other code elements should be named. For example, you might use a naming convention that requires variables to be named using camelCase and requires functions to be named using PascalCase.

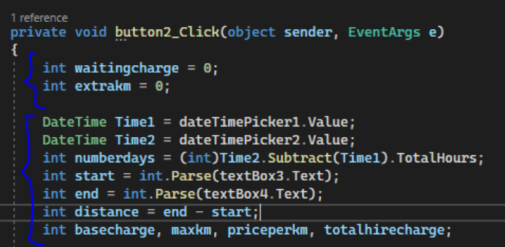


Fig. 4.7 Naming conventions

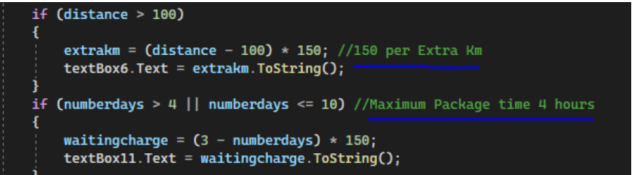
* Indentation: Indentation standards define how code should be indented to indicate the structure and hierarchy of the code. For example, you might use a standard that requires code blocks to be indented by four spaces or one tab.
* Spacing: Spacing standards define how code should be spaced to improve readability. For example, you might use a standard that requires a space after every comma and requires a blank line between functions.
* Comments: Commenting standards define how and when code should be commented to provide explanations and context for the code. For example, you might use a standard that requires comments to be placed above every function and requires all comments to be written in English.
* 

Fig. 4.8 Usage of comments

* Code organization: Code organization standards define how code should be organized within a file or project to improve readability and maintainability. For example, you might use a standard that requires all functions to be placed at the top of the file or that requires related functions to be grouped together.

By using coding standards in your application development, you can improve the consistency, readability, and maintainability of your code. This can make it easier for you and your team to work with the code and can help ensure that your application is of high quality.

# References

Ekunwe, I., 2022. *An Introduction to Algorithms,* s.l.: Dev.to.

Geeks, G. f., 2022. *Programming Paradigm,* s.l.: s.n.

Huang, S., n.d. *What is Big O Notation Explained: Space and Time Complexity,* s.l.: freeCodeCamp.org.

Jawady, M. H., 2019. *Can you recommend a good visualization tool for sorting algorithms?,* s.l.: Dev.to.

Microsoft, n.d. *.NET documentation,* s.l.: s.n.