МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

ШПАК Володимир Олександрович

# Додаток автоматизованого синтезу програмного коду мовою JavaScript / Software application for automated synthesis of software code in the JavaScript language

спеціальність: 123 – Комп’ютерна інженерія
освітньо-професійна програма – Комп’ютерна інженерія

Кваліфікаційна робота

Виконав: студент групи КІ-41
ШПАК Володимир Олександрович

Науковий керівник
к.т.н. Батько Ю.М.

ТЕРНОПІЛЬ - 2023

# РЕЗЮМЕ

Кваліфікаційна робота на тему “Додаток автоматизованого синтезу програмного коду мовою JavaScript” зі спеціальності 123 “Комп’ютерна інженерія” освітнього ступеня “бакалавр” містить 78 сторінок пояснюючої записки, 29 рисунки, 2 таблиці, 6 додатків.

Метою кваліфікаційної роботи є розроблення програмного засобу розпізнавання природньої мови та синтез програмного коду, мовою JavaScript.

Методи дослідження це методи теорії алгоритмів і використовують для структуризації нечітких порівнянь, зчитування природньої мови, та створення програмного додатку на основі методів функціонального програмування.

Результатом кваліфікаційної роботи буде алгоритм, який зможе ефективно та точно синтезувати програмний код мовою JavaScript на основі вхідних даних у форматі природної мови. Результати можуть бути використані для створення нових автоматизованих інструментів, які забезпечать швидкий та ефективний синтез програмного коду мовою JavaScript на основі вхідних даних у форматі природної мови.

Ключові слова: АЛГОРИТМ, PYTHON, GOOGLE-SPEECH-TO-TEXT, JAVASCRIPT, API.

# RESUME

The qualification work on the topic "Software application for automated synthesis of software code in JavaScript" from specialty 123 "Computer engineering" of the bachelor's degree contains 78 pages of explanatory note, 29 figures, 2 table, 6 appendices.

The purpose of the qualification work is to develop a software tool for natural language recognition and synthesis of software code in the JavaScript language.

Research methods are algorithm theory methods and are used to structure fuzzy comparisons, read natural language, and create a software application based on functional programming methods.

The result of the qualification work will be an algorithm that can efficiently and accurately synthesize JavaScript programming code based on input data in natural language format. The results can be used to create new automated tools that will provide fast and efficient synthesis of JavaScript programming code based on input data in natural language format.

Keywords: ALGORITHM, PYTHON, GOOGLE SPEECH TO TEXT, JAVASCRIPT, API.

ЗМІСТ

Перелік умовних скорочень……..……10
Вступ…….……..11
1 Природні та структуровані мови, класифікація та особливості використання..13
1.1 Природні та структуровані мови ……....13
1.2 Мови програмування, класифікація, сфери застосування…….....16
1.3 Програмні засоби розпізнавання природньої мови …..…..……..22
1.4 Постановка завдань кваліфікаційної роботи....…….....26
2 Методи та алгоритми розпізнавання природніх мов……..27
2.1 Алгоритми розпізнавання природніх мов.....27
2.2 Структури мов програмування Javascript…….……31
2.3 Алгоритм синтезу програмного коду мовою JS на основ розпізнавання природніх мов.....34
3 Програмний додаток синтезу програмного коду мовою Javascript.....38
3.1. Структура програмного додатку.....……..38
3.2 Реалізація модулів програмного додатку.....……...40
3.3 Тестування та порівняння програмного додатку з аналогами.....43
Техніко-економічний розділ…….....50
4.1 Визначення трудомісткості розробки програмного забезпечення……...50
4.2 Розрахунок витрат на створення програмного забезпечення……..55
4.4 Розрахунок можливої ціни.....61
4.3 Розрахунок показників економічної ефективності розробки програмного продукту…….62
Висновки……64
Список використаних джерел ……65

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КР.КІ.8091476.00.00.000 ПЗ | | |
| Змн. | Лист | № докум. | Підпис | Дата | | | |
| Розробив | | Шпак В.О. | | | АЛГОРИТМ АВТОМАТИЗОВАНОГО СИНТЕЗУ ПРОГРАМНОГО КОДУ МОВОЮ JAVASCRIPT | Літ. | Арк. 8 / Акрушів 71 |
| Перевір. | | Батько Ю.М. | | | | | |
| Консульт. | | Савка Н.Я. | | | | ЗУНУ,ФКІТ, КІ-41 | |
| Н. Контр. | | Мельник Г.М. | | | | | |
| Затвердив | | Дубчак Л.О | | | | | |

Додаток А Модуль main.py ……..…….....70
Додаток Б Модуль ask.py …….....…….....……..….71
Додаток В Модуль reask.py……..……72
Додаток Г Модуль arrMethods.py……....……73
Додаток Д Довідка про використання.....75
Додаток Е Світлокопії виданих публікацій.....……76

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

| | | |
|---|---|---|
| API | – | Application Programming Interface |
| JS | – | JavaScript |
| NLP | – | Natural Language Processing |
| SISO | – | Single Input, Single Output |
| k-NN | – | k-Nearest Neighbors |
| IDE | – | Integrated Development Environment |
| ES6 | – | ECMAScript 2015 |
| БД | – | База Даних |
| ЕОМ | – | Електронна обчислювальна машина |
| ПК | – | Персональний комп’ютер |

# ВСТУП

У сучасному цифровому світі, де швидкість технологічного розвитку постійно зростає, процес розробки програмного забезпечення стає все більш складним і вимагає від розробників швидкості, ефективності та інноваційних підходів. В цьому контексті, тема “Алгоритм синтезу програмного коду мовою JavaScript на основі розпізнавання природних мов” стає надзвичайно актуальною.

Мова JavaScript є однією з найпопулярніших мов програмування, особливо в контексті веб-розробки. Велика кількість проектів та інноваційних рішень розробляються саме з використанням JavaScript. Проте, ручне написання програмного коду може бути складним і часоємним процесом, особливо для непрофесійних розробників.

Тому, актуальність теми полягає в пошуку способів автоматизації процесу генерації програмного коду мовою JavaScript. Використання розпізнавання природних мов для синтезу коду може забезпечити ефективний і точний спосіб створення програмного забезпечення, що відповідає вимогам та намірам розробників.

Технології розпізнавання природних мов та обробки природної мови перебувають на передній лінії досліджень у сфері штучного інтелекту. Застосування цих технологій у синтезі програмного коду відкриває нові горизонти для автоматизації розробки, спрощення процесу програмування та полегшення доступу до програмування для широкого кола користувачів.

Одним із підходів є використання природної мови для описування задач або вимог до програмного продукту, а потім перетворення цих описів у функціональний код. Це може зменшити необхідність в глибокому розумінні мов програмування та скоротити час, витрачений на написання коду.

Такий підхід допомагає знизити бар'єри входження до програмування, оскільки він не вимагає глибоких знань мов програмування або синтаксичних

правил. Це може забезпечити більш широкий доступ до програмування для непрофесійних розробників, дослідників, студентів та інших зацікавлених осіб.

У цілому, технології NLP та обробки природної мови мають великий потенціал у полегшенні процесу програмування та розробки програмного забезпечення, і вони продовжують розвиватися, привносячи нові можливості та горизонти у цю галузь.

Отже, “Алгоритм синтезу програмного коду мовою JavaScript на основі розпізнавання природних мов” є актуальною темою, яка об'єднує інноваційні підходи до розробки програмного забезпечення з потужністю та гнучкістю мови JavaScript. Дослідження у цій галузі може вести до винаходу нових інструментів та платформ, що революціонізують спосіб, яким буде розроблено програмне забезпечення.

Метою кваліфікаційної роботи є розроблення програмного засобу розпізнавання природньої мови та синтез програмного коду, мовою JavaScript. Для досягнення мети, потрібно виконати такі завдання:

– провести аналіз природніх та структурованих мов, виділити їх основні характеристики та ключові елементи;

– дослідити мови програмування та інтегровані середовища для виявлення найефективніших засобів для реалізації теми кваліфікаційної роботи;

– проаналізувати програмні засоби для розпізнавання природньої мови.

– розглянути існуючі методи та алгоритми для обробки і порівняння природніх мов та обрати найбільш результативний серед них.

– реалізувати алгоритм в рамках певної платформи або середовища розробки.

– провести тестування розробленого алгоритму на наборі прикладів та порівняти результати з іншими існуючими методами та алгоритмами.

За результатами роботи опубліковано тези доповіді на VII науково-практичній конференції «Інтелектуальні комп’ютерні системи та мережі» [1]. Копії публікації наведено у додатку Е

# 1 ПРИРОДНІ ТА ФОРМАЛЬНІ МОВИ КЛАСИФІКАЦІЯ ТА ОСОБЛИВОСТІ ВИКОРИСТАННЯ

## 1.1 Природні та формальні мови

В сучасному світі, комунікація це багатоплановий процес передачі інформації у будь якій формі, яку можна поділити на усну, письмову, та візуальну. Можливість спілкуватися мають всі організми на землі (люди, тварини, рослини, комахи і тд).

Засобом комунікації для людей є мова. Це складна система звуків і знаків, що постійно розвивається, яка використовується для спілкування. Кожному символу в системі надається значення. У процесі спілкування мова виступає посередником між відправником (мовцем/письмом) і одержувачем (слухачем/читачем).

Існує два види мов: природні та формальні. Вважається, що у світі існує близько 7 тисяч природніх мов. У Європі розмовляють лише 230 мовами, в той час як в країнах Азії – понад 2000 мовами.

Англійська мова є найвідоміша з глобальних мов і вона вважається другою мовою для багатьох країн, тому 1/6 жителів Землі – англомовні [7].

Але мова має і інше значення окрім комунікації для людей. Мова є ознакою нації та символом державності. Це невід'ємна органічна цілісність, адже це є творча сила розвитку думки, розвитку духовного життя людини взагальному.

Пунктуація має вирішальне значення для визначення меж речей (коми, крапки, двокрапки) і для визначення деяких аспектів значення (питання знаки оклику, лапки) [8].

Формальні мови, це мови, вигадані людьми задля вирішення специфічних завдань, наприклад створення правил запису математичних формул, хімічної структури речовини, опису алгоритмів і тд [9]. Формальні мови відрізняються від природних мов, оскільки вони є:

– Однозначними, тобто вираз має одне значення незалежно від контексту.

– Виразними, тобто формальна мова повинна бути чіткою та виразною.

– Безпомилковими, тобто орфографічно та пунктуаційно правильними, адже від цього залежить коректність алгоритму, програми, тощо.

Людське розуміння відносної ролі та обчислювальної потужності сканерів, парсерів, регулярних виразів і контекстно-вільних граматик базується на теорії автоматів. У теорії автоматів формальна мова – це набір рядків символів, взятих із кінцевого алфавіту. Формальна мова може бути задана або набором правил (таких як регулярні вирази або контекстно-вільна граматика), які генерують мову, або формальною машиною, яка приймає (розпізнає) мова. Формальна машина приймає рядки символів як вхідні дані та виводить “так” або “ні”. Кажуть, що машина приймає мову, якщо вона каже “так” усім і лише тим рядкам, які є в мові. Крім того, мову можна визначити як набір рядків, для яких певна машина каже “так” [10].

$$f = x_1 \rightarrow x_2 = \overline{x_1}^{\vee} x_2 ,$$
$$x \rightarrow 0 = \bar{x}^{\vee} 0 = \bar{x} ,$$
$$x_1{}^{\vee} x_2 = (x_1 \rightarrow 0) \rightarrow x_2 .$$

Структуровані мови програмування – це формальні мови з чіткою структурою та синтаксисом, що використовуються для вираження алгоритмів та процедур. Головна мета цих мов полягає в полегшенні розробки програм, збільшенні їх зрозумілості та підтримки.

Структуровані мови підтримують різні конструкції керування, такі як послідовне виконання, умовні оператори та цикли. Це дозволяє розробникам організовувати програмний код в логічні блоки, що сприяє зрозумінню та підтримці коду.

Одна з важливих концепцій структурованих мов – це принцип “один вхід, один вихід” (SISO – Single Input, Single Output), що означає, що кожна конструкція керування (така як функція, процедура або блок коду) має точно визначений вхід і вихід [11]. Тобто, для отримання результату виконання

конструкції потрібно передати йому певні вхідні дані, і після обробки цих даних конструкція повертає результат через свій вихід.

Структуровані мови допомагають розробляти програми з кращою структурою та зрозумілістю, спрощують супровід, модифікацію та налагодження коду. Вони також допомагають зменшити можливість виникнення помилок та забезпечують більшу надійність програмного забезпечення.

Формальні мови, як правило, мають строгі правила синтаксису, які керують структурою операторів. Наприклад, у математиці висловлювання 2+2=4 має правильний синтаксис, але 2+=2$4 не має. У хімії $H_2O$ є синтаксично правильною формулою, але $2X_x$ – ні [9].

Синтаксичні правила бувають двох варіантів, що стосуються токенів і структури. Токени – це основні елементи мови, такі як слова, числа та хімічні елементи. Однією з проблем 2+=2$4 є те, що це не коректний маркер у математиці. Так само $2X_x$ не є дійсним, оскільки немає елемента з абревіатурою $X_x$.

Другий тип синтаксичного правила стосується способу поєднання токенів. Рівняння є некоректним, тому що, хоча + і = є законними маркерами, неможливо мати одне право за іншим. Подібним чином у хімічній формулі індекс стоїть після назви елемента, а не перед ним. Як результат, правила синтаксису визначають допустимі способи поєднання токенів у коректні структури мови. Їх дотримання дозволяє використовувати мову зрозуміло та безпомилково.

Формальна мова описуються правилами, які називаються граматикою, що визначають, які символи можуть утворювати валідні рядки мови [12]. Формальна граматика це четвірка G, яка являє собою кортеж з 4 об'єктів.

$$G = \{N, T, P, S\},$$

де, множини $\begin{cases} T - \text{алфавіт термінальних символів} \\ N - \text{алфавіт нетермінальних символів} \end{cases}$,

S – початковий нетермінальний символ, тобто $S \in N$,

P – множини правил (не порожнє).

Точніше кажучи, у формальній мові слова та речення можуть бути складені лише з термінальних символів, які представляються, наприклад, літерами алфавіту. Нетермінальні символи (також відомі як змінні) відповідають певним поняттям у мові.

Ці правила мають вигляд $\alpha \rightarrow \beta$, де $\alpha$, $\beta$ – рядки об'єднання термінальних та нетермінальних символів($T \cup N$) при цьому є деякі обмеження: $\alpha$, містить у собі, хоча б один нетермінал. Це обмеження забезпечує, що граматика буде здатна досягти та генерувати речення з термінальних символів шляхом послідовної заміни нетерміналів [12].

Різниця між природними і формальними мовами виявляється у їх призначенні, структурі та використанні. Природні мови призначені для міжособистого спілкування, тоді як формальні мови використовуються для точного вираження інформації в конкретних областях, де важлива точність і безперечність вираження

Варто зазначити, що природні та формальні мови не виключають одна одну, але використовуються у різних контекстах. Люди використовують природні мови для повсякденного спілкування, але також можуть використовувати формальні мови для специфічних технічних або наукових цілей.

## 1.2 Мови програмування

Мова програмування – це формальний набір правил і символів, які використовуються для створення програм і скриптів, які керують роботою комп'ютера або іншого пристрою. Вона дає розробникам можливість створювати команди, які комп'ютер може розуміти і виконувати. Ці команди можуть включати операції зі змінними, обчислення, умовні вирази, цикли і взаємодію зі зовнішнім середовищем, таким як зчитування та запис даних [13,14].

Одна з важливих переваг мов програмування полягає у їх стандартизації та переносимості. Це означає, що програми, написані на певній мові програмування, можуть бути виконані на різних комп'ютерах та операційних системах, якщо вони підтримують відповідну мову.

Стандартизація мов програмування зазвичай відбувається через визначення мовних стандартів, які встановлюють правила та синтаксис мови. Це забезпечує єдність та узгодженість в способі написання програм на цій мові [15]. Такі стандарти зазвичай підтримуються виробниками компіляторів або інтерпретаторів для різних платформ.

Переносимість означає, що програми, розроблені на певній мові програмування, можуть бути виконані на різних платформах без необхідності значних змін або перекомпіляції [16]. Це забезпечує більшу гнучкість та ефективність розробки програмного забезпечення, оскільки розробникам не потрібно створювати окремі версії програм для кожної платформи.

Проте, варто відзначити, що хоча багато мов програмування є стандартизованими та переносимими, існують також певні варіації та розширення мов, які можуть бути специфічними для певних платформ або реалізацій. Також, можуть виникати проблеми з переносимістю, якщо програма використовує платформозалежні функції або особливості конкретної операційної системи.

Все ж таки, загальна стандартизація та переносимість мов програмування дозволяють програмістам писати програми, які можуть працювати на різних платформах і зменшують залежність від конкретного обладнання або операційної системи.

Семантика і синтаксис – це два важливі аспекти мов програмування. Семантика мови програмування визначається правилами, за якими виконуються оператори та конструкції мови [17]. Наприклад, деякі мови мають автоматичне управління пам'яттю, де ресурси виділяються та звільняються автоматично, що полегшує роботу програміста. Інші мови вимагають вручну виділяти та звільняти пам'ять, що може бути складним та призводити до помилок, але надає більшу контроль та ефективність.

Синтаксис мови програмування визначає правила, за якими код повинен бути написаний для коректного виконання. Різні мови мають свої власні правила синтаксису та стиль написання коду. Наприклад, деякі мови вимагають закінчення кожного оператора крапкою з комою, тоді як інші мови не потребують цього [18]. Також, різні мови можуть мати свої власні конструкції та ключові слова для виконання певних завдань.

Важливо враховувати ці різниці при вивченні та використанні різних мов програмування, оскільки правила та стиль написання коду можуть відрізнятися і впливати на розуміння та взаємодію з кодом.

Всі існуючі мови програмування здебільшого можна розділити на дві групи (рисунок 1.1):

– мови низького рівня;

– мови високого рівня.

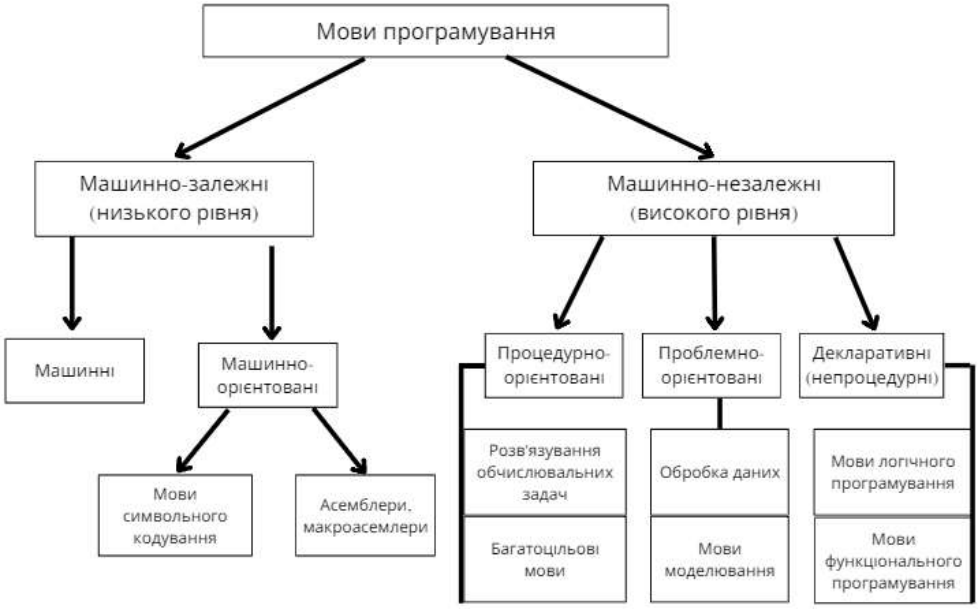


Рисунок 1.1 – Класифікація мов програмування

Мова низького рівня – це мова, близька до машинної мови. Вона схожа на “побітовий код”, який компілюється для мов високого рівня.

Програма, з якою працює процесор, є машинним кодом, яка складається з послідовності чисел. Такий запис містить лише номери команд процесора, необхідні дані та адреси комірок пам’яті. Наприклад (для зручності двійкова

система записується у шістнадцятковій системі, де два символи відповідають одному байту даних – шістнадцяткова система числення є досить популярною у програмуванні): BD 31 41 09 1D AA F4 0E 8A 37 C3 CD 40 F2 A9 BD 23 18 66 6D 6F [19].

До мов низького рівня відносяться машинні коди і мови асемблера. На рисунку 1.2, зображено кілька прикладів. Низькорівнева мова програмування – це мова, в якій її інструкції мають прямий контроль над апаратним забезпеченням, і тому їхня функція залежить від фізичної структури комп'ютерів, які їх підтримують [20].

Використання терміну "низький" не означає, що ця мова менш важлива, ніж мова високого рівня; це просто означає, що між мовою та апаратним забезпеченням менше абстракції.

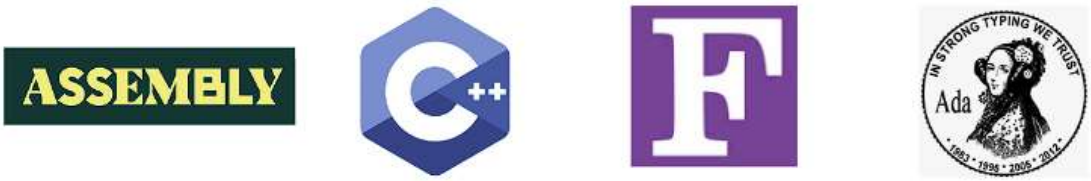


Рисунок 1.2 – Приклади низькорівневих мов

– Мова асемблера – це мова програмування низького рівня, яка використовує мнемоніку для представлення машинних інструкцій. Код мови асемблера перетворюється на машинну мову асемблером.

– C і C++ вважаються мовами програмування високого рівня, але вони забезпечують низькорівневий доступ до апаратних ресурсів комп'ютера, що робить їх популярними для системного програмування та розробки вбудованих систем.

– FORTRAN (Formula Translation) – це мова програмування низького рівня, яка спочатку була розроблена для наукових та інженерних програм. Вона все ще широко використовується сьогодні для високопродуктивних обчислювальних програм.

– Ada – це мова програмування низького рівня, розроблена спеціально для вбудованих систем і програм реального часу. Вона забезпечує прямий доступ до апаратних ресурсів комп’ютера, що робить її хорошим вибором для завдань програмування низького рівня.

Створення таких мов спрямоване на можливість користувачам писати ефективні програми, які враховують та використовують структуру обчислювальних машин або особливості конкретної машини.

Мова високого рівня – це будь-яка мова програмування, яка дозволяє розробляти програми в більш зручному для користувача програмуванні контексті та, загалом, не залежить від апаратної архітектури комп'ютера.

Вона використовує більш високий рівень абстракції від комп'ютера та більше уваги виділяє логіці програмування, а не основним апаратним компонентам, таким як адресація пам'яті та використання регістрів [21].

Високорівневі мови (рисунок 1.3) призначені для використання людиною-оператором або програмістом. Їх називають “ближчими до людини”. Іншими словами, їхній стиль програмування та контекст легше вивчити та реалізувати, ніж мови низького рівня, і весь код, як правило, зосереджений на конкретній програмі, яку потрібно створити [14].

ЕОМ (Електронна Обчислювальна Машина) працює з машинною мовою, яка складається з набору низькорівневих команд, зрозумілих для апаратної частини комп'ютера. Тому перед виконанням програми, написаної на високорівневій мові, необхідно її перетворити на машинний код. Цей процес виконується за допомогою інтерпретаторів або компіляторів. Інтерпретатори зчитують високорівневий код по одній команді за раз, перетворюють його на відповідний машинний код та виконують цю команду безпосередньо. Компілятори перетворюють весь високорівневий код на машинний код в одному процесі, що дозволяє оптимізувати виконання програми [22]. Отриманий машинний код виконується ЕОМ, яка розуміє ці команди та виконує відповідні операції. Таким чином, інтерпретатори та компілятори допомагають зрозумілій для програмістів високорівневій мові перетворитись на низькорівневий машинний код, який може бути виконаний ЕОМ.


Рисунок 1.3 – Приклади мов високого рівня

– Python – популярна мова програмування високого рівня, відома своєю простотою, читабельністю та універсальністю. Вона зазвичай використовується в веб-розробці, аналізі даних, штучному інтелекті та наукових обчисленнях [23].

– Java – це мова програмування високого рівня, яка широко використовується для розробки веб-додатків і мобільних додатків, а також для створення корпоративних додатків і програмного забезпечення на стороні сервера.

– C# (вимовляється як “сі-шарп”) – мова програмування високого рівня, розроблена Microsoft. Він зазвичай використовується для створення настільних додатків Windows, відеоігор і веб-додатків на платформі .NET [24].

– Ruby – це мова програмування високого рівня, відома своєю простотою та продуктивністю. Він зазвичай використовується для веб-розробки та створення програм на стороні сервера.

– JavaScript – це мова програмування високого рівня, яка використовується для створення динамічних веб-додатків і веб-браузерів [25]. Вона також використовується для створення програм на стороні сервера та мобільних додатків з використанням фреймворків, таких як Node.js і React Native.

– PHP – мова сценаріїв на стороні сервера, яка використовується для веб-розробки. Він зазвичай використовується для створення систем керування вмістом, веб-сайтів електронної комерції та інших динамічних веб-додатків.

Середовище розробки (IDE – Integrated Development Environment) – це програмне забезпечення, яке надає розробникам зручний інтерфейс для написання, відлагодження, тестування та керування програмним кодом. Воно забезпечує інтеграцію різних інструментів та функціональностей, які

полегшують процес розробки програмного забезпечення [26]. Існує велика кількість IDE, наприклад, PyCharm [27], Visual Studio Code [28], тощо. В порівнянні з блокнотом або Microsoft Word, вони спеціалізуються на розробках програм, мовою програмування, підтримують автодоповнення коду, візуальне налагодження, інтегровану систему керування версіями, вбудовану підтримку віртуальних середовищ та інші корисні інструменти, які полегшують процес розробки.

Для досліждення теми “Алгоритм автоматизованого синтезу програмного коду мовою JavaScript” було обрано дві мови програмування, таких як Python та JavaScript, оскільки вони є популярні, зручні та прості в використанні. Наявність інструментів та бібліотек, які полегшують досліження та дозволяють аналізувати та маніпулювати синтаксисом є беззаперечним плюсом даних мов.

Також обрано 2 середовища розробки для кожних з мов, PyCharm та VSCode, оскільки вони є зручними, безкоштовними та використовуються з обраними вище мовами програмувань.

## 1.3 Програмні засоби розпізнавання природньої мови

Такі домашні імена, як Echo (Alexa), Siri та Google Translate, мають принаймні одну спільну рису. Усі вони є продуктами застосування обробки природної мови (NLP), що відноситься до набору технік, які включають застосування статистичних методів, з або без розуміння лінгвістики, для розуміння тексту заради вирішення реальних завдань [29-31].

Існує декілька популярних прикладів програмних засобів, які розпізнають та оброблюють природню мову:

– NLTK (Natural Language Toolkit): NLTK є одним з найпопулярніших інструментів NLP. Він надає широкий спектр функцій для обробки тексту, токенізації, виокремлення частин мови, синтаксичного аналізу та інших завдань NLP. Але інструмент може бути повільним для обробки великих обсягів даних, і

деякі його функції, наприклад розпізнавання української мови, вимагають додаткових зусиль для токенізації українських текстів [32].

– SpaCy: SpaCy є швидким і ефективним інструментом для обробки природної мови. Він має широкі можливості для розпізнавання частин мови, залежностей, назвних сутностей та інших аспектів NLP, але він не має вбудованої моделі для української мови, тому потрібно створювати власний модуль та запускати процес навчання на ньому.

– Google Cloud Natural Language API: Цей API від Google надає можливість розпізнавання природної мови в реальному часі. Він може аналізувати текст для виявлення сутностей, відносин між ними, категорій та інших аспектів, але вимагає налаштування облікового запису та може мати вартість в залежності від обсягу використання.

– SpeechRecognition: SpeechRecognition є пакетом, який надає простий інтерфейс для розпізнавання мовлення з різних джерел, включаючи аудіофайли та мікрофон. Він підтримує кілька розпізнавачів української мови, таких як Google Speech Recognition, Sphinx та інші [33-35].

Серед усіх перелічених інструментів, варто виокремити SpeechRecognition, оскільки це найбільш ефективний пакет для теми кваліфікаційної роботи.

Розпізнавання тексту в основному виходить шляхом перетворення текстів у придатні для використання обчислювальні представлення, які є дискретними або безперервними комбінаторними структурами, такими як вектори або тензори, графи та дерева [36].

Розпізнавання природної мови (Natural Language Processing – NLP) – це галузь комп'ютерної науки, яка займається обробкою та аналізом людської мови. В останні роки, з відкриттям нових технологій та розвитком штучного інтелекту, NLP стала дедалі більш важливою для розвитку програмних засобів та прикладних систем.

Прогрес у розвитку мовних технологій став можливим завдяки зниженню вартості обчислювальних ресурсів та застосуванню нейронних мереж для вирішення різних завдань. Останні досягнення в галузі машинного навчання

знаходять широке застосування в розпізнаванні мови, що призводить до високої точності в цій сфері. Прибутки від NLP показані на рисунку 1.4.

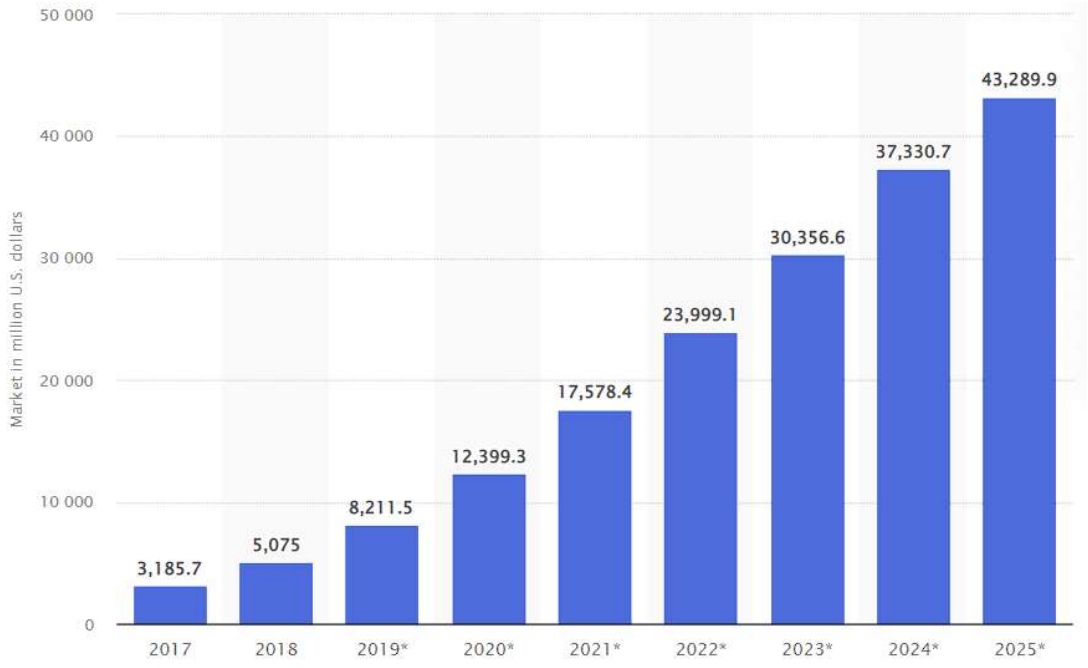


Рисунок 1.4 – Прибутки від NLP з 2017 по 2025 рік

Одним із ключових напрямків NLP є розпізнавання природньої мови, яке має на меті розуміння та інтерпретацію людської мови комп'ютером [37,38]. Це включає в себе відповіді на запитання, автоматичний переклад, виявлення емоцій та інші застосування.

Приклади використання NLP:

– Автоматичний переклад мови. NLP може бути використана для автоматичного перекладу тексту з однієї мови на іншу. Для цього використовуються спеціальні алгоритми та моделі машинного навчання, які можуть розпізнавати та перекладати текст.

– Розпізнавання та інтерпретація запитань. NLP може бути використана для розуміння та інтерпретації запитань, які вводяться користувачами, та надання відповідей на ці запитання. Це може бути корисно для різних застосувань, таких як веб-пошук, відповіді на запитання користувачів, та інше.

– Аналіз тексту та виявлення емоцій. NLP може бути використана для аналізу тексту та виявлення емоцій, які відображені в цьому тексті. Це може бути

корисно для аналізу соціальних мереж, оглядів продуктів та послуг, та інших сфер.

– Голосовий інтерфейс. Застосування NLP дозволяє використовувати голосовий інтерфейс, що дозволяє користувачам взаємодіяти з комп'ютером, використовуючи голосові команди. Це особливо корисно для осіб з обмеженою рухливістю.

– Автоматичне зведення тексту. NLP можна використовувати для автоматичного зведення тексту, тобто створення короткого опису тексту або статті. Це може бути цінним для отримання загального уявлення про текст без необхідності читати його повністю.

– Аналіз настрою та емоцій. NLP може бути використана для аналізу настрою та емоцій в тексті. Це допомагає виявити відображені емоції, такі як позитивність чи негативність, що містяться в тексті.

– Автоматична класифікація тексту. NLP може застосовуватись для автоматичної класифікації тексту на основі його змісту. Це корисно для групування текстів за тематикою або жанром.

– Генерація тексту. Застосування NLP дозволяє автоматично генерувати текст, наприклад, для створення новинних статей або контенту для веб-сайтів.

– Автоматичне виявлення іменованих сутностей. NLP може використовуватись для автоматичного виявлення іменованих сутностей, таких як імена людей, місця та організації, у тексті. Це допомагає виявляти ключові слова та поліпшувати розуміння змісту тексту.

Засоби розпізнавання природньої мови дозволяють комп'ютерам розуміти та інтерпретувати людську мову в різних контекстах. Вони використовують алгоритми та моделі, які автоматично визначають мовні структури, витягують семантичну інформацію, розпізнають іменовані сутності, класифікують текст та здійснюють машинний переклад та інші завдання [39].

Використання програмних засобів розпізнавання природньої мови може значно полегшити та покращити обробку текстової інформації, забезпечуючи швидке та точне розуміння тексту. Продовження розвитку та удосконалення

таких засобів може привести до подальших проривів у розумінні та взаємодії між комп'ютерами та людьми, відкриваючи нові можливості в різних галузях, включаючи штучний інтелект, автоматизацію та обробку мовної інформації.

## 1.4 Постановка завдань кваліфікаційної роботи

– Провести аналіз природніх та структурованих мов, виділити їх основні характеристики та ключові елементи.

– Дослідити мови програмування та інтегровані середовища для виявлення найефективніших засобів для реалізації теми кваліфікаційної роботи.

– Проаналізувати програмні засоби для розпізнавання природньої мови.

– Розглянути існуючі методи та алгоритми для обробки і порівняння природніх мов та обрати найбільш результативний серед них.

– Реалізувати алгоритм в рамках певної платформи або середовища розробки.

– Провести тестування розробленого алгоритму на наборі прикладів та порівняти результати з іншими існуючими методами та алгоритмами.

# 2 МЕТОДИ ТА АЛГОРИТМИ РОЗПІЗНАВАННЯ ПРИРОДНІХ МОВ

## 2.1 Алгоритми розпізнавання природніх мов

Алгоритм – це детальний інструктивний набір кроків, який описує послідовність дій, необхідних для вирішення певної задачі або досягнення певної мети [40].

Алгоритми та структури даних є найбільш фундаментальними поняттями в обчислювальній техніці. Вони є основними будівельними блоками, з яких будується складне програмне забезпечення. Розуміння цих основоположних концепцій є надзвичайно важливим у розробці програмного забезпечення, і це включає наступні три характеристики:

– Як алгоритми маніпулюють інформацією, що міститься в структурах даних.

– Як дані впорядковані в пам'яті.

– Які характеристики продуктивності окремих структур даних [41,42].

Існують декілька видів алгоритмів розпізнаванняя природніх мов, наприклад:

– Байєсовський класифікатор є статистичним алгоритмом, який використовує теорему Байєса для визначення ймовірності належності об'єктів до певного класу на основі їх ознак. Він отримав свою назву на честь Томаса Байєса, англійського математика, який розробив цю теорему. Байєсовський класифікатор використовує статистичні моделі для врахування апріорної ймовірності належності об'єкта до класів та умовної ймовірності спостережуваних ознак при даному класі. За допомогою теореми Байєса, класифікатор розраховує апостеріорну ймовірність належності об'єкта до кожного класу і обирає клас з найвищою ймовірністю як прогнозований клас для об'єкта [43].

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}.$$

– Він є ефективним інструментом для автоматичної класифікації об'єктів, таких як текстові документи, електронні листи, зображення, медичні дані та інші. Байєсовський класифікатор також широко використовується в спам-фільтрах, системах розпізнавання мови, системах рекомендацій та інших задачах аналізу даних.

– Метод найближчих сусідів (рисунок 2.1) (k-Nearest Neighbors, k-NN): заснований на принципі схожості. Він класифікує нові тексти, порівнюючи їх з набором навчальних даних та вибираючи k найближчих сусідів. В задачі розпізнавання мови, алгоритм навчається на заздалегідь підготовленому наборі текстів, де кожен текст має відповідну мовну мітку. Застосовуючи метод найближчих сусідів, новий текст може бути класифікований шляхом порівняння його з найближчими текстами з навчального набору за допомогою певної метрики відстані (наприклад, косинусної відстані). Класифікація здійснюється шляхом врахування більшості класів найближчих сусідів. Наприклад, якщо більшість найближчих сусідів належить до англійської мови, то новий текст буде віднесений до класу англійської мови [44].

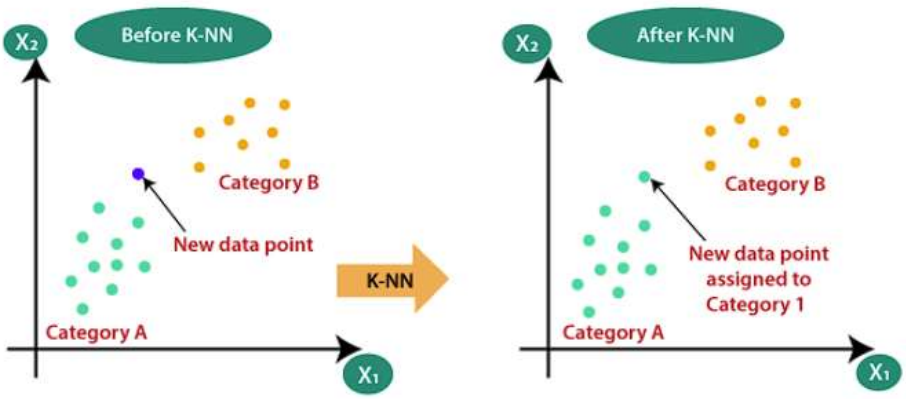


Рисунок 2.1 – Примітивний приклад роботи k-NN

– Підхід заснований на правилах (рисунок 2.2): використовує набір правил та шаблонів, щоб визначити граматичні структури або смислові відношення в тексті. У цьому підході, текстові дані аналізуються системою відповідно до заданих правил та умов для здійснення класифікації, витягування інформації або виконання певних дій. Правила формуються на основі лінгвістичних знань,

експертного досвіду або доменних знань про конкретну проблему. Вони можуть бути виражені у форматі “якщо-тоді” або у вигляді логічних правил, що визначають, які дії слід здійснювати на підставі певних умов.

Процес роботи з методом, заснованим на правилах, включає такі етапи:

- Побудова правил: Експерти або лінгвісти створюють правила, використовуючи свій досвід та знання про мову. Ці правила можуть включати шаблони, ключові слова, граматичні правила, синтаксичні залежності та інші лінгвістичні особливості.
- Застосування правил: Система використовує визначені правила до вхідних текстових даних. Вона аналізує текст та порівнює його з правилами для здійснення класифікації або витягування інформації. Наприклад, якщо текст містить певне ключове слово або фразу, система може призначити відповідний тег або клас.
- Післяобробка результатів: Після застосування правил система може виконати додаткові операції для покращення результатів, такі як фільтрація шуму, корекція помилок або об'єднання рішень з різних правил для отримання більш точного виходу.

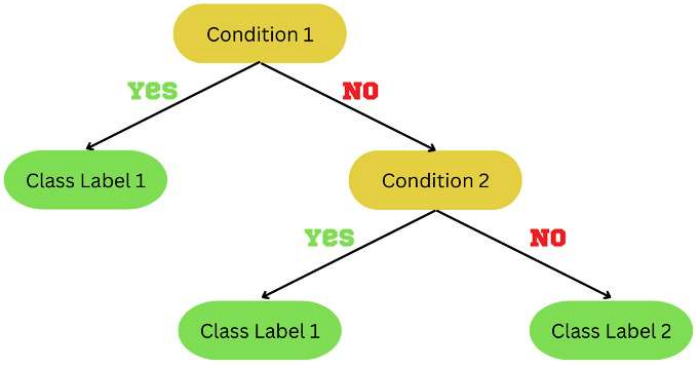


Рисунок 2.2 – Метод заснований на правилах

Метод, заснований на правилах, має свої переваги та обмеження. Він може бути ефективним у випадках, коли правила можуть чітко описати лінгвістичні аспекти аналізованого тексту. Однак, цей метод може бути обмеженим у

розробці та підтримці правил для широкого спектру мовних випадків і може вимагати постійного оновлення у разі змін у мові або контексті аналізу.

– Моделі на основі уваги (рисунок 2.3): Моделі на основі уваги базуються на ідеї, що певні частини вхідного тексту є більш важливими для розуміння тексту, ніж інші. Ці моделі надають більший ваговий коефіцієнт (увагу) на ці важливі частини тексту, що дозволяє моделі фокусуватися на релевантних деталях та залежностях.

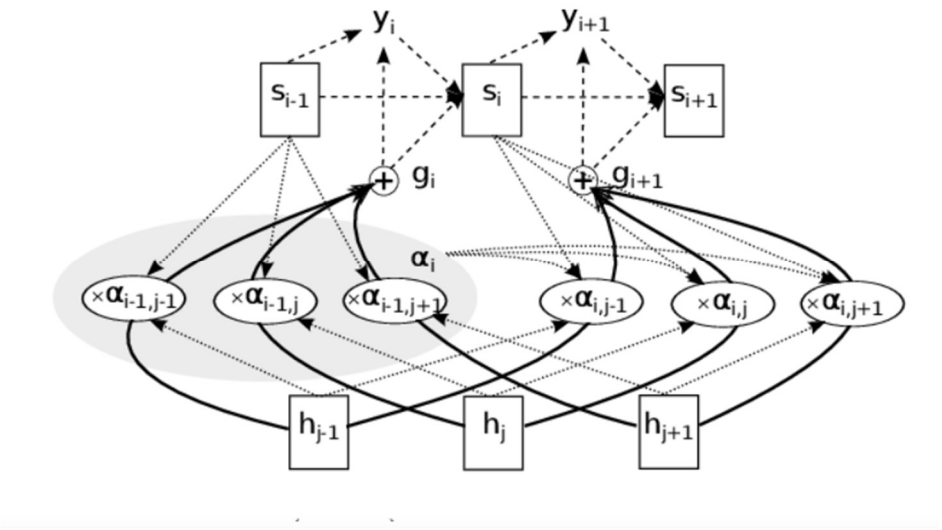


Рисунок 2.3 – Архітектура моделі на основі уваги

При розробці алгоритму синтезу природних мов, було обрано бібліотеку “Fuzzy Wuzzy”, що не ґрунтується на моделі машинного навчання, але використовує алгоритми, такі як розташування Левенштейна, відстань Джаро, алгоритм співвідношення для нечіткого співставлення рядків [45]. Нечітке зіставлення рядків – це техніка пошуку рядків, які максимально відповідають заданому шаблону [1]. Тобто, нечітке зіставлення рядків – це процес пошуку максимально подібних відповідників. При цьому, некоректно введені або частково введені слова є достатніми для знаходження відповідника [46]. Бібліотека “Fuzzy Wuzzy” є потужним інструментом для порівняння та аналізу текстових рядків у Python. Вона дозволяє швидко і зручно виконувати операції порівняння, виправлення помилок і пошуку найбільш схожих елементів, що дає

змогу покращити точність і ефективність багатьох завдань обробки тексту і аналізу даних.

## 2.2 Структури мови програмування JavaScript

Алгоритм синтезу природньої мови полягає у перетворенні голосових команд на конкретні інструкції, використовуючи мову програмування JavaScript. Для виконання цих інструкцій потрібно ознайомитися зі структурами мови програмування JavaScript.

JavaScript є мультипарадигмовою мовою програмування, яка підтримує об'єктно-орієнтований, імперативний та функціональний стилі.

Усі мови програмування працюють, перетворюючи англомовний синтаксис на машинний код, який виконує операційна система. JavaScript можна віднести до категорії скриптових або інтерпретованих мов [47]. Код JavaScript інтерпретується безпосередньо ядром JavaScript, що означає, що він перекладається в машинний код без окремого етапу компіляції, як це відбувається у деяких інших мовах програмування. Таким чином, всі скриптові мови є мовами програмування, але не всі мови програмування є скриптовими.

JavaScript відповідає специфікації ECMAScript, що означає, що він відповідає стандарту програмування загального призначення, прийнятому в усьому світі (ECMAScript 2020). Хоча JavaScript іноді називають ECMAScript, це був батьківський стандарт, який Брендон Айх використав у 1995 році для створення цієї мови для Netscape Navigator. Початкова кодова назва JavaScript була Mocha, була запущена як LiveScript, а потім перейменована на JavaScript. Microsoft випустила власну подібну версію під назвою Jscript, також засновану на стандарті ECMAScript, щоб уникнути конфлікту торгових марок [48].

JavaScript має різноманітні структури, які дозволяють організовувати та керувати виконанням програмного коду. Основними структурами мови програмування JavaScript є:

– Інструкції (рисунок 2.4): Інструкції виконуються послідовно в порядку їх написання. Це можуть бути прості вирази, оператори присвоєння, умовні оператори (if-else, switch), цикли (for, while, do-while) та інші.

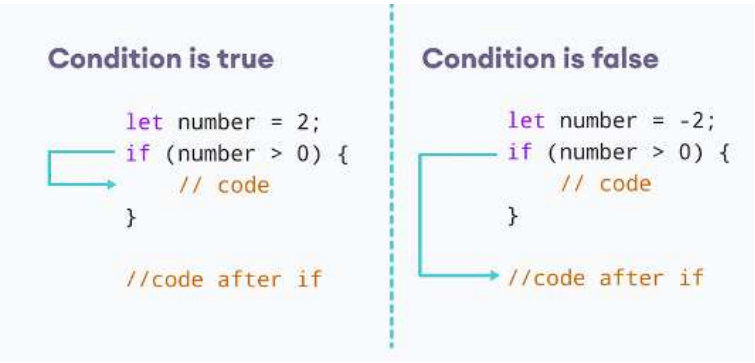


Рисунок 2.4 – Інструкція в JavaScript

– Функції (рисунок 2.5): Функції – це блоки коду або підпрограми, які виконують певні завдання і можуть бути викликані з інших частин програми. Вони дозволяють організувати код в більш логічні і модульні частини, сприяючи повторному використанню коду і полегшуючи його розробку та обслуговування [49]. Вони можуть приймати аргументи (вхідні параметри), виконувати певну логіку або обчислення і повертати результат (вихідні значення). Вони також можуть мати доступ до змінних, що визначені у вищому контексті (глобальні змінні) або передані як аргументи.

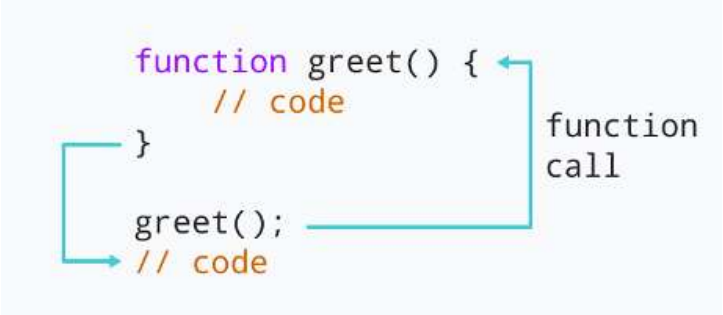


Рисунок 2.5 – Функція в JavaScript

– Об'єкти (рисунок 2.6): Об'єкти є основними структурами даних в JavaScript. Вони дозволяють збирати дані та функції разом в одному контейнері.

Об'єкти можуть мати властивості (properties) та методи (methods), які визначають їх характеристики та поведінку.

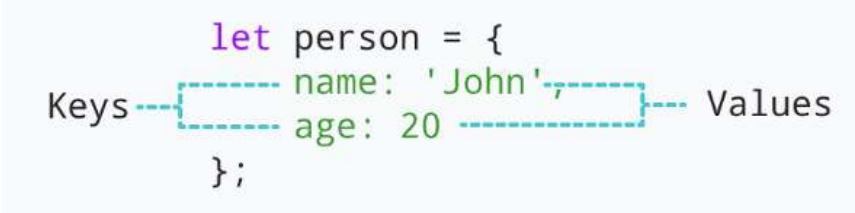


Рисунок 2.6 – Об’єкт в JavaScript

– Масиви (рисунок 2.7): Масиви є упорядкованими списками значень. Вони дозволяють зберігати та керувати багатьма значеннями за допомогою індексів. Елементи масиву зберігаються у пам'яті послідовно, і до них можна отримати доступ, використовуючи їх індекси. Індекси масиву починаються з 0 для першого елемента, 1 для другого елемента і так далі.

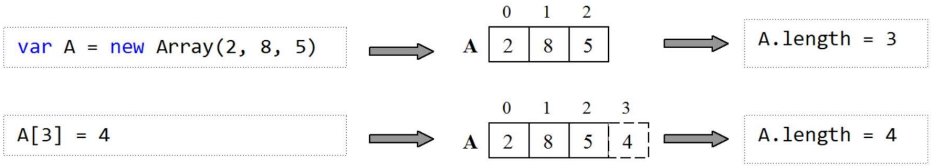


Рисунок 2.7 – Масив в JavaScript

– Класи: В JavaScript з'явилися класи зі стандартом ECMAScript 2015 (ES6). Класи дозволяють оголошувати об'єктні типи та визначати їх властивості та методи [50].

```
class Person {
    constructor(name,age){
          this.name = name;
          this.age = age;
    }

    sayHello(){
          console.log(`Hello, my name is ${this.name}`);
    }
}
```

Це лише кілька прикладів структур мови програмування JavaScript. JavaScript є дуже гнучкою мовою, яка надає багато можливостей для організації та виконання програмного коду.

У завданні кваліфікаційної роботи, було використано декілька структур даних на мові JS, такі як масиви, інструкції, функції.

## 2.3 Алгоритм синтезу програмного коду мовою JS на основі розпізнавання природніх мов

Алгоритм побудований на функціональній та інструкційній структурах. Для користувачів з обмеженим функціонуванням голосового апарату, у таких випадках можна розглянути використання текстового вводу для аналізу і розпізнавання природної мови. Замість використання голосового вводу, користувач може набрати свої команди або запити на клавіатурі або за допомогою інших текстових введених пристроїв.

Покрокове виконання розробленого алгоритму:

Крок 1. Запуск програми.

Крок 2. Вибір типу вхідних даних користувачем.

Крок 3. Якщо “голосовий” то type = 1, якщо “текстовий” то type = 0.

Крок 4. Якщо type =1, введення інструкцій за допомогою голосової команди.

Крок 5. Якщо type=0, введення інструкцій за допомогою текстової команди.

Крок 6. Умова перетворення природньої мови, якщо розпізнано, то перехід до кроку 7, якщо не розпізнано, перехід до кроку 12.

Крок 7. Вхід в незкінченний цикл.

Крок 8. Нечітке співставлення слів з ключовими словами.

Крок 9. Якщо співставлення знайдено то перехід до кроку 10, якщо ж ні то перехід до кроку 12.

Крок 10. Функція ситезу коду, за певними правилами кодогенерації.

Крок 11. Вивід результату синтезу програмного коду.

Крок 12. Питання про повторну спробу.

Крок 13. Відповідь-команда користувача, за допомогою:

– Голосового введення, якщо type = 1.

– Текстового введення, якщо type = 0.

Крок 14. Якщо “так”, то крок 4, якщо “ні”, то крок 15.

Крок 15. Вихід з циклу і закінчення роботи алгоритму та програмного додатку.

Розглянувши даний алгоритм синтезу програмного коду мовою JS на основі розпізнавання природніх мов можна виділити його переваги та недоліки.

Переваги:

– Автоматизація і спрощення розробки: Алгоритм автоматизує процес розробки програмного коду, що дозволяє економити час і зусилля розробника.

– Зменшення ймовірності помилок: Використання алгоритму дозволяє уникнути людських помилок при введенні інструкцій, оскільки вони можуть бути розпізнані автоматично. Це сприяє покращенню якості та надійності коду, що генерується.

– Зручність для користувача: Завдяки можливості вводити команди голосом або текстом, алгоритм стає зручним і доступним для широкого кола користувачів. Він може полегшити взаємодію з програмою та забезпечити зручний інтерфейс для введення команд.

– Покращення продуктивності: Завдяки автоматизації та спрощенню процесу розробки, алгоритм може покращити продуктивність розробників. Вони можуть швидше створювати код і експериментувати з різними реалізаціями, що дозволяє зосередитись на більш складних завданнях.

– Потенційна можливість удосконалення: Розвиток технологій розпізнавання мови, обробки природної мови та генерації коду може зробити алгоритм ще більш потужним і ефективним.

Недоліки:

– Обмежена точність розпізнавання природної мови: Існуючі технології розпізнавання природної мови можуть мати обмежену точність. Це може

призводити до помилок в інтерпретації команд користувача або неправильного перетворення мови на код.

– Вразливість до невідомих команд і контекстів: Алгоритм може мати обмежений набір команд та ключових слів, що розпізнаються. Якщо користувач надає невідому команду або вводить інструкцію у контексті, який не був передбачений, алгоритм може не правильно розпізнати або обробити таку ситуацію.

– Залежність від якості вхідних даних: Якість вхідних даних, які надає користувач, може суттєво впливати на роботу алгоритму. Якщо команда користувача надана недостатньо чітко або неправильно сформульована, алгоритм може не здатися її розпізнати або правильно обробити.

Взагалом, в майбутньому, алгоритм може здійснити удосконалення функціоналу, наприклад як:

– Покращення точності розпізнавання мови: Одним із головних напрямків удосконалення буде покращення точності розпізнавання мови за допомогою використання більш потужних алгоритмів машинного навчання, які здатні краще розпізнавати природну мову та враховувати контекст.

– Розширення гнучкості генерації коду: Одним із недоліків описаного алгоритму є вразливість до невідомих команд і контекстів. Для його удосконалення можна використати додаткові правила кодогенерації, які дозволяють генерувати більш різноманітний і структурований код.

– Підтримка більшої кількості мов програмування: Наразі алгоритм орієнтований на генерацію коду мовою JavaScript. Однак, для більшого розмаїття застосувань, було б корисно розширити його можливості до підтримки інших мов програмування, таких як Python, Java, C++ тощо. Це дозволило б розробникам мати більший вибір та зручність у використанні алгоритму.

– Розробка інтуїтивного інтерфейсу користувача: Важливою складовою удосконалення алгоритму є розробка інтуїтивного та зручного інтерфейсу користувача. Це дозволить користувачам легко взаємодіяти з алгоритмом, вводити інструкції та отримувати зрозумілі результати. Інтерфейс може

включати в себе графічний або голосовий ввід, можливість налаштування параметрів та перегляду згенерованого коду.

– Тестування та зворотний зв'язок користувачів: Важливим етапом удосконалення буде проведення тестування алгоритму з реальними користувачами та збір їх зворотного зв'язку. Це дозволить виявити проблеми та слабкі місця алгоритму, а також отримати цінні пропозиції щодо його поліпшення.

Алгоритм автоматизованого синтезу програмного коду може мати широкі застосування в різних галузях, де швидкий та ефективний синтез програмного коду може покращити продуктивність та зменшити час розробки, наприклад:

– Розробка веб-додатків: Алгоритм може бути корисним для розробки веб-додатків, де швидкий синтез програмного коду може допомогти зменшити час розробки та полегшити процес створення функціональних веб-сторінок і додатків.

– Прототипування: Завдяки швидкому синтезу програмного коду, алгоритм може використовуватися для швидкого створення прототипів додатків або функцій. Це дозволяє розробникам швидко експериментувати з ідеями та перевіряти їх життєздатність без великих зусиль.

– Автоматизація рутинних завдань: Алгоритм може бути використаний для автоматизації рутинних завдань, які вимагають написання однотипного програмного коду. Наприклад, автоматичне генерування скриптів для обробки даних, генерація шаблонного коду або автоматичне створення тестових сценаріїв.

– Навчання програмуванню: Алгоритм може використовуватися для навчання програмуванню, особливо початківцям. Шляхом введення команд голосом або текстом, користувачі можуть спостерігати за згенерованим кодом і вивчати основні принципи програмування та структури коду.

# 3 ПРОГРАМНИЙ ДОДАТОК СИНТЕЗУ ПРОГРАМНОГО КОДУ МОВОЮ JAVASCRIPT

## 3.1 Структура програмного додатку

В ході реалізації алгоритму було використано 2 мови програмування Python та JavaScript. Python використовувався, як основа мова програмування для програмування алгоритму, яка несе у собі необхідні бібліотеки, а саме “Fuzzy Wuzzy” та “Speech Recognition” (рисунок 3.1).

```
pip install SpeechRecognition , fuzzywuzzy
```

Рисунок 3.1 – Встановлення в програму додаткових бібліотек

JavaScript використовувався як мова програмування для тестування алгоритму синтезу програмного додатку. Додатково було встановлено NodeJs, задля того, щоб вивести результат програмного додатку безпосередньо на серверній частині програми, без необхідності використання браузера. Задля відслідковування та оновлення результатів, при змінні вхідних інструкцій користувача, була додана утиліта Nodemon (рисунок 3.2), за допомогою якої створилась можливість автоматичного перезапуску програми NodeJs щоразу, коли у вихідному коді виявляються зміни та усунення необхідності вручну зупиняти та перезапускати сервер кожного разу, коли відбулись зміни у коді, таким чином підвищуючи ефективність розробки.

```
npm install nodemon
```

Рисунок 3.2 – Встановлення утиліти Nodemon

Програмний додаток складається з 7 файлів (рисунок 3.3 ), в який входить основний файл, вихідний файл, база даних та 4 модулі.

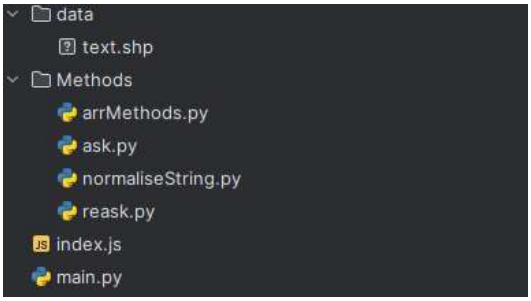


Рисунок 3.3 – Структура програмного додатку

– text.shp – база даних, яка містить в собі розпізнаний текст голосової команди користувача;

– arrMethods.py – модуль з набором внутрішніх 6 функцій, кожна з яких, перетворює текст на синтаксис мови JavaScript за попередньо встановленому шаблону, наприклад:

```
str = '12,3423,24,5665,3456,234,1'
js_code = 'let myArray = [];\n'
nums = str.split(",")
for num in nums:
    js_code += 'myArray.push(' + num + ');\n'
js_code += "console.log(myArray)\n"
print(js_code)
```

Результатом виконання даного шаблону є синтезований синтаксис створення масиву в JavaScript у вигляді змінної (рисунок 3.4).

```
let myArray = [];
myArray.push(12);
myArray.push(3423);
myArray.push(24);
myArray.push(5665);
myArray.push(3456);
myArray.push(234);
myArray.push(1);
console.log(myArray)


Process finished with exit code 0
```

Рисунок 3.4 – Результат синтезу програмного коду по шаблону створення масиву

– normaliseString.py – окремий модуль, який несе у собі функцію, для форматування текстової команди користувача згідно обраної моделі, а саме приведення усіх символів до нижнього регістру та заміни символу “.” на “,”;

– ask.py – модуль для записування, зчитування та трансформації аудіофайлу в текстовий формат за допомогою бібліотеки “SpeechRecognition”;

– reask.py – блок управління резервного голосового запиту та керування завершенням роботи програми;

– index.js – вихідний файл, у якому відбувається зберігання та виконання синтезованого програмного коду JavaScript, який був створений у модулі arrMethods.py;

– main.py – головний файл у якому відбувається виклик необхідних функцій, обробка команд користувача та керування загальним ходом виконання програми.

## 3.2 Реалізація модулів програмного додатку

В модулі main.py (додаток А) основна логіка реалізована у функції main(). Вона починається з виклику функції ask(), яка запитує користувача про команду та записує отриманий текст у базу даних “data/text.shp”. Далі виконується цикл while, який перевіряє статус виконання програми та обробляє команди користувача. У тілі циклу, текст з файлу “data/text.shp” зчитується у змінну text, і на ньому застосовуються різні умови та порівняння тексту з використанням модуля “fuzzywuzzy”. Залежно від отриманих команд, викликаються відповідні функції для виконання операцій з масивами. Результати виконання цих функцій записуються у змінну js_code. Далі перевіряється статус продовження виконання програми за допомогою функції reask(). Якщо користувач підтверджує продовження, цикл while продовжується, і програма знову запитує команду користувача за допомогою функції ask(). Якщо користувач вказує припинити,

програмний додаток виходить з циклу і завершує своє виконання. Останній рядок: “if __name__ == "__main__"” забезпечує виклик функції main(), якщо модуль main.py запускається безпосередньо.

Модуль ask.py (додаток Б) відповідає за запит команди у користувача. Для цього використовується бібліотека “speech_recognition”, яка дозволяє отримувати розпізнаний текст з аудіо. У функції ask(), спочатку створюється об'єкт “Recognizer” з бібліотеки “speech_recognition”. Далі відбувається налаштування мікрофона для зчитування аудіо з джерела звуку. Після цього з'являється повідомлення: “Говоріть команду...” для користувача. За допомогою методу listen() об'єкту “Recognizer” аудіо записується з мікрофона. Далі використовується функція recognize_google() для розпізнавання тексту з отриманого аудіо. Результат розпізнавання зберігається у змінну “my_string”. Отриманий текст записується у файл “data/text.shp” за допомогою контекстного менеджера open() і методу write(). У випадку, якщо розпізнавання не вдалося або виникла помилка під час розпізнавання, виводиться відповідне повідомлення, а функція повертає значення “False”. У разі успішного розпізнавання та запису команди, функція повертає значення “True”.

Модуль reask.py (додаток В), відповідає за запит у користувача щодо продовження виконання програми. Використовується також бібліотека “speech_recognition” для розпізнавання голосу. Знову ж таки, повторюються всі процедури до запису у змінну “my_string”. Текст переводиться у нижній регістр за допомогою методу lower(). Потім виконується порівняння зі списком допустимих варіантів (options), що містить певні ключові слова, які вказують на позитивну відповідь.

Якщо текст збігається з будь-яким із варіантів у списку “options”, функція повертає значення “True”, що означає продовження виконання програми, якщо ні, то виводиться повідомлення про завершення роботи програми, а функція повертає значення “False”

Модуль normaliseString.py містить функцію normal(), яка виконує нормалізацію тексту. У функції normal() приймається один аргумент text, який є вхідним текстом, що потребує нормалізації. В модулі використовується словник

replacements, в якому вказані пари “старий символ” та “новий символ”. Наприклад, символ “.” замінюється на “,”. Далі за допомогою циклу “for” перебираються всі пари ключ-значення в словнику “replacements”. Кожен старий символ “old” замінюється на новий символ “new” в значенні “text” за допомогою методу replace(). Це виконується для кожної пари в словнику, що забезпечує заміну всіх необхідних символів.

```
def normal(text):
    replacements = {'.': ',', 'кома': ','}
    for old, new in replacements.items():
        text = text.replace(old, new)
    return text
```

Модуль arrMethods.py (додаток Г) містить в собі декілька функцій:

– Функція findStrInArr(str) отримує нормалізований рядок “str”, перебирає його символи і формує новий рядок “new_string”, який містить лише цифри та коми. Далі створюється змінна “js_code”, в яку записується результат синтезу програмного коду мовою JavaScript для створення масиву. Кожне число з рядка “new_string” додається до масиву “myArray” за допомогою методу push(). Код JavaScript записується до файлу index.js, і функція повертає масив “myArray” та список чисел words.

– Функція sortArray(js_code) отримує змінну з створеним по шаблону кодом і додає до нього синтаксис для стандартного сортування масиву “myArray”. Знову ж таки, код JavaScript записується до файлу index.js, і функція повертає оновлену змінну “js_code”.

– Функція binarySearch(js_code, new_text) отримує синтаксис JavaScript у змінні “js_code” і рядок “new_text”. Вона виділяє число, додає код JavaScript для бінарного пошуку цього числа в відсортованому масиві “arrSort”. Код JavaScript записується до файлу index.js, і функція повертає оновлену змінну “js_code.

– Функція removeCode(js_code) очищає змінну “js_code”, присвоюючи їй порожню стрічку. Файл index.js також очищується. Функція повертає очищену змінну.

В кожній функції модуля arrMethods.py виконується запис коду JavaScript до файлу index.js за допомогою функції write() об'єкта файлу. Після кожного запису в файл виводиться відповідне повідомлення про успішне виконання дії.

## 3.3 Тестування та порівняння програмного додатку з програмами-аналогами

Тестування додатку розпочинається з відкриття 2 середовищ розробки, PyCharm та Visual Studio Code. В останньому перечисленому середовищі відкривається пустий файл index.js та за допомогою терміналу, запускається утиліта nodemon як показано на рисунку 3.5.


Рисунок 3.5 – Запуск утиліти nodemon

Після попередніх дій, було відкрито структуру програмного додатку в PyCharm, та за допомогою інтерфейсу графічної кнопки старт, відбудеться запуск програмного додатку (рисунок 3.6).

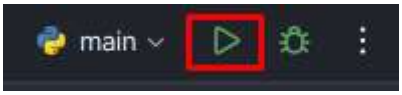


Рисунок 3.6 – Графічна кнопка старт у IDE PyCharm

Відбувається запуск програмного додатку, та виконується модуль ask.py, який дає користувачу текстову підказку, щоб прослухати голосову команду. До прикладу було проголошено наступне: “створи масив з числами 5 кома 7 кома 3 кома 15 кома 4 кома 2”. Після цього вивелося повідомлення в консоль про успішну компіляцію аудіозапису в текстовий формат (рисунок 3.7).

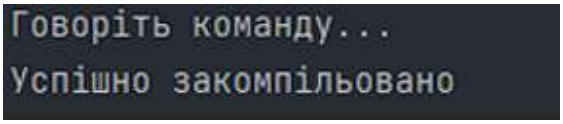


Рисунок 3.7 – Результат виконання модуля ask.py

В базу даних text.shp надійшов перетворений голосовий запиту у вигляді тексту.


Рисунок 3.8 – Приклад текстової команди в базі даних text.shp

Як видно на рисунку 3.8, значення дещо відрізняються від сказаного, але за допомогою нечіткого співставлення слів бібліотекою “fuzzywuzzy” співпадіння попереньо записаної інструкції “створи масив” та щойно розпізнаної команди користувача “створити масив” досягло відмітки вищої ніж 80%, відповідно умова для запуску функції, яка створює масив, успішно виконалася (рисунок 3.9).

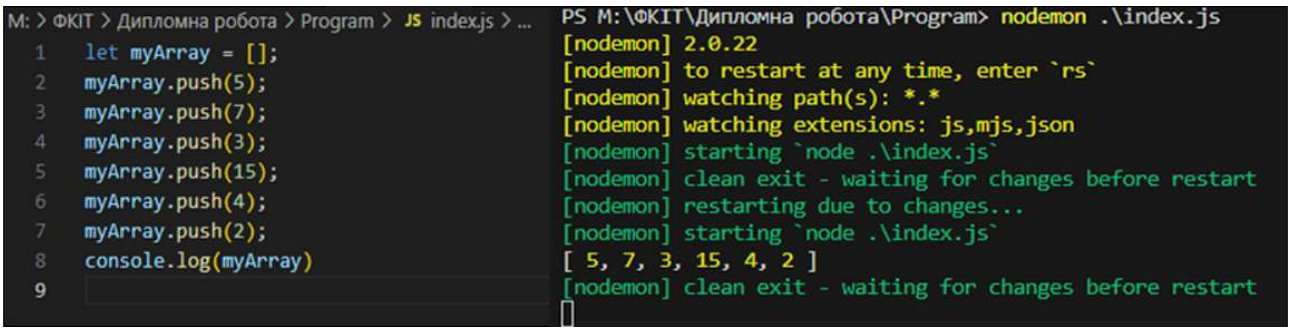


Рисунок 3.9 – Результат обробки команди згідно інструкції “створи масив”, за шаблоном мови JavaScript у файлі index.js

Після виконання функції для створення масиву на JavaScript, виконується блок резервного голосового запиту reask.py, який виводить відповідне повідомлення, та в залежності від відповіді, завершує або продовжує виконання програмного додатку (рисунок 3.10).

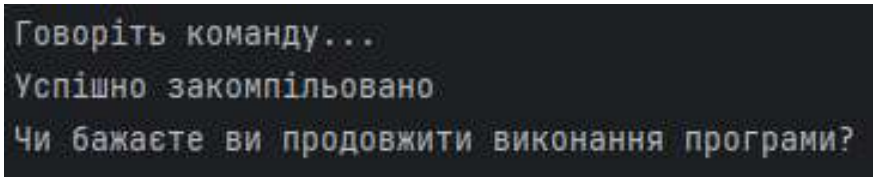


Рисунок 3.10 – Результат виконання модуля reask.py

Якщо користувач промовив голосову команду “Ні”, то відбувається вихід з програмного додатку (рисунок 3.11).

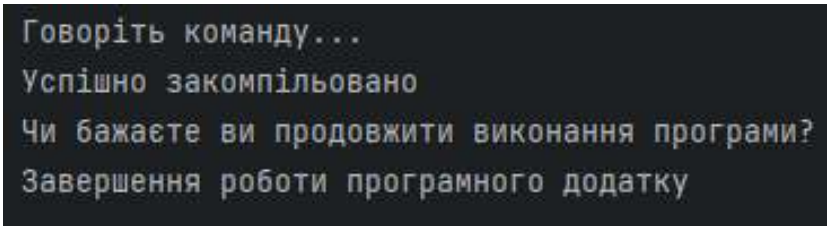


Рисунок 3.11 – Вихід з програмного додатку, згідно з голосовою командою користувача.

Якщо ж користувач промовив голосову команду “Так”, програмний цикл інструкцій розпочинається заново. До прикладу, користувач промовляє голосову команду: “відсоруй масив стандартно” (рисунок 3.12).

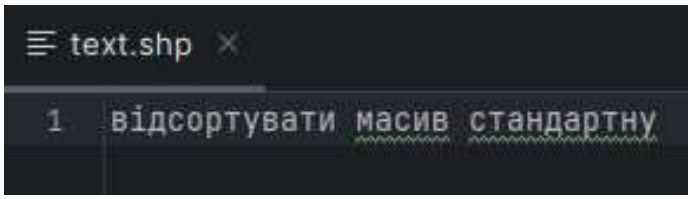


Рисунок 3.12 – Результат трансформації голосової команди “відсортуй масив стандартно” в текстовий вигляд у файлі text.shp

У модулі main.py виконується перевірка на наявність відповідної функції, що задовільнить команду користувача. Якщо ж відповідної функції немає у наявності, виводиться відповідне сповіщення в консолі, та виконується модуль reask.py, як показано на рисунку 3.13.

```
Говоріть команду...
Успішно закомпільовано
Відсутня функція такого типу...
Чи бажаєте ви продовжити виконання програми?
```

Рисунок 3.13 – Приклад роботи програми, при відсутності реалізації команди користувача

Оскільки у цьому модулі є відповідна інструкція, виконується функція, що сортує попередньо введений масив по зростанню (рисунок 3.14).

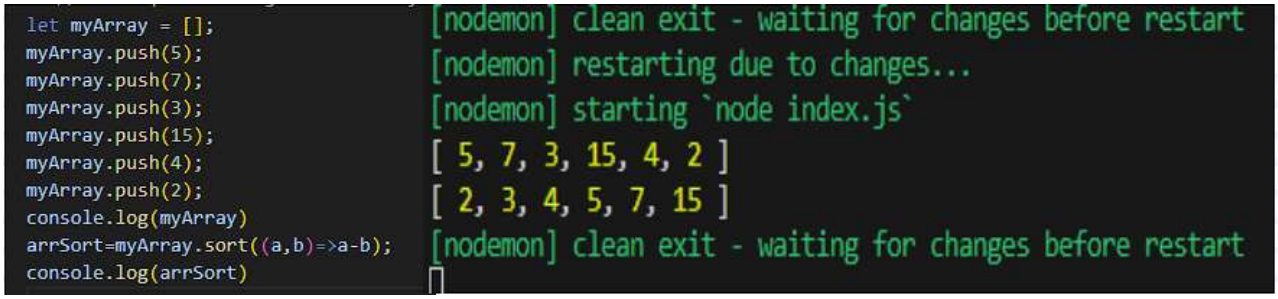


Рисунок 3.14 – Результат обробки команди користувача “відсортуй масив стандартно”, за шаблоном мови JavaScript у файлі index.js

Програма знову ж таки по попереднім крокам, виконує команду. До прикладу: “Знайди індекс числа 5, за допомогою метода бінарного пошуку” (рисунок 3.15).


Рисунок 3.15 – Результат обробки команди користувача “знайди індекс числа 5, за допомогою метода…”, за шаблоном мови JavaScript у файлі index.js

Результат є абсолютно правильним, оскільки нумерація індексів чисел в масиві розпочинається з 0.

Повторюються попередні дії та програмний додаток отримує вхідні дані у вигляді “очисти все”. Оскільки дана вказівка існує у головному модулі main.py, виконується функція очищення вихідного файлу index.js (рисунок 3.16).


Рисунок 3.16 – Результат виконання функції очищення на прикладі вихідного файлу index.js

В результаті тестування додатку, створена певна інструкція користувачу, для розуміння можливості застосунку та правильної вимови тієї, чи іншої команди.

Таблиця 3.1 – Вхідні і вихідні дані створеного програмного застосунку

| Вхідні дані | Вихідні дані |
|---|---|
| Створи масив з числами 23 кома 4 кома 9 кома 45. | [ 23, 4, 9, 45 ]. |
| Відсортуй масив стандартно. | [ 4, 9, 23, 45 ]. |
| Знайди індекс числа (вказуєте число, яке потрібно знайти, наприклад 9) за допомогою бінарного пошуку. | Елемент 9 знаходиться під індексом 1. |

Наразі це обмежені функції, але в майбутньому можна додавати безліч функцій, які допоможуть керувати програмою на мові JavaScript.

Щодо порівнянь з програмами аналогами, під час досліждення теми синтезу програмного коду мовою JavaScrpt на основі розпізнавання природньої мови, не було знайдено жодної схожої програми. Але існує декілька ресурсів, які окремо виконують дані функції, наприклад, перетворення аудіо в текст, чи перетворення синтаксису Python в JavaScript.

На сьогоднішній день, українська мова не має такої широкої підтримки в порівнянні з деякими іншими мовами, такими як англійська, і специфічні деталі можуть залежати від конкретних технологій та інструментів. Проте, існує наприклад голосовий асистент від компанії Apple, під назвою Siri.

Під час тестування техніки Apple, а саме модель “Iphone 11”, можна зазначити, що асистент Siri, краще розпізнає вимову, швидше працює, але є декілька недоліків. Наприклад, Apple не надає публічний API для розпізнавання голосу через Siri. API Siri є пропрієтарним і доступним лише для використання в середовищі Apple-продуктів, таких як iOS, iPadOS, macOS та watchOS.

Це означає, що розпізнавання голосу, що використовується в Siri, обмежено до функціональності, яка доступна в рамках окремих пристроїв та платформ Apple. Іншими словами, немає змоги прямо взаємодіяти з Siri Speech Recognition API поза екосистемою Apple.

Тобто, неможливо створити подібну програму, яка синтезує мовлення в JavaScript код, за допомогою Apple Siri Speech Recognition

Щодо перетворення мови Python в JavaScript, на першому місці в пошуковому запиті Google, знайдено сайт конвертор “Python to JavaScript” [51].

Розпочато тестування даного веб-застосунку.

У полі для введення під назвою “Python code” введено мовою Python, цикл, який має виводити в консоль, слово hello 5 разів

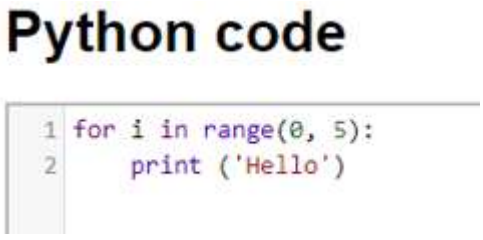


Рисунок 3.17 – Процес тестування додатка-конвертора мовою Python

В результаті, отримано JavaScript код,але є певні помилки в такому простому циклі.

Наприклад, “var” використовувався для створення змінних до введення стандарту ES6. На даний момент рекомендується використовувати let або const для оголошення змінних замість var. Введення let та const в ES6 принесло покращення у керуванні областями видимості та управлінні змінними.

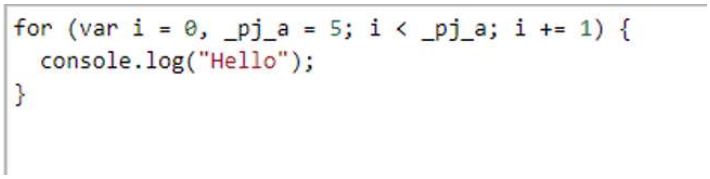


Рисунок 3.18 – Результат вихідного коду, на мові програмування JavaScript, у тестованому конверторі.

Або ж назва змінних є нечитабельною та незрозумілою користувачу, а в розробленому програмному додатку, все логічно зрозуміло, і використовується актуальний стандарт ES6.

При виконанні складніших запитів, наприклад таких як метод бінарного пошуку та інші, конвертер видає помилку, або ж некоректний код.

# 4 ТЕХНІКО-ЕКОНОМІЧНИЙ РОЗДІЛ

В цьому розділі кваліфікаційної роботи (КР) проводиться економічне обґрунтування доцільності розробки програмного засобу за алгоритмом синтезу програмного коду, мовою JavaScript на основі розпізнавання природньої мови. Застосування синтезу програмного коду може мати декілька економічних переваг. Наприклад, використання автоматичної генерації коду на основі природної мови може суттєво знизити витрати на розробку програмного забезпечення. Замість того, щоб розробники витрачали час і зусилля на написання кожного рядка коду вручну, вони можуть просто ввести голосову команду природною мовою, а додаток автоматично згенерує відповідний код.

Це дозволяє зберегти ресурси, знизити затрати на розробку та прискорити процес впровадження нових функцій або проектів. Зокрема здійснюється обрахунок трудомісткості, витрати на розробку вартості прогнозованої ціни та показників економічної ефективності створеного програмного додатку. Також у заключній частині обґрунтовуються висновки, щодо доцільності розробки програми.

## 4.1 Визначення трудомісткості розробки програмного забезпечення

Для визначення трудомісткості розробки програмного додатку, потрібно визначити етапи його розробки:

- дослідження алгоритму рішення задачі;
- розробка блок-схеми алгоритму;
- програмування по готовій блок-схемі;
- налагодження програми на персональному комп’ютері(ПК);
- підготовка документації;

Трудомісткість розробки ПЗ розраховують за формулою:

$$t = t_o + t_u + t_а + t_n + t_{налаг} + t_д \quad (4.1)$$

де $t_o$ – витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_u$ – витрати праці на дослідження алгоритму рішення задачі;

$t_a$ – витрати праці на розробку блок-схеми алгоритму;

$t_n$ – витрати праці на програмування по готовій блок-схемі;

$t_{налаг}$ – витрати праці на налагодження програми на ПК;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється. Умовне число операторів (підпрограм) розраховують за формулою:

$$Q = q * C * (1 + p), \quad (4.2)$$

де $q$ – передбачуване число операторів;

$C$ – коефіцієнт складності програми;

$p$ – коефіцієнт кореляції програми в ході її розробки.

$$Q = 200 * 1{,}7 * (1 + 0{,}05) = 357,$$

де 200 – кількість передбачуваних функцій;

1,7 – коефіцієнт складності завдання, який характеризує відносну складність програми завдання по відношенню до так званої типової задачі, складність якої прийнята рівною одиниці (величина “С” лежить в межах від 1,25 до 2);

0,05 – коефіцієнт корекції програми, тому що на практиці при розробці програми в середньому вноситься 3–5 корекцій, кожна з яких веде до переробки

від 5% до 10% готової програми, тобто величина p знаходиться в межах 0,005-0,1.

Знайдемо всі витрати праці $t_u$, яке визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q*B}{(75..85)*k}, \quad (4.3)$$

де $B$ – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

$k$ – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

$$t_u = \frac{357*1,2}{75*0,8} = 7,14 \text{ людино-годин},$$

де 357 – знайдене за формулою 4.2, умовне число операторів;

1,2 – якість постановки задачі, виданої для розробки програми, з зв'язку з тим, що завдання, як правило, вимагають уточнення і доопрацювання (практика показує, що в більшості випадків цей коефіцієнт в залежності від складності завдання приймається від 1,2 до 1,5);

0,8 – коефіцієнт кваліфікації розробника, який враховує k-ступінь підготовленості виконавця до дорученої йому роботі котрий визначається залежно від стажу роботи і становить:

- для працюючих до двох років – 0,8;
- від двох до трьох років – 1,0;
- від трьох до п'яти років – (1,1-1,2);
- від п'яти до семи років – (1,3-1,4);
- понад сім років – (1,5-1,6).

Витрати праці на блок-схеми алгоритму розраховують за формулою:

$$t_a = \frac{Q}{(20..25)*k} \quad (4.4)$$

Підставивши в формулу 4.4 знайдені показники, розрахуємо дані витрати:

$$t_a = \frac{357}{20*0.8} = 22{,}31 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі розраховують за формулою:

$$t_n = \frac{Q}{(20..25)*k}. \quad (4.5)$$

Підставивши в формулу 4.5 знайдені показники, отримаємо наступні витрати:

$$t_n = \frac{357}{25*0.8} = 17{,}85 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ПК розраховують за формулою:

$$t_{\text{налаг}} = \frac{Q}{(4..5)*k}. \quad (4.6)$$

Підставивши в формулу 4.6 знайдені показники, розрахуємо дані витрати:

$$t_{\text{налаг}} = \frac{357}{5*0{,}8} = 89{,}25 \text{ людино-годин.}$$

Витрати праці на підготовку документації розраховують за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \tag{4.7}$$

де $t_{\text{др}}$ – трудомісткість підготовки матеріалів і рукопису, що розраховують за формулою:

$$t_{\text{др}} = \frac{Q}{(15..20)*k}; \tag{4.8}$$

$$t_{\text{др}} = \frac{357}{20*0.8} = 22{,}31 \text{ людино-годин.}$$

$t_{\text{до}}$ – трудомісткість редагування, друкування й оформлення документації, яка обчислюється за формулою:

$$t_{\text{до}} = 0{,}75 * t_{\text{др}}; \tag{4.9}$$

$$t_{\text{до}} = 0{,}75 * 22{,}31 = 16{,}73, \text{ людино-годин.}$$

Підставивши знайдені значення за формулами (4.8, 4.9), отримаємо витрати праці на підготовку документації:

$$t_{\text{д}} = 22{,}31 + 16{,}73 = 39{,}04, \text{ людино-годин.}$$

Оскільки, знайдено всі невідомі значення, можна розрахувати трудомісткість роботи за формулою 4.1

$$t = 50 + 7{,}14 + 22{,}31 + 17{,}85 + 89{,}25 + 39{,}04 = 225{,}59, \text{ людино-годин.}$$

Отже, трудомісткість створення програмного додатку на основі алгоритму синтезу програмного коду мовою JavaScript рівна 225,59 людино-годин, що включає у себе основні етапи цієї розробки.

## 4.2 Розрахунок витрат на створення програмного забезпечення

Розрахуємо середньогодинну оплату програміста. Для цього необхідно спочатку визначити його річний фонд грошового забезпечення. Це можна зробити, знаючи місячне грошове забезпечення програміста. Воно складає приблизно 20000,00 гривень. Таким чином, річний фонд грошового забезпечення 240000 гривень.

Кількість робочих годин у році розрахуємо за формулою:

$$N_p = (N - N_n - N_b) * 8\,, \tag{4.10}$$

де $N$ – загальна кількість днів у році,

$N_n$ – кількість святкових днів у році,

$N_b$ – кількість вихідних днів у році.

Приймається, що кількість святкових днів у році – 14, а вихідних – 104, тому оскільки відомо всі значення, можна розрахувати кількість робочих годин у році за формулою 4.10:

$$N_{\text{р}} = (365 - 14 - 104) * 8 = 1976 \text{ годин}$$

Середньогодинна оплата праці програміста визначається за формулою:

$$C_n = \frac{\Phi_p}{N_p}, \tag{4.11}$$

де $Ф_р$ – річний фонд грошового забезпечення.

Підставивши знайдені значення в формулу 4.11, отримаємо середньогодинну оплату праці програміста

$$C_n = \frac{240000}{1976} = 121{,}45 \text{ грн.}$$

Витрати на оплату праці розробника програми обчислюють за формулою:

$$B_{\text{оп}} = C_n * T_з \tag{4.12}$$

де $T_з$, - загальна кількість годин роботи програміста над проектом.

Підставивши відомі елементи, витрати на оплату праці становлять:

$$B_{\text{оп}} = 121{,}45 * 225{,}59 = 27397 \text{ грн.}$$

Витрати у соціальні фонди складають 20,5%

$$B_ф = B_{\text{оп}} * k \tag{4.13}$$

де $k$ – коефіцієнт відрахувань у державні соціальні фонди

$$B_ф = 27397 * 20{,}5 = 5616 \text{ грн.}$$

З матеріальних витрат, це лише вартість ПК. Модель :Lenovo Z50-70, тому вартість ПК розраховуємо за формулою:

$$Ц_{ПК} = Ц_Р * (1 = К_{УН}), \tag{4.14}$$

де $Ц_р$ – ринкова вартість ПК (15000 гривень),

$К_{УН}$ – коефіцієнт, що враховує витрати на установку й налагодження ПК(приймається рівним 12%).

Підставивши відомі значення, отримаємо вартість ПК:

$$Ц_{ПК} = 15000 * (1 + 0{,}12) = 16800 \text{ грн.}$$

Річні поточні витрати на експлуатацію програмного забезпечення визначаються за формулою:

$$B_e = B_{E_p} + B_{А_р} + B_{РЕМ_р} + B_{ДК_p} + B_{І_p}, \quad (4.15)$$

де $B_{A_p}$ – річні відрахування на амортизацію,

$B_{E_p}$ – річні витрати на електроенергію для ПК,

$B_{РЕМ_p}$ – річні витрати на ремонт ПК,

$B_{ДК_p}$ – річні витрати на додаткові комплектуючі ПК,

$B_{І_p}$ – інші витрати.

Суму річних амортизаційних відрахувань визначаємо за такою формулою:

$$В_{А_р} = Ц_{ПК} * Н_А, \quad (4.16)$$

де $Н_А$ – норма амортизаційних відрахувань (дорівнює 15% у квартал).

$$B_{A_p} = 16800 * (4 * 0{,}15) = 10080 \text{ грн.}$$

Витрати на електроенергію, що споживає ПК, визначаємо за формулою

$$B_{E_p} = Р_{ПК} * Ф_{ПК} * Ц_Е * К_{ІВ}, \quad (4.17)$$

де $Р_{ПК}$ – паспотна потужність ПК,

$Ф_{ПК}$ – річний фонд корисного часу роботи ПК,

$Ц_{Е}$– вартість 1 кВт/год електроенергії,

$К_{ІВ}$ – коефіцієнт інтенсивного використання ПК (0,7 - 1).

Розрахуємо річний фонд часу роботи ПК. Визначивши дійсний річний фонд часу ПК у годинах, можна оцінити собівартість годин машинного часу.

Дійсний річний фонд часу ПК дорівнює:

$$Ф_{ПК} = Ф_{д} = N_P - (Ф_m + Ф_{річ}) \quad (4.18)$$

де $Ф_m$ - місячний фонд часу на профілактику і ремонт ПК (час профілактики щомісячно – 5 годин),

$Ф_{річ}$ - річний фонд часу на профілактику і ремонт ПК (час профілактики щорічно – 6 діб)

Знайдемо $Ф_m$ та $Ф_{річ}$ підставивши у них відомі значення:

$$Ф_m = 5 * \text{кількість місяців у році},$$

$$Ф_{м} = 5 * 12 = 60 \text{ годин.}$$

$$Ф_{річ} = 6 * \text{норма годин за робочу зміну},$$

$$Ф_{річ} = 6 * 8 = 48 \text{ годин.}$$

Оскільки всі невідомі показники були обчисленні, відповідно:

$$Ф_{ПК} = Ф_{д} = 1976 - (60 + 48) = 1868 \text{ годин.}$$

Отже, згідно формулі 4.17, витрати на електроенергію складають:

$$B_{E_p} = 0{,}15 * 1868 * 1{,}44 * 0.8 = 322{,}8 \text{ грн.}$$

Витрати на поточний і профілактичний ремонт (приймаються рівними 6% від вартості ПК) становлять:

$$B_{\text{РЕМ}_\text{р}} = \text{Ц}_{\text{ПК}} * 0{,}06; \quad (4.19)$$

$$B_{РЕМ_р} = 16800 * 0{,}06 = 1\ 008 \text{ грн.}$$

Витрати на додаткові комплектуючі – витрати необхідні для забезпечення експлуатації ПК (приймаються рівними 2% від вартості ПК):

$$B_{\text{ДК}_\text{р}} = \text{Ц}_{\text{ПК}} * 0{,}02; \quad (4.20)$$

$$B_{\text{ДК}_р} = 16800 * 0{,}02 = 336 \text{ грн.}$$

Інші витрати, тобто непрямі витрати пов'язані з експлуатацією ПК (приймаються рівними 5-10% від вартості ПК):

$$B_{\text{I}_\text{р}} = \text{Ц}_{\text{ПК}} * 0{,}07, \quad (4.21)$$

$$B_{\text{I}_р} = 16800 * 0.07 = 1176 \text{ грн}$$

Отже, знайшовши усі показники можна вирахувати річні поточні витрати на експлуатацію програмного забезпечення визначаються за формулою 4.15:

$$B_е = 322{,}8 + 10080 + 1008 + 336 + 1176 = 12922{,}8 \text{ грн.}$$

У ході розробки програми, персональний комп’ютер використовується на таких етапах програмування:

– написання програми за готовою схемою алгоритму;

– налагодження програми на ПК;

– підготовки документації по задачі.

Таким чином, витрати машинного часу склали( $t_{маш}$):

$$t_{маш} = t_n + t_{налаг}^k + t_д, \quad (4.22)$$

де $t_{налаг}^k$ - витрати праці на налагодження програми за умови комплексного налагодження завдання.

$$t_{налаг}^k = 1{,}5 * t_{налаг}; \quad (4.23)$$

$$t_{налаг}^k = 1{,}5 * 89{,}25 = 133{,}87 \text{ годин.}$$

Згідно з формулою 4.22, розрахуємо витрати машинного часу:

$$t_{маш} = 17{,}85 + 133{,}87 + 39{,}04 = 190{,}76 \text{ годин.}$$

Витрати на оплату машинного часу розраховуємо за формулою:

$$В_{маш} = t_{маш} * С_{ПК}; \quad (4.24)$$

де $C_{ПК}$ - собівартість машинного часу обчислювальної техніки (розраховує бухгалтерія підприємства).

Собівартість однієї години роботи ПК дорівнює:

$$C_{ПК} = \frac{B_e}{Ф_{ПК}} \quad (4.25)$$

$$C_{ПК} = \frac{12922{,}8}{1868} = 6{,}92 \text{ грн}$$

Відповідно до формули 4.24, витрати на оплату машинного часу:

$$В_{маш} = 190,76 * 6,92 = 1320 \text{ грн.}$$

Загальні витрати на розробку програмного комплексу обчислюються за формулою:

$$B_{заг} = B_{оп} + B_{ф} + B_{маш} + Ц_{ПК} + B_{е}; \quad (4.26)$$

$$B_{заг} = 27397 + 5616 + 1320 + 16800 + 12922,8 = 64055,8 \text{ грн.}$$

На основі отриманих даних складаємо кошторис витрат на розробку програмного забезпечення (таблиця 4.1).

Таблиця 4.1 – Кошторис витрат на розробку ПЗ

| № | Найменування витрат | Сума витрат(грн) |
|---|---|---|
| 1 | Витрати на оплату праці розробника | 27397 |
| 2 | Витрати у соціальні фонди | 5616 |
| 3 | Матеріальні витрати | 16800 |
| 4 | Річні витрати на експлуатацію ПЗ | 12922,8 |
| 5 | Витрати на оплату машинного часу | 1320 |
| РАЗОМ витрати на розробку | | 64055,8 |

Отже, загальні витрати на розробку програмного додатку оцінюються в 64055,8 гривень, включаючи різні чинники.

## 4.3 Визначення прогнозованої ціни

Величина можливої ціни додатку повинна визначатися з урахуванням ефективності, якості і термінів її виконання на рівні інтересів замовника

Договірна ціна розраховується формулою:

$$B_i = B_{\text{заг}}(1+\frac{p}{100}) \quad (4.27)$$

де $p$ – середній рівень рентабельності на поточний період(приймається в розмірі 20-30%.)

Тоді, ціна розробленого додатку становить:

$$B_i = 64055(1+\frac{20}{100}) = 76866 \text{ грн.}$$

4.4 Розрахунок показників економічної ефективності розробки ПЗ.

Показник інвестиційних вкладень з урахуванням інфляційних процесів обчислюємо за формулою:

$$K_i = \varphi_i + R_i, \quad (4.28)$$

де $\varphi_i$ - коефіцієнт інфляції на поточний період (102,5%),

$R_i$ - інвестиційні платежі в і-му періоді (капітальні вкладення).

Візьмемо за інвестицію ($R_i$), загальну витрату на розробку програми, тоді, показник інвестиційних вкладень з урахуванням інфляційних процесів складає:

$$\text{К}_\text{і} = 1{,}025 * 76866 = 78787 \text{ грн}$$

Дохід від розробки ПЗ у і-му періоді розраховуємо за формулою:

$$D_i = J_i(B_i - C_i). \quad (4.29)$$

де $C_i$ – собівартість програмного продукту (сума витрат на розробку ПЗ);

$J_i$- кількість ПЗ.

При умові, якщо кількість екземплярів додатку буде продана в кількості 3 штук, дохід від розробки ПЗ у і-му періоді складатиме:

$$D_{\text{і}} = 3(78787 - 64055) = 44196 \text{ грн.}$$

Економічна ефективність полягає у відношенні результату від розробленого програмного продукту до затрачених ресурсів:

$$E = \frac{D_i}{B_{\text{заг}}} \quad (4.30)$$

$$E = \frac{44196}{78787} = 0{,}56$$

Тоді термін окупності можна розрахувати за такою формулою:

$$T = \frac{1}{E} \quad (4.31)$$

$$T = \frac{1}{0{,}52} = 1{,}78 \text{ роки}$$

Враховуючи основні економічні показники, що стосуються розробки програмного продукту, можна зробити висновок, щодо доцільності запропонованої розробки. Отримано суттєвий економічний ефект від розробки програмного продукту, а термін окупності капітальних вкладень трохи менший за 2 роки, тому така розробка є економічно вигідною та конкурентоздатною на ринку подібних ІТ продуктів

# ВИСНОВКИ

1.В результаті кваліфікаційної розробки, проведено аналіз природніх та структурованих мов, за допомогою пошуку інформації в наукових статтях та книгах, виділено ключові аспекти та характеристики для створення формальних мов та використання їх у сучасному світі. Таким чином, даний аналіз дав чітке поняття різниці між мовами, та особливості їх використання.

2.Досліджено різновиди мов програмування та середовищ розробки (IDE) шляхом чергового порівняння кожної з них. Тому, ця інформація дає ясність у питанні вибору створення архітектури, для реалізації програмного додатку синтезу програмного коду мовою JavaScript на основі розпізнавання природньої мови.

3.Проаналізовано програмні засоби для розпізнавання природньої мови шляхом підставлення в ці засоби попередньо сформованих вимог для виконання кваліфікаційної роботи. В результаті чого, було знайдено оптимальний засіб, для розпізнавання природньої мови, а саме інструмент “SpeechRecognition”.

4.Розглянуто існуючі методи та алгоритми для обробки і порівняння природніх мов за допомогою пошуку інформацій в відповідних статтях, журналах. Таким чином, шляхом порівняння даних методів, було обрано бібліотеку для нечіткого співставлення слів “Fuzzy Wuzzy”.

5.Реалізовано створений алгоритм у програмному додатку в рамках обраної архітектури та середовищ розробки на основі власного досвіду та допомогти наукового керівника. Це дало можливість отримати готову програму, для точного аналізу переваг та недоліків створеного алгоритму.

6.Протестовано програмний додаток, виправлено декілька помилок, в ході тестування додатку та проведено порівняння з програмами аналогами шляхом перевірки кожної з них. В результаті чого, було знайдено переваги створеної програми над тестованими зразками. Досліджено конкурентні особливості, які в майбутньому можна реалізувати у створеному застосунку.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Шпак В.О., Пошук слів-відповідників за допомогою бібліотеки “Fuzzy Wuzzy”, на мові Python. Збірник тез доповідей VII науково-практичної конференції молодих вчених і студентів “Інтелектуальні комп’ютерні системи та мережі” 23 травня 2023 р. Тернопіль. Україна. С 32.

2. ДСТУ 3008:2015 Національний стандарт України. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Введ. 01.07.2017. К.: ДП “УкрНДНЦ”, 2016. 25 с.

3. ДСТУ 8302:2015 Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Введ. 01.07.2016. К.: ДП “УкрНДНЦ”, 2017. 16 с.

4. Методичні вказівки до випускних кваліфікаційних робіт освітнього рівня “Бакалавр” спеціальності “Комп’ютерна інженерія”/ О.М. Березький, Г.М. Мельник, Л.О.Дубчак, Ю.М. Батько / Під ред. О.М. Березького. Тернопіль: ЗУНУ, 2021. – 52 с.

5. Методичні вказівки до виконання практичних робіт з дисципліни «Техніко-економічне обґрунтування розробки комп’ютерних систем”/ Н.Я. Савка, І.Р. Паздрій / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 40 с.

6. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.

7. Друзь М. С. англійська мова як мова міжнародної комунікації. міжкультурна комунікація в контексті глобалізаційного діалогу: стратегії розвитку. ч 2. 2022. URL: https://doi.org/10.36059/978-966-397-280-0-14.

8. Srivastava S., Lamba S. Hindi-CNL Coder - A Desktop Application for Learning Programming using Native Controlled Natural Language. 2020 IEEE 20th

International Conference on Advanced Learning Technologies (ICALT), Tartu, Estonia, 6–9 July 2020. 2020A.

9. Смолій Н., Лісовиченко О., Смолій В. Засоби моделювання: формальна мова описання поведінки клітинних автоматів. Адаптивні системи автоматичного управління. 2022. Т. 2, № 41. С. 16–21.

10. Anderson J., Hekmatnejad M., Fainekos G. PyFoReL: A Domain-Specific Language for Formal Requirements in Temporal Logic. 2022 IEEE 30th International Requirements Engineering Conference (RE), Melbourne, Australia, 15–19 August 2022. 2022. URL: https://doi.org/10.1109/re54965.2022.00037

11. Devgade P. The Natural Language Query using NLP by Generation of SQL Query. International Journal for Research in Applied Science and Engineering Technology. 2021. Т. 9, № 5. С. 381–383.

12. Формальні мови, граматики та автомати: Навчальний посібник/ Гавриленко С.Ю. – Харків: НТУ « ХПІ », 2021. – 133 с.

13. Пекарський Б.Г. Основи програмування: Навчальний посібник.- Кондор,2018.-364 с.

14. А. В. Яковенко. Основи програмування. Python. Частина 1 підручник для студ. спеціальності 122 "Комп'ютерні науки", спеціалізації "Інформаційні технології в біології та медицині"; КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.

15. Юрченко А.О., Самойленко Л.О. Мови програмування, які вивчаються у ЗЗСО. Діджиталізація освітнього простору України: міжнародна науково-практична конференція (Суми, 13-18 вересня 2019 р.). С.26-35.

16. Притика, О. В. Про особливості мови програмування С++ [Текст] / О. В. Притика, А. О. Юрченко // Інформаційні технології в професійній діяльності : матеріали XIV Всеукраїнської науково-практичної конференції. – Рівне : РВВ РДГУ, 2021. – С. 155–156.

17. Khanfor A., Yang Y. An Overview of Practical Impacts of Functional Programming. 2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW), Nanjing, 4–8 December 2017. 2017. URL: https://doi.org/10.1109/apsecw.2017.27

18. Програмування [Текст] : навч. посіб. / О. М. Березький, Ю. М. Батько, М. М. Касянчук [та ін.]. – Тернопіль : ТНЕУ, 2018. – 256 с.

19. Zhirkov I. Good Code Practices. Low-Level Programming. Berkeley, CA, 2017. P. 241–262. URL: https://doi.org/10.1007/978-1-4842-2403-8_13

20. Конспект лекцій з курсу “Програмування на C#” для студентів спеціальності економічна кібернетика та компютерні науки: URL: https://kleban.page/courses/csharp-basics

21. Sharan K. New Input/Output 2. Java Language Features. Berkeley, CA, 2018. C. 487–547

22. Денисенко О. сучасні методи паралельного програмування. мистецтво наукової думки. 2019. № 8. с. 64–68

23. Nagpal A., Gabrani G. Python for Data Analytics, Scientific and Technical Applications. 2019 Amity International Conference on Artificial Intelligence (AICAI), Dubai, United Arab Emirates, 4–6 February 2019. 2019. URL: https://doi.org/10.1109/aicai.2019.8701341

24. Hoch T., Küveler G. Prozessüberwachung. C/C++ anwenden. Wiesbaden, 2019. C. 3–7

25. Flanagan D. JavaScript: The Definitive Guide. 7-ме вид. O’Reilly Media, 2020. 706 с.

26. Aljafer H., Cantrell G. Learning and Teaching Undergraduate Introductory Programming Courses in Java – The Use of an IDE VS Command Line. 2020 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 16–18 December 2020. 2020.

27. PyCharm: the Python IDE for Professional Developers by JetBrains. JetBrains. URL: https://www.jetbrains.com/pycharm

28. Microsoft. Documentation for Visual Studio Code. Visual Studio Code - Code Editing. Redefined. URL: https://code.visualstudio.com/docs

29. Delip Rao, Brian McMahan. Natural Language Processing with PyTorch - Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019. – 210 с.

30. Falconer K. NLP Professional: Your Future in NLP. McNidder & Grace, 2022. -176 с

31. Design and Construction of a Knowledge Database for Learning Japanese Grammar Using Natural Language Processing and Machine Learning Techniques / J. Liu et al. 2022 4th International Conference on Natural Language Processing (ICNLP), Xi'an, China, 25–27 March 2022. 2022. URL: https://doi.org/10.1109/icnlp55136.2022.00068

32. Speech to Text Translation enabling Multilingualism / S. Bano et al. 2020 IEEE International Conference for Innovation in Technology (INOCON), BANGLURU, 6–8 November 2020. 2020. URL: https://doi.org/10.1109/inocon50539.2020.9298280

33. Jurafsky D., James H. Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Draft of December 30, 2020, 623 с.

34. SpeechRecognition. PyPI. URL: https://pypi.org/project/SpeechRecognition/

35. Induced Local Attention for Transformer Models in Speech Recognition / T. Watzel та ін. Speech and Computer. Cham, 2021. С. 795–806. URL: https://doi.org/10.1007/978-3-030-87802-3_71

36. Tatsuoka M., Kaneko M. Wire congestion aware high level synthesis flow with source code compiler. 2018 International Conference on IC Design & Technology (ICICDT), Otranto, 4–6 June 2018. 2018.

37. Chatbot using NLP / A. R et al. Advances in Intelligent Systems and Technologies. 2022. P. 105–110.

38. Ticketing Chatbot Service using Serverless NLP Technology / E. Handoyo et al. 2018 5th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Semarang, 27–28 September 2018. 2018. URL: https://doi.org/10.1109/icitacee.2018.8576921

39. Becker S., Ghahramani Z., Dietterich T. G. Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Conference. MIT Press, 2019. 1600 с.

40. Бородкіна, І. Л. Теорія алгоритмів [Текст] : посібник / І. Л. Бородкіна, Г. О. Бородкін. – К. : ЦУЛ, 2018. – 184 с

41. Baka B., Agarwal D. B. Hands-On Data Structures and Algorithms with Python: Write complex and powerful code using the latest features of Python 3.7, 2nd Edition. Packt Publishing, 2018. 398 p.

42. Lima G. S. Algorithms with Python. Independently Published, 2019.

43. Дебела І. М. БАЙЄСОВСЬКИЙ МЕТОД ОЦІНКИ АЛЬТЕРНАТИВНИХ РІШЕНЬ. Таврійський науковий вісник. Серія: Економіка. 2021. № 8. С. 76–81. URL: https://doi.org/10.32851/2708-0366/2021.8.11

44. Ivanishyn I. USING VIOLA-JONES AND K NEAREST NEIGHBORS METHODS FOR FACE DETECTION AND RECOGNITION. International scientific journal "Internauka". 2022. № 11(130). URL: https://doi.org/10.25313/2520-2057-2022-11-8281

45. Vukatana K. OCR and Levenshtein distance as a measure of image quality accuracy for identification documents. 2022 International Conference on Electrical, Computer and Energy Technologies (ICECET), Prague, Czech Republic, 20–22 July 2022. 2022. URL: https://doi.org/10.1109/icecet55527.2022.9872824

46. C. Wagner, S. Miller, J. Garibaldi, D. Anderson and T. Havens, "From Interval-Valued Data to General Type-2 Fuzzy Sets", Fuzzy Systems IEEE Transactions on, vol. 23, pp. 248-269, Apr. 2015.

47. Python M. JavaScript for Beginners: The Simplified for Absolute Beginner's Guide to Learn and Understand Computer Programming Coding with JavaScript Step by Step. Basics Concepts and Practice Examples Inside. Independently Published, 2020.

48. Swift J. JavaScript for GIS. Geographic Information Science & Technology Body of Knowledge. 2020. Vol. 2020, Q3. URL: https://doi.org/10.22224/gistbok/2020.3.5

49. Shah D. Node. Js. BPB Publications, 2018. 207 c

50. Shen V. R. L., Wei C.-S., Juang T. T.-Y. Javascript Malware Detection Using A High-Level Fuzzy Petri Net. 2018 International Conference on Machine Learning and Cybernetics (ICMLC), Chengdu, 15–18 July 2018. 2018.

51. Python to javascript converter. Toolbox for Developers. URL: https://extendsclass.com/python-to-javascript.html