Київський політехнічний інститут імені Ігоря Сікорського
Інститут атомної та теплової енергетики
Кафедра ЦТЕ

Алгоритмізація та програмування – 2: Процедурне програмування

КУРСОВА РОБОТА

« Парсинг даних із сайту KTM.com »

Варіант № 12

Дата «5» червня 2023

Виконав: студент 1 курсу
гр. ТР-25
Кудрявцев Єгор Олегович

Оцінка «_____»
Дата «___» _____ 2023

Перевірив: Мірошниченко
Іван Володимирович,_____
(П.І.Б., підпис)

Київ – 2023

ЗАВДАННЯ

на курсову роботу студента

ТР-25 Кудрявцев Єгор Олегович

1. Тема роботи: Парсинг сайту «https://www.ktm.com/en-int/», збереження моделей транспорту та їх силок у .txt файл.

2. Термін здачі студентом закінченого роботи: 09.05.2023
3. Вихідні дані до роботи: Середовище розробки – Replit.com, мова програмування C#

4. Зміст пояснювальної записки (перелік питань, які розробляються):

Збереження html коду сторінки сайту, а також моделей транспорту та їх силок у .txt файл.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових рисунків):

6. Консультанти по роботі, з вказівкою розділів роботи, які до них відносяться

7. Дата видачі завдання: 01.02.2023

№ з/п	Назва етапів виконання курсової роботи	Строк виконання етапів розробки	Примітка
1	Отримання завдання	1-3 тижні	
2	Вибір та дослідження методів, вибір відповідних структур даних, розробка алгоритмів	4-6-й тижні	
3	Програмна реалізація	6-10-й тижні	
4	Демонстрація першого варіанту	11-ий тиждень	
5	Тестування програми	11-ий тиждень	
6	Оформлення звіту	12-й тиждень	
7	Захист курсової	13-й тиждень	

Керівник _____

(підпис)

Завдання прийняв до виконання ЄКуд

(підпис)

Зміст

ВСТУП

Парсер - це програмний інструмент або модуль, який використовується для аналізу тексту або послідовності символів згідно з певними правилами, відомими як граматика. Основна мета парсера - розпізнати структуру вхідних даних та витягнути корисну інформацію з цих даних.

Вони часто застосовуються у випадках, коли необхідно аналізувати складні або структуровані дані, такі як програмний код, мови розмітки, документи або дані з бази даних. Вони використовуються для перетворення вхідних даних у більш зрозумілу або внутрішню форму для подальшого оброблення, виконання операцій чи отримання необхідної інформації.

Парсери можуть працювати за різними принципами. Наприклад, найпоширенішим типом парсерів є синтаксичні парсери, які використовують контекстно-вільну граматику для аналізу вхідних даних. Вони базуються на алгоритмах, таких як LL(k), LR(k) або LALR, і дозволяють виявляти синтаксичні помилки, розпізнавати правильну структуру та створювати синтаксичне дерево.

Застосування парсерів широкі і розповсюджені. Вони використовуються у компіляторах для перетворення мови програмування у машинний код, у веб-розробці для обробки інформації з веб-сторінок, у системах обробки мови для аналізу та інтерпретації тексту, а також у багатьох інших областях, де потрібно аналізувати, розпізнавати або обробляти структуровані дані.

Розробляючи сьогоднішній парсер, я хочу показати можливості програмістів діставати велику кількість інформації натискання лише однієї кнопки. Проте для такого результату потрібно буде створити робочий код, який повинен мати непогану швидкість виконання та діставати дані з обраного сайту, коли користувач цього захоче.

1. ЩО ТАКЕ ПАРСЕР

1.1 Використання

Парсери мають широкий спектр застосувань і використовуються в різних областях інформатики та програмування. Ось декілька прикладів, де парсери є необхідними і корисними:

1. **Компіляція мов програмування:** Парсери грають ключову роль у процесі компіляції. Вони аналізують синтаксичну структуру вихідного коду програми і перетворюють його на внутрішнє представлення, таке як синтаксичне дерево або проміжний код. Це дозволяє компілятору виконувати оптимізації, перетворювати код в ефективний машинний код або іншу цільову форму.
2. **Розбір мов розмітки:** Парсери використовуються для аналізу мов розмітки, таких як HTML, XML, JSON та інші. Вони дозволяють зчитувати, обробляти та витягувати дані зі структурованих документів. Наприклад, парсер HTML може виявляти елементи, атрибути та текстовий контент на веб-сторінці, що дозволяє веб-скраперам отримувати потрібну інформацію з сайтів.
3. **Робота з базами даних:** Парсери використовуються для обробки та аналізу запитів до баз даних. Вони дозволяють розпізнавати синтаксичну структуру SQL-запитів та виконувати операції, такі як вибірка, вставка, оновлення або видалення даних з бази. Парсери також використовуються для роботи з іншими форматами зберігання даних, такими як CSV або JSON.
4. **Мовні процесори та інтерпретатори:** Парсери використовуються у системах обробки мови для аналізу та розпізнавання природної мови. Вони можуть визначати граматику речень, розбивати текст на токени, визначати частини мови, розпізнавати іменовані сутності та виконувати інші завдання обробки мови.
5. **Веб-скрапінг та автоматизація:** Парсери використовуються для аналізу веб-сторінок та витягування інформації з них. Вони дозволяють отримувати дані з таблиць, списків, заголовків або інших елементів веб-сторінок. Це корисний інструмент для веб-скрапінгу, автоматизації веб-процесів та отримання даних для аналітики.

6. **Мовні інтерфейси та обробка команд:** Парсери використовуються для розпізнавання та обробки команд, введених користувачами у текстових інтерфейсах або командних рядках. Вони дозволяють розуміти команди, виконувати дії та обмінюватися інформацією з користувачем.

1.2 Як можна його створити

Загалом створення парсера може бути завданням, що потребує вивчення і розуміння граматики або синтаксису даних, які потрібно розпізнати. Ось загальний опис кроків, які можуть бути виконані для створення парсера:

1. **Визначення граматики:** Потрібно почати з визначення граматики або правил, які описують синтаксис або структуру даних, які потрібно розпізнати. Граматика може бути виражена у формі контекстно-вільних граматик, регулярних виразів, діаграм чи іншому форматі, залежно від потреб і вибраного інструменту.
2. **Вибір інструменту:** Оберіть інструмент або бібліотеку, яку ви будете використовувати для реалізації парсера. Існує багато доступних варіантів, таких як ANTLR, PLY, YACC, регулярні вирази або навіть вбудовані можливості деяких мов програмування.
3. **Реалізація парсера:** Реалізуйте парсер, використовуючи вибраний інструмент або бібліотеку. Це може включати визначення правил граматики, створення лексичного аналізатора для розпізнавання токенів та створення синтаксичного аналізатора для розпізнавання синтаксичних правил та створення синтаксичного дерева.
4. **Тестування парсера:** Після реалізації парсера важливо провести тестування для перевірки його працездатності. Створіть набір тестових вхідних даних, які покривають різні сценарії використання, включаючи правильні та неправильні дані. Перевірте, чи правильно парсер розпізнає структуру даних та чи повертає очікувані результати.
5. **Опрацювання результатів:** Після успішного розпізнавання структури даних парсером ви можете обробляти та використовувати отримані результати за своїми

потребами. Це може включати валідацію, збереження до бази даних, витягування потрібної інформації або подальший аналіз.

6. **Вдосконалення та оптимізація:** Якщо потрібно, можна майже завжди вдосконалити й оптимізувати роботу парсера. Це може включати оптимізацію швидкодії, розширення підтримуваних функцій, обробку виняткових ситуацій або вдосконалення гнучкості та надійності.

Загалом створення парсера є складним завданням, особливо при роботі зі складною граматикою або великими обсягами даних. Важливо мати розуміння синтаксичних правил і граматики вхідних даних, а також знати інструменти і технології, які ви використовуєте для реалізації парсера.

1.3 Які мови підійдуть для написання парсеру

Взагалі є багато мов програмування, які підходять для написання парсерів, і вибір конкретної мови залежить від вашого досвіду, потреб проекту та екосистеми інструментів, які вам потрібні. Ось кілька популярних мов, які використовуються для створення парсерів:

1. **Python:** [4][7]Python є популярною мовою програмування для розробки парсерів. Вона має багато бібліотек, таких як PLY (Python Lex-Yacc), ANTLR, pyparsing та інші, які полегшують створення парсерів. Python має простий і зрозумілий синтаксис, що робить його популярним вибором для початківців та досвідчених розробників.
2. **Java:** [6]Вона також є однією з найпопулярніших мов для створення парсерів. Вона має потужну екосистему інструментів, таких як ANTLR, JavaCC, JFlex, які надають широкі можливості для створення парсерів. Java є мовою програмування загального призначення, що робить її популярною для великих проектів та підприємственого рівня.
3. **C/C++:** [5]Дані мови є популярними виборами для написання швидких та ефективних парсерів. Вони надають прямий доступ до пам'яті та дозволяють

оптимізувати швидкодію парсера. Для реалізації парсерів у мовах C/C++ можна використовувати бібліотеки, такі як Bison, Flex, Boost.Spirit та багато інших.

4. **JavaScript:** [6]JavaScript широко використовується веб-розробкою, і він також може бути використаний для створення парсерів. За допомогою бібліотек, таких як PEG.js, nearley, chevrotain та інших, можна створювати парсери, які працюють безпосередньо у веб-браузерах або на серверних платформах.
5. **Ruby:** Ruby є елегантною та легкою у використанні мовою програмування, яка також має багато інструментів для створення парсерів. Бібліотеки, такі як Racc, Treetop, Parslet та інші, дозволяють створювати парсери з високою рівнем абстракції та експресивності.
6. **Go:** Мова програмування Go стала все більш популярною для розробки парсерів. Вона має ефективну систему збирання сміття, швидку компіляцію та вбудовану підтримку конкурентності, що робить її привабливою для створення швидких та надійних парсерів.

Це лише найбільш популярні приклади мов програмування, які можна використовувати для створення парсерів. Вибір мови залежить від особистих уподобань, проектних вимог та досвіду у конкретній мові.

2. ПОСТАНОВКА ЗАДАЧІ

2.1 Визначення зручного сайту для демонстрації

Загалом кажучи, простіше всього спарсити статичні сайти, які мають просту структуру та не використовують складних механізмів динамічного відображення контенту. Ось деякі приклади таких сайтів:

1. **Сайти, побудовані на основі HTML:** [1]Більшість веб-сайтів використовують HTML для структури та відображення контенту. Статичні сайти, які мають просту HTML-структуру без складних JavaScript-інтерактивностей, можуть бути легко спарсені. Такі сайти можуть включати блоги, статичні сторінки, новинні сайти тощо.
2. **Веб-каталоги та оголошення:** Сайти, які містять каталоги товарів або послуг, а також оголошення, часто мають структурований контент, що полегшує парсинг. Наприклад, сайти з автомобільними оголошеннями, нерухомістю, туристичними пропозиціями та інші подібні ресурси можуть бути досить простими для спарсингу.
3. **Вікіпедія та інші веб-енциклопедії:** Веб-енциклопедії, такі як Вікіпедія, зазвичай мають структурований контент зі статичними сторінками, які містять інформацію про різні теми. Оскільки ці сайти мають чітко визначену структуру і формат, парсинг статей та отримання інформації можуть бути досить простими завданнями.
4. **Новинні сайти:** Багато новинних сайтів надають структуровану інформацію про статті, заголовки, авторів та дати публікацій. Якщо структура сайту не змінюється часто, парсинг новинних сайтів може бути відносно простим.
5. **Веб-сайти з відкритим API:** Деякі веб-сайти надають відкриті API для доступу до свого контенту. Це може включати соціальні мережі, блогінгові платформи, сервіси електронної комерції та багато інших. Використання API дозволяє отримувати структуровану інформацію без необхідності парсити HTML-код веб-сторінок.

Це були перечислені приклади сайтів, які є найбільш зручними для парсингу. За своїм бажанням я обрав сайт-енциклопедію мотоциклів марки KTM –

“https://www.ktm.com/en-int/”. Даний сайт не блокує дію парсерів і є відкритим для усіх користувачів.

2.2 Вибір мови програмування серед C/C++/C#

Коли мова програмування стосується парсингу сайтів, кожна з мов C/C++, C# має свої переваги та особливості. Ось огляд кожної з цих мов:

- C/C++: [2]Ці мови є дуже потужними і широко використовуються в програмуванні загального призначення. Вони мають прямий доступ до пам'яті та можуть бути корисними при парсингу великих обсягів даних, особливо якщо швидкодія є критичним фактором. Крім того, існує багато бібліотек, таких як libxml2 та Gumbo, які надають потужні засоби для парсингу HTML та XML. Однак, парсинг сайтів за допомогою C/C++ може вимагати більшої кількості коду і деталей, оскільки ці мови є низькорівневими. Проте є один недолік сервісу Replit, який не дозволяє додавати модулі до цих мов програмування (принаймні під час виконання даної роботи), тому написання парсера даними мовами на цьому сайті є поки що неможливим.
- C#: Дана мова є об'єктно-орієнтованою мовою програмування, яка має широку підтримку у середовищі .NET. Вона пропонує багато функціональностей та інструментів, які полегшують розробку парсерів. У .NET-екосистемі є багато бібліотек для парсингу, наприклад, HtmlAgilityPack, AngleSharp та CsQuery. Крім того, C# має зручний синтаксис та багато вбудованих можливостей для роботи з рядками і регулярними виразами, що сприяє ефективному парсингу. На сайті Replit є функція додавання модулів для мови C#, саме тому вона й буте чудовим вибором для написання парсеру.

Отже, використовуючи будь-яку з цих трьох мов, можна створити парсер, але через особливості сервісу Replit, це можливо буде зробити лише мовою C#.

3. СТВОРЕННЯ ПАРСЕРУ МОВОЮ C#

3.1 Як зібрати дані з інтернету за допомогою C# у Replit

[3] Збір даних з Інтернету за допомогою C# може бути виконаним за допомогою різних підходів та інструментів. Веб-скрапінг означає видобування даних з веб-сторінок шляхом аналізу їх HTML-коду. У C# можна використовувати бібліотеки, такі як HtmlAgilityPack або AngleSharp, щоб завантажувати веб-сторінки, отримувати доступ до їх елементів та видобувати потрібні дані. Можна шукати елементи за допомогою CSS-селекторів або XPath-виразів, а потім отримувати значення атрибутів, тексту або інших даних з цих елементів.

Важливу увагу потрібно приділити збереженню даних. Для гарного оформлення результату виконання коду, можна додати запис назв моделей мотоциклів, та силки на них у файл .txt. Також бажано додати друк усіх записаних даних у консоль. Ще можна додати збереження html коду сторінки, це дасть змогу передивитися через деякий час, що саме змінилося, або виконати програму без доступу в інтернет.

3.2 Підключення необхідних бібліотек

У коді будуть використовуватиса дані бібліотеки:

```
1   using System;
2   using System.Collections.Generic;
3   using System.IO;
4   using System.Net.Http;
5   using System.Threading.Tasks;
6   using HtmlAgilityPack;
```

Розглянемо їх детальніше й для чого вони потрібні:

1. **HtmlAgilityPack:** Дана бібліотека є потужним інструментом для роботи з HTML-кодом. Вона надає зручний спосіб отримання доступу до елементів,

атрибутів та тексту веб-сторінок. Ця бібліотека дозволяє використовувати CSS-селектори або XPath-вирази для пошуку потрібних елементів на сторінці. Ви можете отримувати дані, видаляти або змінювати елементи, а також виконувати багато інших операцій з HTML. HtmlAgilityPack є дуже популярним інструментом для парсингу веб-сторінок у C#.

2. **System.Net.Http:** Вона є частиною стандартної бібліотеки .NET і використовується для взаємодії з веб-серверами за допомогою протоколу HTTP. Вона містить клас HttpClient, який дозволяє виконувати HTTP-запити до веб-серверів і отримувати їх відповіді. HttpClient забезпечує можливість завантажувати HTML-сторінки, JSON-дані та інші ресурси з Інтернету. Завдяки цій бібліотеці ви можете здійснювати запити GET, POST, PUT, DELETE та інші типи запитів до веб-серверів.
3. **System.IO:** System.IO є ще однією стандартною бібліотекою .NET і використовується для роботи з файлами та потоками даних. У контексті збору даних з Інтернету, System.IO може бути використана для збереження отриманих даних у файл або для зчитування збережених файлів. Наприклад, ви можете використовувати клас StreamReader для читання HTML-сторінок з файлу або клас StreamWriter для запису даних у файл.

Усі ці бібліотеки є потужними інструментами для збору даних з Інтернету використовуючи C#. Вони надають широкі можливості для отримання, аналізу та збереження даних з веб-сторінок. Завдяки їх функціональності ви можете легко взаємодіяти зі сторінками, отримувати вміст, видобувати потрібні дані та здійснювати інші операції, які необхідні для вашого проекту зі збору даних з Інтернету.

3.3 Робота коду, блок-схема та результат

Для початку розглянемо створений код і розберемо його більш детально:

```
// Вхідна точка початку коду
class Program
{
    static async Task Main(string[] args)
    {
        try // Конструкція, яка в даному випадку покаже, що саме трапилося та продовжить виконання коду
        {
            // Завантажуємо HTML-код сторінки сайту KTM
            string url = "https://www.ktm.com/en-int.html";
            using (HttpClient client = new HttpClient())
            {
                // Виклик HML із сайту
                string html = await client.GetStringAsync(url);

                // Записуємо HTML-код у файл
                File.WriteAllText("data.html", html);
                Console.WriteLine("HTML code saved to 'data.html'");
```

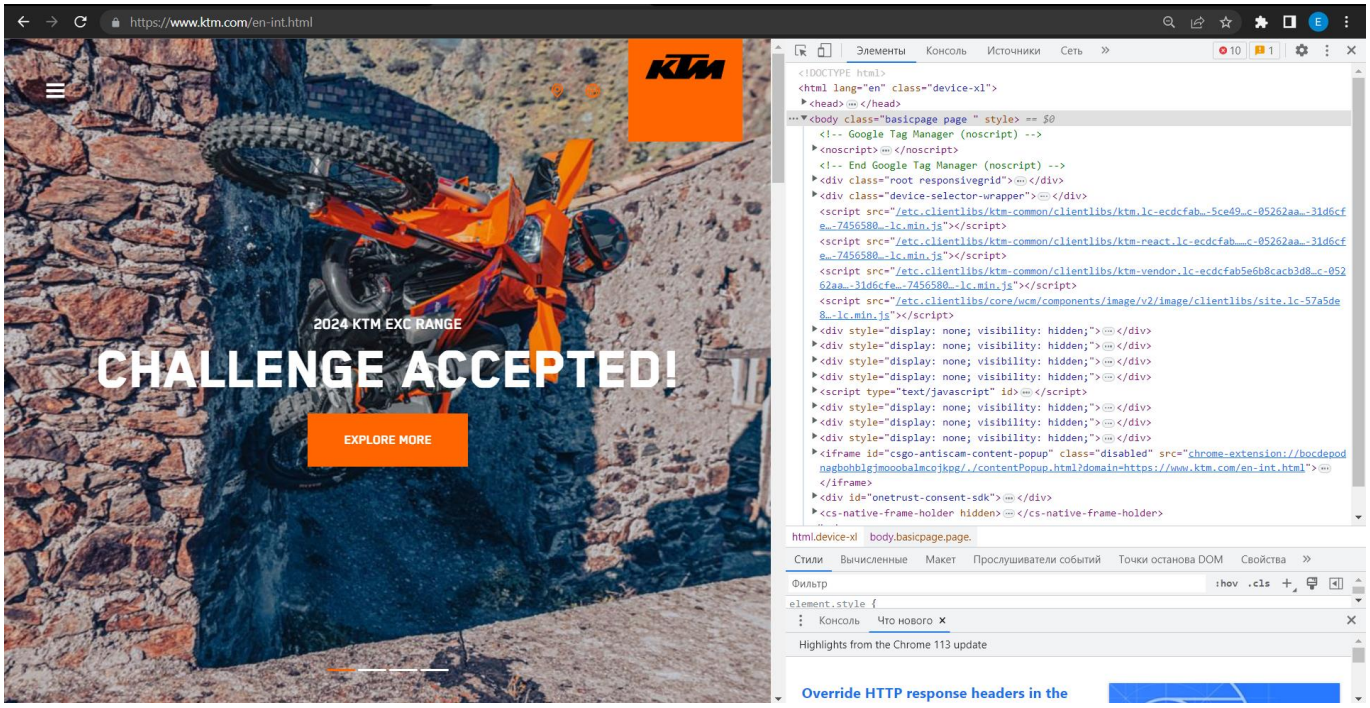
Тут ми маємо вхідну точку програми, задаємо силку на вебсторінку сайту та використовуючи конструкцію try: робимо виклик html коду з сайту. У результаті отриманий код записується у файл “data.html” та виводиться відповідне сповіщення.

```
> dotnet run
HTML code saved to 'data.html'
```

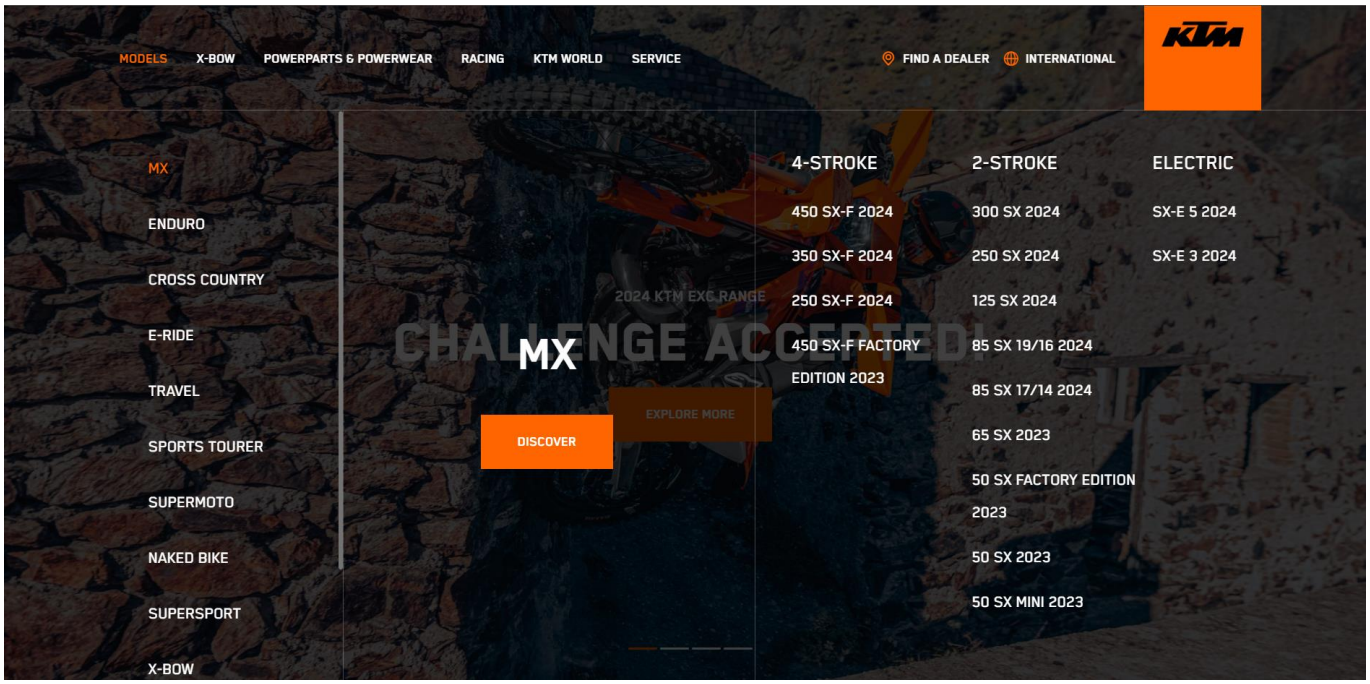
```
                // Завантажуємо HTML з файлу
                string savedHtml = File.ReadAllText("data.html");

                // Виконуємо парсинг HTML
                HtmlDocument doc = new HtmlDocument();
                doc.LoadHtml(savedHtml);
```

Після збереження html коду, відкриваємо його та починаємо використовувати бібліотеку HtmlAgilityPack, яка дозволяє витягувати необхідну інформацію з сайту.

Тепер розглянемо сам сайт:

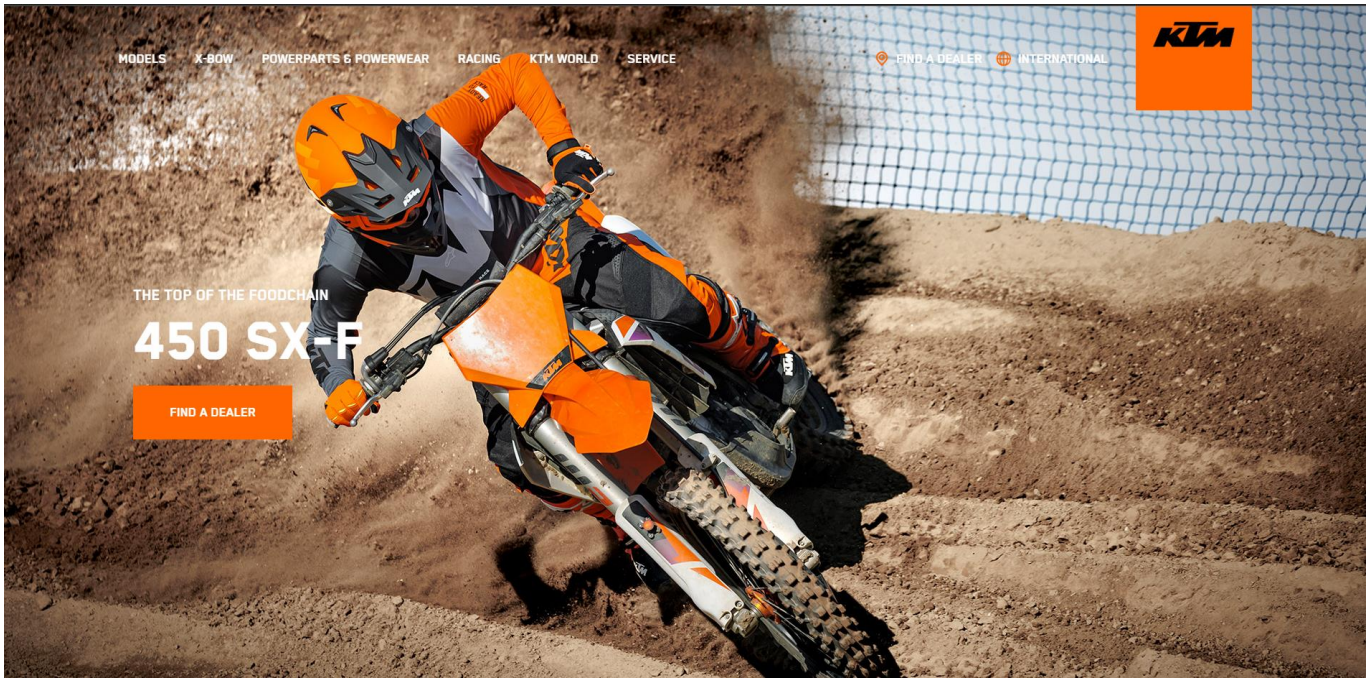


Даний сайт має кнопку меню:

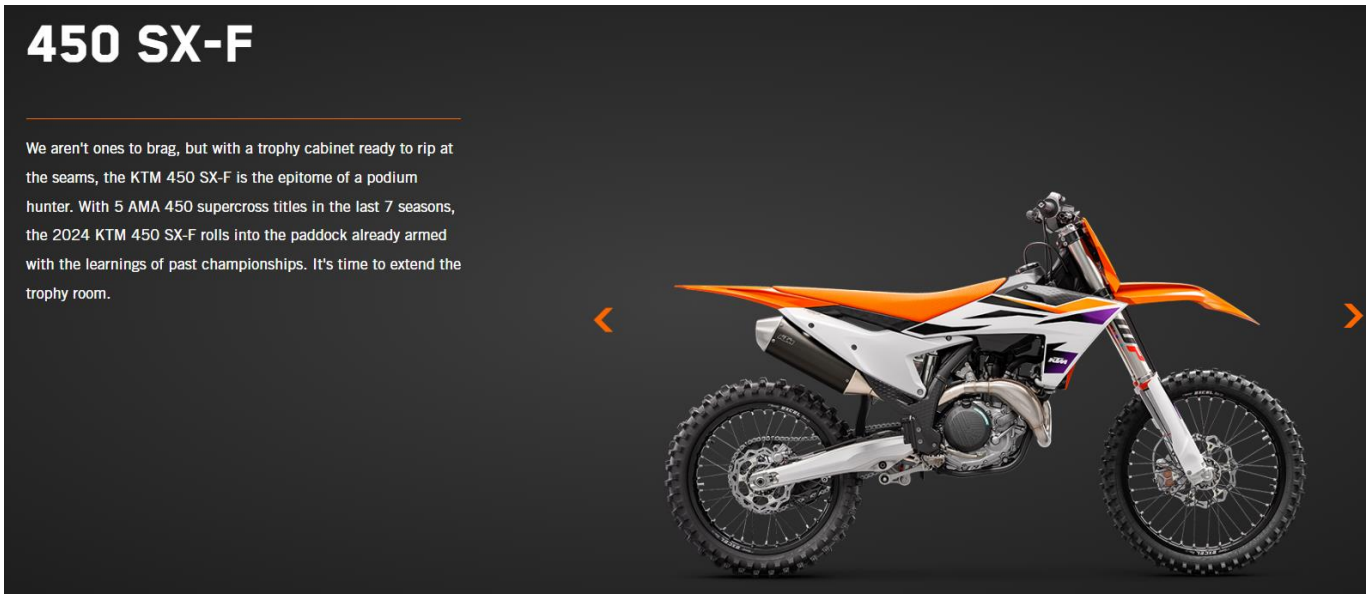


Після відкриття якої, ми бачимо вкладку MODELS, де є різні категорії транспорту. Натиснувши на першу – MX, ми отримуємо велику кількість мотоциклів.

Виберемо 450 SX-F 2024:



Результат – перехід на сторінку даного транспорту, у якого є багато розписаної індивідуальної інформації:



TOTAL TRACTION

SELECTABLE MAPS + TC

The map select switch on the 2024 KTM SX models features a simpler design, allowing for easier functionality in switching between 2 engine maps. Map 1 provides a more linear power curve. while map 2 punches all-out with aggressive throttle and explosive power. Luckily, traction control is easy toggled to an on or off position from the same switch, ensuring maximum traction and a distinct advantage in wet or muddy conditions.

TECHNICAL DETAILS

ENGINE

TRANSMISSION	5-speed
STARTER	Electric starter
STROKE	63.4 mm
BORE	95 mm
CLUTCH	Wet, DDS multi-disc clutch, Brembo hydraulics
DISPLACEMENT	449.9 cm^3
EMS	Keihin EMS
DESIGN	1-cylinder, 4-stroke engine

CHASSIS

WEIGHT (WITHOUT FUEL)	102.6 kg
TANK CAPACITY (APPROX.)	7.2 l
FRONT BRAKE DISC DIAMETER	260 mm
REAR BRAKE DISC DIAMETER	220 mm
FRONT BRAKE	Disc brake
REAR BRAKE	Disc brake
CHAIN	520, Non-sealed
FRAME DESIGN	Central double-cradle-type 25CrMo4 steel

Кожен транспортний засіб має свою власну сторінку. Тепер потрібно знайти в HTML коді, де заховані назви та силки, які нам потрібно дістати.

```
// Знаходимо назви моделей
var modelNamesNode = doc.DocumentNode.SelectSingleNode("//div[@class='content']//ul//li");
string modelNames = modelNamesNode?.InnerText.Trim();

// Знаходимо посилання на моделі
var modelLinksNodes = doc.DocumentNode.SelectNodes("//ul[@class='segments-list js-segment-list']//a");
// Список, у який буду додавати моделі
List<Dictionary<string, string>> models = new List<Dictionary<string, string>>();

// Перевірка на існування об'єктів
if (modelLinksNodes != null)
{
    int nameIndex = 0;
    foreach (var node in modelLinksNodes)
    {
        string modelName = node.InnerText.Trim();
        string modelLink = node.GetAttributeValue("href", "");

        // Фільтрація зайвої силки DISCOVER та додання елементів
        if (modelName != "DISCOVER" && modelName != modelNames.Split('\n')[nameIndex])
        {
            Dictionary<string, string> model = new Dictionary<string, string>
            {
                { "Model name", modelName },
                { "Link", modelLink }
            };
            models.Add(model);
        }
        else
        {
            nameIndex++;
        }
    }
}
```

Розглянемо даний фрагмент коду більш детально:

1. **Знаходження назв моделей:** Використовуючи XPath-вираз //div[@class='content']//ul//li, програма шукає перший вузол <li> відповідної структури в HTML-документі. Цей вузол містить назви моделей. Отриманий текст з цього вузла очищується від зайвих пробілів та зберігається у змінну modelNames.
2. **Знаходження посилань на моделі:** Використовуючи XPath-вираз //ul[@class='segments-list js-segment-list']//a, програма знаходить всі вузли <a> відповідної структури в HTML-документі. Ці вузли містять посилання на моделі. Отримані вузли зберігаються у змінну modelLinksNodes.
3. **Створення списку моделей:** Оголошується порожній список models, який буде містити словники з назвами моделей та відповідними посиланнями.
4. **Перевірка на існування об'єктів:** Перевіряється, чи знайдено посилання на моделі. Якщо modelLinksNodes не дорівнює null, то виконується код всередині блоку if.

5. **Ітерація по знайдених посиланнях:** Використовуючи цикл foreach, програма проходить по кожному вузлу <a> у modelLinksNodes. Для кожного вузла отримується текст назви моделі та значення атрибуту "href" (посилання). Отримані значення очищуються від зайвих пробілів та зберігаються у змінні modelName та modelLink відповідно.
6. **Фільтрація непотрібних посилань:** Виконується перевірка, чи назва моделі не є "DISCOVER" і чи не збігається з назвою моделі, знайденою раніше у змінній modelNames. Якщо ці умови виконуються, створюється словник model, де ключ "Model name" містить назву моделі, а ключ "Link" містить відповідне посилання. Цей словник додається до списку models.
7. **Інкрементування nameIndex:** Якщо назва моделі збігається зі значенням, знайденим у modelNames, значення nameIndex збільшується на 1. Це забезпечує правильну відповідність між назвами моделей та їх посиланнями.

Після виконання цього фрагменту коду, список models буде містити словники з назвами моделей та відповідними посиланнями на сторінки моделей, які були вилучені та відфільтровані з HTML-документа. Рухаємося до наступної частини коду.

```
                // Виводимо результат із урахуванням фільтрації
                List<string> excludedModels = new List<string> { "MX", "Enduro", "E-Ride", "Travel", "Sports Tourer",
    "Supermoto", "Naked Bike", "Supersport", "X-BOW", "BRABUS", "Cross Country" };
                using (StreamWriter writer = new StreamWriter("Models links.txt"))
                {
                    foreach (var model in models)
                    {
                        if (excludedModels.Contains(model["Model name"]))
                        {
                            // Виводимо назву категорії, якщо модель є в списку виключень
                            string categoryName = "\n\nCategory name: " + model["Model name"] + "\n";
                            Console.WriteLine(categoryName);
                            writer.WriteLine(categoryName);
                        }
                        else
                        {
                            // Виводимо назву моделі та посилання, якщо модель не входить до списку виключень
                            string modelName = "- Model name: " + model["Model name"];
                            string modelLink = "  Link: " + model["Link"];
                            Console.WriteLine(modelName);
                            Console.WriteLine(modelLink);
                            writer.WriteLine(modelName);
                            writer.WriteLine(modelLink);
                        }
                    }
                }
            }
            catch (Exception ex)  // Друк можливої помилки
            {
                Console.WriteLine("Error: " + ex.Message);
            }
        }
    }
```

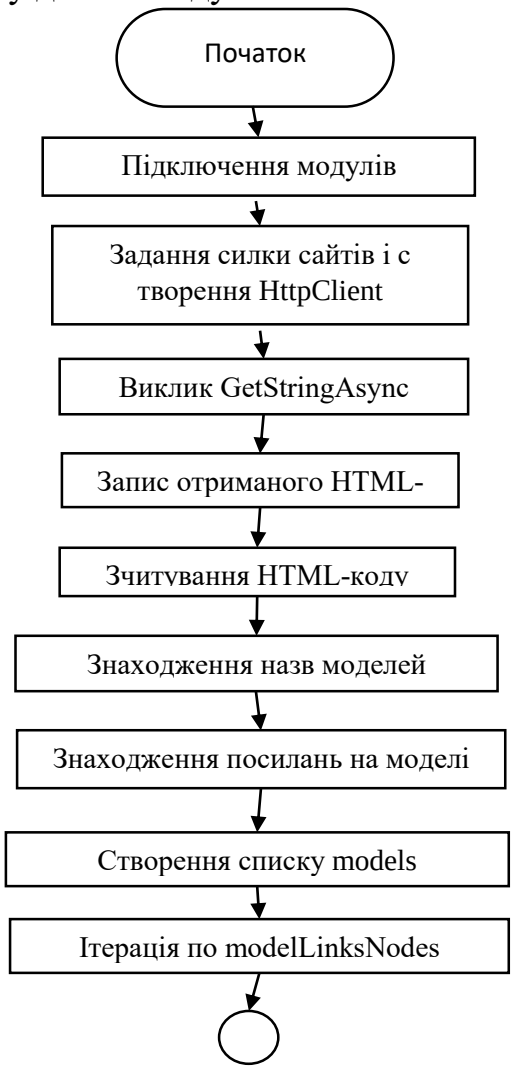
Розглянемо другий основний фрагмент коду більш детально:

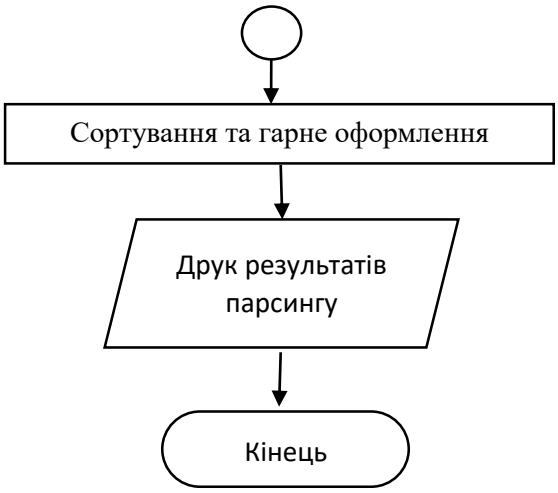
1. **Створення списку excludedModels:** Створюється список excludedModels, який містить назви моделей, які будуть виключені з результатів. Ці моделі не будуть виводитись або зберігатись в файлі.
2. **Виведення результатів з урахуванням фільтрації:** Використовуючи об'єкт StreamWriter, створюється файл "Models links.txt", до якого будуть записуватись результати.
3. **Ітерація по списку models:** Використовуючи цикл foreach, програма проходить по кожному словнику model у списку models.
4. **Перевірка наявності моделі в списку виключень:** Для кожної моделі перевіряється, чи міститься її назва в списку excludedModels за допомогою методу Contains(). Якщо модель знаходиться у списку виключень, виконується код всередині блоку if. Це було створено для вирішення проблеми негарного виводу результату після опрацювання даних усіх моделей.

5. **Виведення назви категорії:** Якщо модель знаходиться у списку виключень, генерується рядок categoryName, який містить назву категорії моделі. Цей рядок виводиться на консоль за допомогою Console.WriteLine() та записується у файл "Models links.txt" за допомогою writer.WriteLine().
6. **Виведення назви моделі та посилання:** Якщо модель не знаходиться у списку виключень, генеруються рядки modelName та modelLink, які містять відповідно назву моделі та посилання. Ці рядки виводяться на консоль та записуються у файл "Models links.txt".

Після виконання коду, на консоль будуть виведені назви категорій моделей, які знаходяться у списку виключень, а також назви моделей та відповідні посилання, які не знаходяться у списку виключень. Усі ці результати також будуть збережені у файлі "Models links.txt". У разі виникнення помилки під час виконання програми, виведеться повідомлення про помилку на консоль.

Розглянемо блок-схему даного коду:

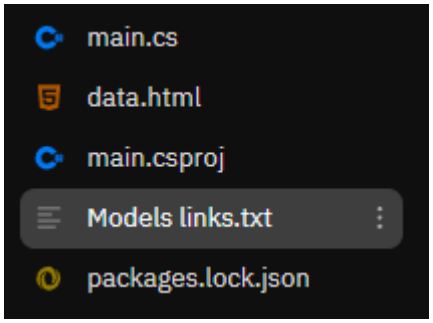





```
> dotnet run
HTML code saved to 'data.html'


Category name: MX

- Model name: 450 SX-F 2024
  Link: https://www.ktm.com/en-int/models/mx/4-stroke/450-sx-f-2024.html
- Model name: 350 SX-F 2024
  Link: https://www.ktm.com/en-int/models/mx/4-stroke/350-sx-f-2024.html
- Model name: 250 SX-F 2024
  Link: https://www.ktm.com/en-int/models/mx/4-stroke/250-sx-f-2024.html
- Model name: 450 SX-F FACTORY EDITION 2023
  Link: https://www.ktm.com/en-int/models/mx/4-stroke/ktm-450-sx-f-factoryedition2023
.html
- Model name: 300 SX 2024
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/300-sx-2024.html
- Model name: 250 SX 2024
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/250-sx-2024.html
- Model name: 125 SX 2024
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/125-sx-2024.html
- Model name: 85 SX 19/16 2024
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/85-sx-19-16-2024.html
- Model name: 85 SX 17/14 2024
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/85-sx-17-14-2024.html
- Model name: 65 SX 2023
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/ktm-65-sx-2023.html
- Model name: 50 SX FACTORY EDITION 2023
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/ktm-50-sx-factoryedition2023.ht
ml
- Model name: 50 SX 2023
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/ktm-50-sx-2023.html
- Model name: 50 SX MINI 2023
  Link: https://www.ktm.com/en-int/models/mx/2-stroke/ktm-50-sx-mini-2023.html
- Model name: SX-E 5 2024
  Link: https://www.ktm.com/en-int/models/mx/electric/ktm-sx-e-5-2024.html
- Model name: SX-E 3 2024
  Link: https://www.ktm.com/en-int/models/mx/electric/ktm-sx-e-3-2024.html


Category name: Enduro

- Model name: 500 EXC-F 2024
  Link: https://www.ktm.com/en-int/models/enduro/4-stroke/ktm-500-exc-f-2024.html
- Model name: 450 EXC-F 2024
  Link: https://www.ktm.com/en-int/models/enduro/4-stroke/ktm-450-exc-f-2024.html
- Model name: 350 EXC-F 2024
  Link: https://www.ktm.com/en-int/models/enduro/4-stroke/ktm-350-exc-f-2024.html
- Model name: 250 EXC-F 2024
```

3.4 Збереження зібраних даних

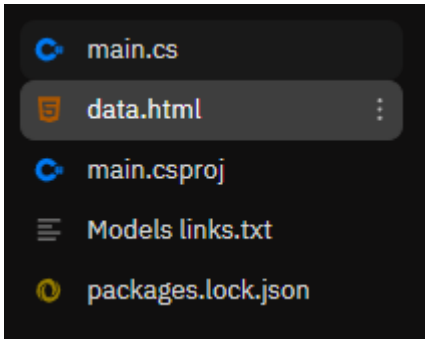
Усі дані зберігаються у файл .txt:




```
1
2
3  Category name: MX
4
5  - Model name: 450 SX-F 2024
6    Link: https://www.ktm.com/en-int/models/mx/4-stroke/450-sx-f-2024.html
7  - Model name: 350 SX-F 2024
8    Link: https://www.ktm.com/en-int/models/mx/4-stroke/350-sx-f-2024.html
9  - Model name: 250 SX-F 2024
10   Link: https://www.ktm.com/en-int/models/mx/4-stroke/250-sx-f-2024.html
11 - Model name: 450 SX-F FACTORY EDITION 2023
12   Link: https://www.ktm.com/en-int/models/mx/4-stroke/ktm-450-sx-f-factoryedition2023.html
13 - Model name: 300 SX 2024
14   Link: https://www.ktm.com/en-int/models/mx/2-stroke/300-sx-2024.html
15 - Model name: 250 SX 2024
16   Link: https://www.ktm.com/en-int/models/mx/2-stroke/250-sx-2024.html
17 - Model name: 125 SX 2024
18   Link: https://www.ktm.com/en-int/models/mx/2-stroke/125-sx-2024.html
19 - Model name: 85 SX 19/16 2024
20   Link: https://www.ktm.com/en-int/models/mx/2-stroke/85-sx-19-16-2024.html
21 - Model name: 85 SX 17/14 2024
22   Link: https://www.ktm.com/en-int/models/mx/2-stroke/85-sx-17-14-2024.html
23 - Model name: 65 SX 2023
24   Link: https://www.ktm.com/en-int/models/mx/2-stroke/ktm-65-sx-2023.html
25 - Model name: 50 SX FACTORY EDITION 2023
26   Link: https://www.ktm.com/en-int/models/mx/2-stroke/ktm-50-sx-factoryedition2023.html
27 - Model name: 50 SX 2023
28   Link: https://www.ktm.com/en-int/models/mx/2-stroke/ktm-50-sx-2023.html
29 - Model name: 50 SX MINI 2023
30   Link: https://www.ktm.com/en-int/models/mx/2-stroke/ktm-50-sx-mini-2023.html
31 - Model name: SX-E 5 2024
32   Link: https://www.ktm.com/en-int/models/mx/electric/ktm-sx-e-5-2024.html
33 - Model name: SX-E 3 2024
34   Link: https://www.ktm.com/en-int/models/mx/electric/ktm-sx-e-3-2024.html
```

На фото лише частина від усіх даних

Також розглянемо html код:



data.html

```
1
2  <!DOCTYPE HTML>
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8"/>
6
7
8
9          <title>KTM - READY TO RACE</title>
10
11
12
13     <meta name="template" content="ktm-language-template"/>
14     <meta name="viewport" content="width=device-width, initial-scale=1"/>
15
```

```
2473  </body>
2474  </html>
2475
```

Саме звідси були витягнуті всі силки та назви нових моделей мотоциклів марки KTM.

ВИСНОВКИ

У результаті виконання даної курсової роботи, я створив програму, яка демонструє процес завантаження HTML-коду з веб-сторінки, його парсинг і витягування певних даних, а також збереження результатів у файл. Даний код виконує наступні дії:

- Завантажує сторінку сайту KTM за допомогою HTTP-запиту і отримує HTML-код сторінки.
- Записує отриманий HTML-код у файл "data.html".
- Зчитує HTML-код з файлу "data.html" у змінну.
- Використовуючи бібліотеку HtmlAgilityPack, парсить HTML-код та створює об'єкт HtmlDocument.
- Знаходить назви моделей та посилання на моделі на сторінці за допомогою відповідних XPath-запитів.
- Фільтрує отримані назви моделей, виключаючи деякі моделі зі списку excludedModels.
- Записує відфільтровані назви моделей та посилання у списку models.
- Виводить результат на консоль та записує його у файл "Models links.txt". Назви моделей, які знаходяться у списку excludedModels, виводяться як назви категорій.

Отже, цей код виконує завдання з отримання інформації про моделі зі сторінки сайту KTM, фільтрує їх та зберігає результат у файл. Ви можете використовувати цей код для отримання даних про моделі для вашої курсової роботи.

Використані джерела

1. Concurrency in C# Cookbook Asynchronous, Parallel, and Programming Multithreaded. Stephen Cleary 2nd edition. Дата публікації: жовтень 2019. Посилання на ресурс: https://sd.blackball.lv/downloadfile.ashx?file=18701
2. C# 10 in a Nutshell: The Definitive Reference. 1st Ed. Joseph Albahari 1st edition. Дата публікації березень 2022. Посилання на ресурс: https://dl.ebooksworld.ir/books/CSharp.10.in.a.Nutshell.Joseph.Albahari.OReilly.9781098121952.EBooksWorld.ir.pdf
3. C# in Depth: Fourth Edition 4th Edition. Jon Skeet. Дата публікації: березень 2019. Джерело використаної інформації: https://dl.ebooksworld.ir/motoman/Manning.Csharp.in.Depth.4th.Edition.www.EBooksWorld.ir.pdf
4. Web Scraping with Python: Collecting More Data from the Modern Web 2nd Edition. Ryan Mitchell. Дата публікації: травень 2018. Джерело використаної інформації: https://edu.anarcho-copy.org/Programming%20Languages/Python/Web%20Scraping%20with%20Python,%202nd%20Edition.pdf
5. C# Programming in Easy Step. Mike McGrath. Рік видання 2016. Джерело використаної інформації: https://docplayer.net/95817164-Mike-mcgrath-c-programming.html
6. Data Visualization with Python and JavaScript: Scrape, Clean, Explore & Transform Your Data 1st Edition. Kyran Dale. Дата публікації: серпень 2016. Джерело використаної інформації: https://www.kyrandale.com/assets/063851ce7ba3e5ed2e6b0a1fa2e95c326d6bfbfde44b09124b9f1df6d48af15b.pdf
7. Automate the Boring Stuff with Python, 2nd Edition. Al Sweigart. Дата публікації: листопад 2019. Джерело використаної інформації: https://electrovolt.ir/wp-content/uploads/2017/07/Automate_The_Boring_Stuff_With_Python_ElectroVolt.ir_.pdf

Додаток

Текст програми (https://replit.com/join/ggoozyuwnb-tr-25-kudriavts):

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using HtmlAgilityPack;
// Бібліотеки (частину потрібно завантажити)

// Вхідна точка початку коду
class Program
{
    static async Task Main(string[] args)
    {
        try // Конструкція, яка в даному випадку покаже, що саме трапилося та продовжить виконання коду
        {
            // Завантажуємо HTML-код сторінки сайту KTM
            string url = "https://www.ktm.com/en-int.html";
            using (HttpClient client = new HttpClient())
            {
                // Виклик HML із сайту
                string html = await client.GetStringAsync(url);

                // Записуємо HTML-код у файл
                File.WriteAllText("data.html", html);
                Console.WriteLine("HTML code saved to 'data.html'");

                // Завантажуємо HTML з файлу
                string savedHtml = File.ReadAllText("data.html");

                // Виконуємо парсинг HTML
                HtmlDocument doc = new HtmlDocument();
                doc.LoadHtml(savedHtml);
```

```
// Знаходимо назви моделей
var modelNamesNode = doc.DocumentNode.SelectSingleNode("//div[@class='content']//ul//li");
string modelNames = modelNamesNode?.InnerText.Trim();

// Знаходимо посилання на моделі
var modelLinksNodes = doc.DocumentNode.SelectNodes("//ul[@class='segments-list js-segment-list']//a");
// Список, у який буду додавати моделі
List<Dictionary<string, string>> models = new List<Dictionary<string, string>>();

// Перевірка на існування об'єктів
if (modelLinksNodes != null)
{
    int nameIndex = 0;
    foreach (var node in modelLinksNodes)
    {
        string modelName = node.InnerText.Trim();
        string modelLink = node.GetAttributeValue("href", "");

        // Фільтрація зайвої силки DISCOVER та додання елементів
        if (modelName != "DISCOVER" && modelName != modelNames.Split('\n')[nameIndex])
        {
            Dictionary<string, string> model = new Dictionary<string, string>
            {
                { "Model name", modelName },
                { "Link", modelLink }
            };
            models.Add(model);
        }
        else
        {
            nameIndex++;
        }
    }
}

// Виводимо результат із урахуванням фільтрації
List<string> excludedModels = new List<string> { "MX", "Enduro", "E-Ride", "Travel", "Sports Tourer", "Supermoto", "Naked Bike", "Supersport", "X-BOW", "BRABUS", "Cross Country" };
using (StreamWriter writer = new StreamWriter("Models links.txt"))
```

```
            {
                foreach (var model in models)
                {
                    if (excludedModels.Contains(model["Model name"]))
                    {
                        // Виводимо назву категорії, якщо модель є в списку виключень
                        string categoryName = "\n\nCategory name: " + model["Model name"] + "\n";
                        Console.WriteLine(categoryName);
                        writer.WriteLine(categoryName);
                    }
                    else
                    {
                        // Виводимо назву моделі та посилання, якщо модель не входить до списку виключень
                        string modelName = "- Model name: " + model["Model name"];
                        string modelLink = "  Link: " + model["Link"];
                        Console.WriteLine(modelName);
                        Console.WriteLine(modelLink);
                        writer.WriteLine(modelName);
                        writer.WriteLine(modelLink);
                    }
                }
            }
        }
        catch (Exception ex)  // Друк можливої помилки
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
}
```