# Съдържание

[1. Съдържание 0](#_Toc133507953)

[2. Въведение 1](#_Toc133507954)

[3. Популярни инструменти и техники, използвани в Python за реализиране на парсинг на сайтове 1](#_Toc133507955)

[1. Scraping 1](#_Toc133507956)

[a) Описание 2](#_Toc133507957)

[b) История 3](#_Toc133507958)

[c) Софтуер 4](#_Toc133507959)

[d) Уеб скрапинг 4](#_Toc133507960)

[e) Технически варианти 5](#_Toc133507961)

[f) Извличане на уеб 7](#_Toc133507962)

[g) Отчет за добив 7](#_Toc133507963)

[4. Аргументиран избор на технология за реализиране на парсинг 8](#_Toc133507964)

[1. Изграждане на уеб скрепер: 8](#_Toc133507965)

[a) HTML структури 8](#_Toc133507966)

[b) Основи на Python 9](#_Toc133507967)

[c) Виртуални среди на Python 11](#_Toc133507968)

[d) Библиотеки на Python 12](#_Toc133507969)

[e) Импортиране на инсталирани библиотеки 13](#_Toc133507970)

[f) Библиотека за заявки на Python 13](#_Toc133507971)

[2. Beautiful Soup 14](#_Toc133507972)

[a) Атрибути с множество стойности 15](#_Toc133507973)

[b) Специални низове 17](#_Toc133507974)

[c) HTML документи 18](#_Toc133507975)

[d) Requests 18](#_Toc133507976)

[3. HTTP 19](#_Toc133507977)

[a) Методи: 20](#_Toc133507978)

[b) Използване на GET заявка 24](#_Toc133507979)

[c) Използване на GET параметри 24](#_Toc133507980)

[5. Приложение за парсинг на сайтове 26](#_Toc133507981)

[6. Заключение: 30](#_Toc133507982)

# Въведение

Аз съм любител на спорта , много често ми се налага да влизам в интернет за да проверявам датите и резултатите на определени мачове. След като получих своето задание за дипломен проект се зарадвах защото мага да направя нещото което ми се налага да използвам постоянно. Приложение за извличане на информация от сайтове, на което аз мога да му задам данни от някои сайт, като например Sportal.Тoва в моя случаи ще се получи чрез web scraping.Web scraing с други думи е събирач на информация като избереш уебсайт тои започва да прелиства кода на самият сайт и да търси сходство със кода на даденият уебсайт и предварително зададени от мен данни в кода на моето приложение.Чрез него аз мога да направя моето приложение, със информация само за отборите които аз харесвам.С две думи искам да създам приложение което да намира вместо мен информация за отборите които харесвам.

# Популярни инструменти и техники, използвани в Python за реализиране на парсинг на сайтове

## Scraping

Уеб скрапинг , уеб събиране или извличане на уеб данни е извличане на данни , използвано за извличане на данни от уебсайтове . Софтуерът за уеб скрапинг може да осъществява директен достъп до World Wide Web чрез протокола за прехвърляне на хипертекст или уеб браузър. Докато уеб скрапирането може да се извърши ръчно от потребител на софтуер, терминът обикновено се отнася до автоматизирани процеси, реализирани с помощта на бот или уеб робот . Това е форма на копиране, при която конкретни данни се събират и копират от мрежата, обикновено в централна локална база данни или електронна таблица, за по-късноизвличане или анализ .

Извличането на уеб страница включва нейното извличане и извличане от нея. Извличането е изтеглянето на страница (което браузърът прави, когато потребител разглежда страница). Следователно обхождането на мрежата е основен компонент на извличането на страници за последваща обработка. След като бъде извлечен, може да се извърши извличане. Съдържанието на дадена страница може да бъде анализирано , претърсено и преформатирано, а данните в нея – копирани в електронна таблица или заредени в база данни. Уеб скреперите обикновено изваждат нещо от страница, за да го използват за друга цел някъде другаде. Пример би бил намирането и копирането на имена и телефонни номера, компании и техните URL адреси или имейл адреси в списък (изтриване на контакти).

Освен извличането на контакти , извличането на уеб се използва като компонент на приложения, използвани за уеб индексиране , уеб копаене и извличане на данни , онлайн наблюдение на промените в цените и сравнение на цените , извличане на прегледи на продукти (за гледане на конкуренцията), събиране на обяви за недвижими имоти, време мониторинг на данни, откриване на промени в уебсайта , проучване, проследяване на онлайн присъствие и репутация, уеб смесване и интегриране на уеб данни .

Уеб страниците са изградени с помощта на текстови езици за маркиране ( HTML и XHTML ) и често съдържат множество полезни данни в текстова форма. Повечето уеб страници обаче са предназначени за крайни потребители , а не за лесна автоматизирана употреба. В резултат на това бяха разработени специализирани инструменти и софтуер за улесняване на изтриването на уеб страници.По-новите форми на уеб скрапинг включват наблюдение на емисии на данни от уеб сървъри.

Например JSON обикновено се използва като транспортен механизъм между клиента и уеб сървъра.

Има методи, които някои уебсайтове използват за предотвратяване на уеб скрапинг, като откриване и забрана на ботове да обхождат (преглеждат) техните страници. В отговор на това има системи за уеб скрапиране, които разчитат на използване на техники в анализиране на DOM , компютърно зрение и обработка на естествен език, за да симулират човешко сърфиране, за да позволят събиране на съдържание на уеб страница за офлайн анализиране

### Описание

Обикновено трансферът на данни между програмите се осъществява с помощта на структури от данни, подходящи за автоматизирана обработка от компютри, а не от хора. Такива формати и протоколи за обмен обикновено са строго структурирани, добре документирани, лесно анализирани и минимизират двусмислието. Много често тези предавания изобщо не са четими от човека.

По този начин ключовият елемент, който отличава извличането на данни от редовния разбор, е, че изходът, който се остъргва, е предназначен за показване на крайния потребител, а не като вход към друга програма. Следователно обикновено не е нито документирано, нито структурирано за удобен разбор. Извличането на данни често включва игнориране на двоични данни (обикновено изображения или мултимедийни данни), форматиране на дисплея, излишни етикети, излишни коментари и друга информация, която е или без значение, или възпрепятства автоматизираната обработка.

Извличането на данни най-често се извършва или за интерфейс към наследена система, която няма друг механизъм, съвместим с текущия хардуер, или за интерфейс към система на трета страна, която не осигурява по-удобен API. Във втория случай операторът на системата на третата страна често ще вижда извличането на екрана като нежелано поради причини като увеличено натоварване на системата, загуба на приходи от реклама или загуба на контрол върху информационното съдържание.

Извличането на данни обикновено се счита за ad hoc, неелегантна техника, често използвана само като "крайна мярка", когато няма друг механизъм за обмен на данни. Освен по-високите разходи за програмиране и обработка, изходните дисплеи, предназначени за консумация от човека, често променят структурата си. Хората могат лесно да се справят с това, но компютърната програма ще се провали. В зависимост от качеството и степента на логиката за обработка на грешки, присъстваща в компютъра, тази повреда може да доведе до съобщения за грешки, повреден изход или дори сривове на програмата.

### История

След раждането на World Wide Web през 1989 г., първият уеб робот, [2] World Wide Web Wanderer , беше създаден през юни 1993 г., който имаше за цел само да измерва размера на мрежата.

През декември 1993 г. беше пусната първата уеб търсачка, базирана на обхождане , JumpStation . Тъй като в мрежата имаше по-малко уебсайтове, търсачките по това време разчитаха на човешки администратори за събиране и форматиране на връзки. За сравнение, JumpStation беше първата WWW търсачка, която разчиташе на уеб робот.

През 2000 г. бяха създадени първият уеб API и API робот . API ( Интерфейс за програмиране на приложения) е интерфейс, който прави много по-лесно разработването на програма , като предоставя градивните елементи. През 2000 г. Salesforce и eBay пуснаха свой собствен API, с който програмистите можеха да имат достъп и да изтеглят някои от данните, достъпни за обществеността. Оттогава много уебсайтове предлагат уеб API за достъп на хората до тяхната публична база данни.

### Софтуер

Има много налични софтуерни инструменти, които могат да се използват за персонализиране на решения за уеб скрапинг. Този софтуер може да се опита автоматично да разпознае структурата на данните на дадена страница или да предостави интерфейс за запис, който премахва необходимостта от ръчно писане на код за уеб скрапиране, или някои скриптови функции, които могат да се използват за извличане и трансформиране на съдържание, и интерфейси на бази данни, които могат да съхраняват събраните данни в локални бази данни. Някои софтуери за уеб скрапинг също могат да се използват за директно извличане на данни от API.

### Уеб скрапинг

Уеб скрапингът е процес на автоматично извличане на данни или събиране на информация от световната мрежа. Това е поле с активно развитие, споделящо обща цел с визията на семантичната мрежа , амбициозна инициатива, която все още изисква пробиви в обработката на текст, семантичното разбиране, изкуствения интелект и взаимодействията човек-компютър .

Обикновено прехвърлянето на данни между програмите се осъществява с помощта на структури от данни , подходящи за автоматизирана обработка от компютри , а не от хора. Такива формати и протоколи за обмен обикновено са строго структурирани, добре документирани, лесно анализирани и минимизират неяснотата. Много често тези предавания изобщо не се четат от хора.

По този начин ключовият елемент, който отличава изчерпването на данни от редовното анализиране , е, че изходът, който се извлича, е предназначен за показване на краен потребител , а не като вход към друга програма. Поради това обикновено не е нито документиран, нито структуриран за удобно анализиране. Изтриването на данни често включва игнориране на двоични данни (обикновено изображения или мултимедийни данни), форматиране на дисплея , излишни етикети, излишни коментари и друга информация, която е или неуместна, или възпрепятства автоматизираната обработка.

Извличането на данни най-често се извършва или за интерфейс към наследена система , която няма друг механизъм, който да е съвместим с текущия хардуер , или за интерфейс към система на трета страна, която не предоставя по-удобен API . Във втория случай операторът на системата на трета страна често ще вижда остъргването на екрана като нежелано поради причини като повишено натоварване на системата , загуба на приходи от реклама или загуба на контрол върху информационното съдържание.

Извличането на данни обикновено се счита за ad hoc , неелегантна техника, често използвана само като „последна мярка“, когато не е наличен друг механизъм за обмен на данни. Освен по-високите разходи за програмиране и обработка, изходните дисплеи, предназначени за човешка консумация, често променят структурата си. Хората могат лесно да се справят с това, но компютърната програма ще се провали. В зависимост от качеството и степента на логиката за обработка на грешки, налична в компютъра, тази повреда може да доведе до съобщения за грешка, повреден изход или дори сривове на програмата.

* с DOM parsing и natural language processing, за да симулират поведение на човек при браузване на защитени по този начин страници.

### Технически варианти

Въпреки че използването на физически "тъп терминал" IBM 3270s бавно намалява, тъй като все повече и повече мейнфрейм приложения придобиват уеб интерфейси, някои уеб приложения просто продължават да използват техниката на изстъргване на екрана, за да заснемат стари екрани и да прехвърлят данните в съвременните предни краища.

Извличането на екрана обикновено се свързва с програмното събиране на визуални данни от източник, вместо да анализира данни, както при уеб извличането. Първоначално остъргването на екрана се отнасяше до практиката на четене на текстови данни от екрана на терминала на компютърния дисплей. Това обикновено се прави чрез четене на паметта на терминала през спомагателния порт или чрез свързване на терминалния изходен порт на една компютърна система към входен порт на друг. Терминът екранно изстъргване също често се използва за обозначаване на двупосочния обмен на данни. Това може да са простите случаи, при които контролиращата програма навигира през потребителския интерфейс или по-сложни сценарии, при които контролиращата програма въвежда данни в интерфейс, предназначен да бъде използван от човек.

Като конкретен пример за класически екранен скрепер, помислете за хипотетична наследена система, датираща от 1960-те години на миналия век – зората на компютъризираната обработка на данни. Компютърните към потребителските интерфейси от тази епоха често са просто текстови тъпи терминали, които не са много повече от виртуални телепринтери (такива системи все още се използват днес)., по различни причини). Желанието да се свърже такава система с по-модерни системи е често срещано. Стабилното решение често изисква неща, които вече не са налични, като изходен код, системна документация, API или програмисти с опит в 50-годишна компютърна система. В такива случаи единственото възможно решение може да бъде да се напише скрепер на екрана, който "се преструва" на потребител на терминал. Скреперът на екрана може да се свърже с наследената система чрез Telnet, да подражава на натисканията на клавиши, необходими за навигация в стария потребителски интерфейс, да обработва получения изход на дисплея, да извлича желаните данни и да ги предава на съвременната система. Сложно и устойчиво внедряване от този вид, изградено върху платформа, осигуряваща управлението и контрола, изисквани от голямо предприятие – например контрол на промените, сигурност, управление на потребителите, защита на данните, оперативен одит, балансиране на натоварването и управление на опашки и т.н. – може да се каже, че е пример за роботизиран софтуер за автоматизация на процесите, наречен RPA или RPAAI за самоуправляващ се RPA 2.0, базиран на изкуствен интелект.

През 1980-те години на миналия век доставчици на финансови данни като Reuters, Telerate и Quotron показват данни във формат 24×80, предназначени за човешки читател. Потребителите на тези данни, особено инвестиционните банки, са написали заявления за улавяне и преобразуване на тези символни данни като числови данни за включване в изчисления за търговски решения, без повторно въвеждане на данните. Общият термин за тази практика, особено в Обединеното кралство, е раздробяване на страници, тъй като резултатите могат да си представят, че са преминали през шредер за хартия. Вътрешно Ройтерс използва термина "логически" за този процес на преобразуване, изпълнявайки сложна компютърна система на VAX / VMS, наречена Logicizer.

По-модерните техники за извличане на екрана включват заснемане на растерните данни от екрана и пускането им през OCR двигател или за някои специализирани автоматизирани системи за тестване, като се съпоставят растерните данни на екрана с очакваните резултати. Това може да се комбинира в случай на GUI приложения, с търсене на графични контроли чрез програмно получаване на препратки към техните основни програмни обекти. Последователност от екрани автоматично се заснема и конвертира в база данни.

Друга съвременна адаптация към тези техники е да се използва, вместо последователност от екрани като вход, набор от изображения или PDF файлове, така че има някои припокривания с общи "извличане на документи" и техники за извличане на отчети.

### Извличане на уеб

Уеб страниците са изградени с помощта на текстови езици за маркиране (HTML и XHTML) и често съдържат множество полезни данни в текстова форма. Повечето уеб страници обаче са предназначени за човешки крайни потребители, а не за лесна автоматизирана употреба. Поради това бяха създадени комплекти инструменти, които изстъргват уеб съдържание. Уеб скрепер е API или инструмент за извличане на данни от уебсайт. Компании като Amazon AWS и Google предоставят инструменти, услуги и публични данни, достъпни безплатно за крайните потребители. По-новите форми на извличане на уеб включват слушане на потоци от данни от уеб сървъри. Например, JSON обикновено се използва като механизъм за съхранение на транспорт между клиента и уеб сървъра.

Напоследък компаниите са разработили системи за уеб остъргване, които разчитат на използването на техники в DOM разбор, компютърно зрение и обработка на естествен език, за да симулират човешката обработка, която се случва при гледане на уеб страница, за да извлекат автоматично полезна информация.

### Отчет за добив

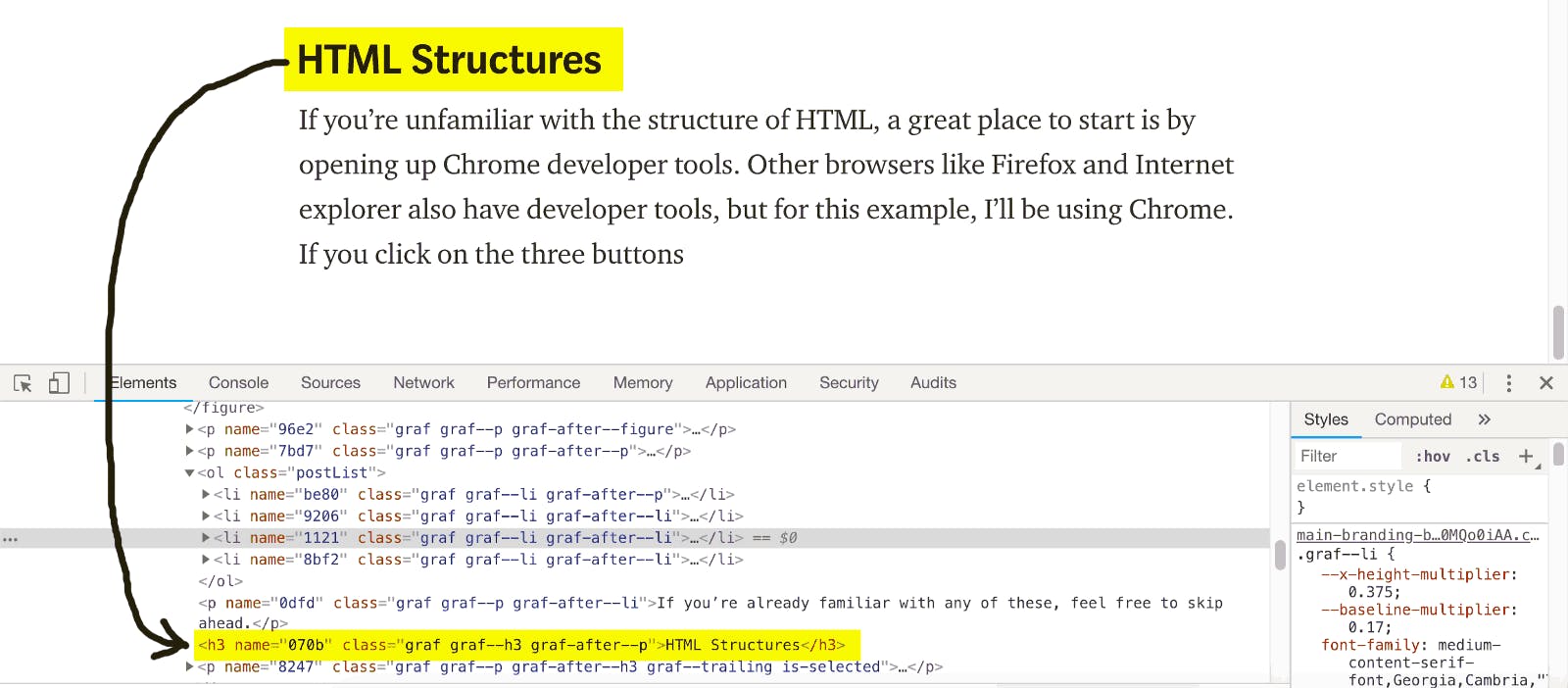
Извличането на доклади е извличането на данни от компютърни доклади, които могат да се четат от човека. Конвенционалното извличане на данни изисква връзка със система с работещ източник, подходящи стандарти за свързване или API и обикновено сложно запитване. Чрез използване на стандартните опции за отчитане на изходната система и насочване на изхода към файл на макара, вместо към принтер, могат да се генерират статични отчети, подходящи за офлайн анализ чрез извличане на отчети. [9] Този подход може да избегне интензивното използване на процесора в работно време, може да сведе до минимум разходите за лицензиране на крайни потребители за ERP клиенти и може да предложи много бързо прототипиране и разработване на персонализирани отчети. Докато извличането на данни и извличането на уеб включват взаимодействие с динамичен изход, извличането на отчети включва извличане на данни от файлове в четим от човека формат, като HTML, PDF или текст. Те могат лесно да бъдат генерирани от почти всяка система чрез прихващане на подаването на данни към принтер. Този подход може да осигури бърз и лесен маршрут за получаване на данни, без да е необходимо да се програмира API към изходната система.

# Аргументиран избор на технология за реализиране на парсинг

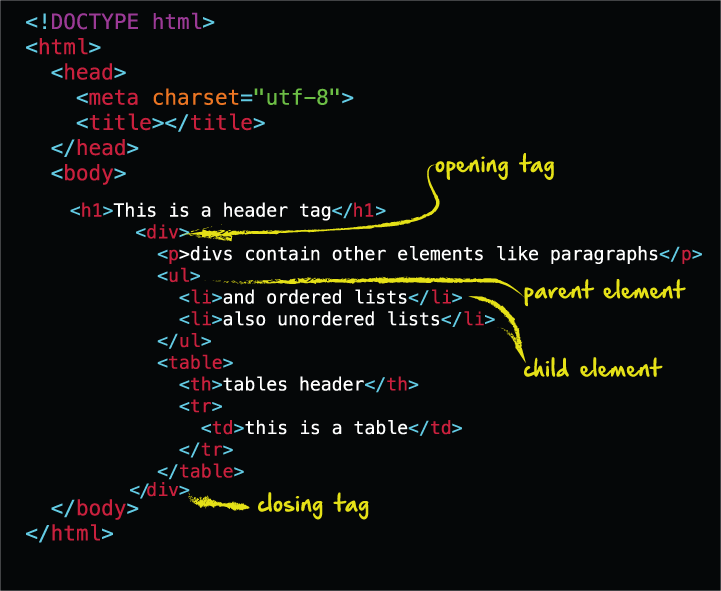
## Изграждане на уеб скрепер:

### HTML структури

* Идентифициране на HTML тагове

Идентифициране на HTML тагове Ако не сте запознати със структурата на HTML, чудесно място да започнете е като отворите инструментите за разработчици на Chrome. Други браузъри като Firefox и Internet Explorer също имат инструменти за разработчици, но за този пример ще използвам Chrome. Ако кликнете върху трите вертикални точки в горния десен ъгъл на браузъра, а след това върху опцията "Още инструменти" и след това върху "Инструменти за разработчици", ще видите панел, който се появява, който изглежда по следния начин:

Можем бързо да видим как е структуриран текущият HTML сайт. Цялото съдържание, съдържащо се в конкретни "тагове". Текущото заглавие е в таг "<h3>", докато повечето от параграфите са в тагове "<p>". Всеки от таговете има и други атрибути като "клас" или "име". Не е нужно да знаем как да изградим HTML сайт от нулата. При изграждането на уеб стъргалка трябва само да знаем основната структура на мрежата и как да идентифицираме конкретни уеб елементи. Chrome и други инструменти за разработчици на браузъри ни позволяват да видим какви маркери съдържат информацията, която искаме да изстържем, както и други атрибути като "клас", които могат да ни помогнат да изберем само конкретни елементи.

Нека да разгледаме как може да изглежда типичната HTML структура:

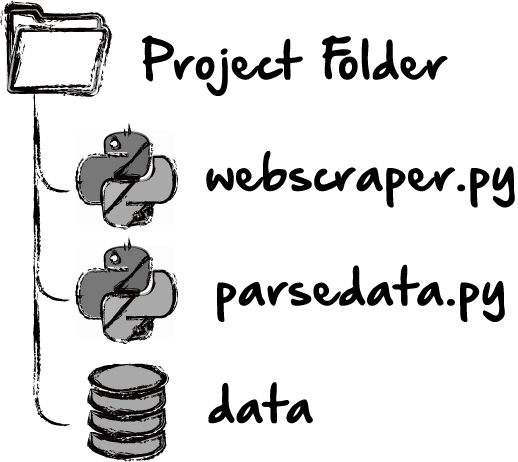
Това е подобно на това, което току-що

разгледахме в инструментите за разработка на хром. Тук можем да видим, че всички елементи в HTML се съдържат в таговете за отваряне и затваряне на "тялото". Всеки елемент има и собствен етикет за отваряне и затваряне. Елементите, които са вложени или с отстъп в HTML структура, показват, че елементът е дъщерен елемент на контейнера или родителския елемент. След като започнем да правим нашия уеб скрепер Python, можем също да идентифицираме елементи, които искаме да изстържем не само въз основа на името на етикета, но и дали елементът е дете на друг елемент. Например, тук можем да видим, че в тази структура има таг <ul>, показващ неподреден списък. Всеки елемент от списъка <li> е наследник на родителския таг <ul>. Ако искаме да изберем и изстържем целия списък, може да искаме да кажем на Python да вземе всички дъщерни елементи на тага <ul>.

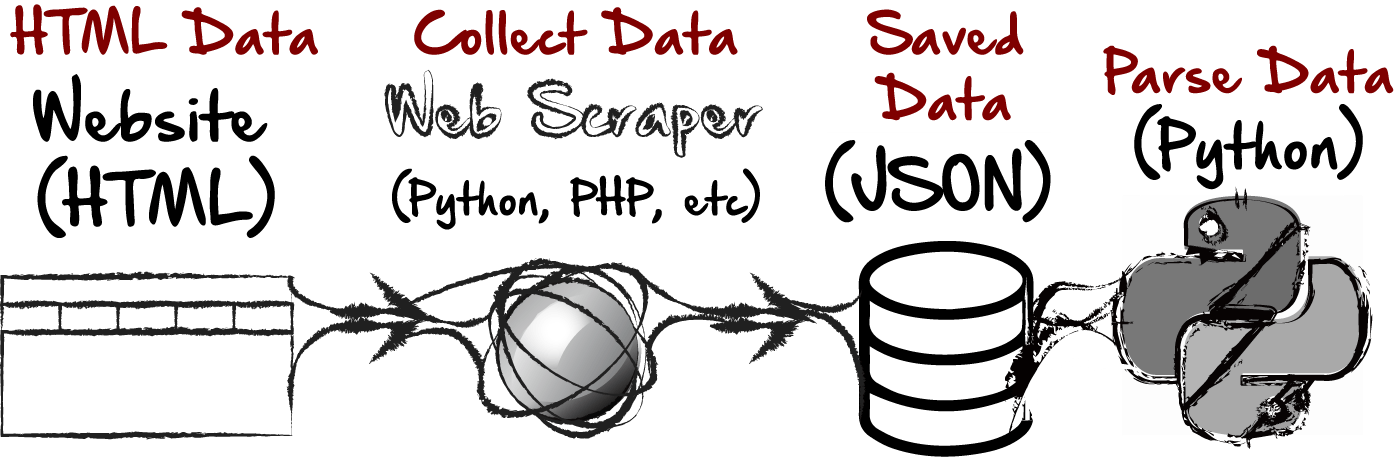
### Основи на Python

* Създаване на нов проект

Ще трябва да вземем решение и за текстов редактор. Използвам ATOM, но има много други подобни възможности, които правят относително едно и също нещо. Тъй като уеб скреперите са доста ясни, нашият избор кой текстов редактор да използваме зависи изцяло от нас. След като вече имаме инсталиран Python и използваме текстов редактор по наш избор, нека създадем нова папка на проекта на Python. Първо отидете до мястото, където искаме да създадем този проект. Предпочитам да хвърля всичко върху моя вече претрупан работен плот. След това създайте нова папка и вътре в папката създайте файл. Ще кръстим този файл „webscraper.py“. Също така ще искаме да направим втори файл, наречен “parsedata.py” в същата папка. В този момент трябва да имаме нещо подобно на това:



Една очевидна разлика е, че все още нямаме никакви данни. Данните ще бъдат тези, които сме извлекли от мрежата. Ако помислим какъв може да бъде нашият работен процес за този проект, можем да си представим, че изглежда по следния начин:



Първо, има необработени HTML данни, които са там в мрежата. След това използваме програма, която създаваме в Python, за да изтрием/съберем данните, които искаме. След това данните се съхраняват във формат, който можем да използваме. И накрая, можем да анализираме данните, за да намерим подходяща информация. Изтриването и анализирането ще се обработват от отделни скриптове на Python. Първият ще събира данните. Вторият ще анализира данните, които сме събрали.

### Виртуални среди на Python

Все още не сме приключили с настройката на проекта. В Python често ще използваме библиотеки като част от нашия проект. Библиотеките са като пакети, които съдържат допълнителна функционалност за нашия проект. В нашия случай ще използваме две библиотеки: Beautiful Soup и Requests. Библиотеката за заявки ни позволява да правим заявки към URL адреси и да осъществяваме достъп до данните на тези HTML страници. Beautiful Soup съдържа някои лесни начини да идентифицираме таговете, които обсъдихме по-рано, направо от нашия скрипт на Python.

Ако инсталираме тези пакети глобално на нашите машини, може да се сблъскаме с проблеми, ако продължим да разработваме други приложения. Например, една програма може да използва библиотеката с заявки, версия 1, докато по-късно приложение може да използва библиотеката с заявки, версия 2. Това може да причини конфликт, което да затрудни изпълнението на едно или и на двете приложения.

За да разрешите този проблем, добра идея е да създадете виртуална среда. Тези виртуални среди са като капсули за приложението. По този начин можем да стартираме версия 1 на библиотека в едно приложение и версия 2 в друго, без конфликт, ако създадем виртуална среда за всяко приложение.

Първо нека изведем прозореца на терминала, тъй като следващите няколко команди са най-лесни за изпълнение от терминала. В OS X ще отворим папката Applications, след което ще отворим папката Utilities. Отворете приложението Terminal. Може да искаме да добавим и това към нашия док.

В Windows можем също да намерим терминал/команден ред, като отворим нашето меню "Старт" и потърсим. Това е просто приложение, намиращо се в C:\Windows\System32.

Сега, когато терминалът е отворен, трябва да отидем до нашата папка на проекта и да използваме следната команда, за да изградим виртуалната среда:

python3 -m venv урок-env

Тази стъпка създава виртуалната среда, но в момента тя е просто латентна. За да използваме виртуалната среда, ще трябва също да я активираме. Можем да направим това, като изпълним следната команда в нашия терминал:

tutorial-env\Scripts\activate.bat

### Библиотеки на Python

* Инсталиране на библиотеки

Сега, след като нашата виртуална среда е настроена и активирана, ще искаме да инсталираме библиотеките, които споменахме по-рано. За да направим това, ще използваме терминала отново, като този път ще инсталираме библиотеките с инсталатора на pip. Нека изпълним следните команди:

Инсталирайте Beautiful Soup:

pip инсталирайте bs4

Заявки за инсталиране:

заявки за инсталиране на pip

И сме готови. Е, поне имаме нашата среда и библиотеки работещи.

### Импортиране на инсталирани библиотеки

Първо, нека отворим нашия файл webscraper.py. Тук ще настроим цялата логика, която действително ще изисква данните от сайта, който искаме да изчерпим.

Първото нещо, което ще трябва да направим, е да уведомим Python, че всъщност ще използваме библиотеките, които току-що инсталирахме. Можем да направим това, като ги импортираме в нашия Python файл. Може би е добра идея да структурираме нашия файл, така че цялото ни импортиране да е в горната част на файла, а след това цялата ни логика да идва след това. За да импортираме и двете си библиотеки, просто ще включим следните редове в горната част на нашия файл:

от bs4 импортиране на BeautifulSoupimport заявки

след това да

Ако искахме да инсталираме други библиотеки към този проект, бихме могли да го направим чрез инсталатора на pip и ги импортираме в горната част на нашия файл. Едно нещо, което трябва да знаете е, че някои библиотеки са доста големи и могат да заемат много място. Може да е трудно да внедрим сайт, върху който сме работили, ако е претрупан с твърде много големи пакети.

### Библиотека за заявки на Python

Заявките с Python и Beautiful Soup основно ще имат три части:

* URL

URL адресът е просто низ, който съдържа адреса на HTML страницата, която възнамеряваме да изтрием.

* Response

Response е резултат от GET заявка. Всъщност ще използваме URL променливата в GET заявката тук. Ако погледнем какъв е отговорът, това всъщност е HTTP код на състоянието. Ако заявката е била успешна, ще получим успешен код за състояние като 200. Ако е имало проблем със заявката или сървърът не отговори на заявката, която направихме, кодът за състояние може да е неуспешен. Ако не получим това, което искаме, можем да потърсим кода на състоянието, за да отстраним каква може да е грешката. Ето един полезен ресурс за намиране на значението на кодовете, в случай че трябва да ги отстраним:

##### Content

Content е съдържанието на отговора. Ако отпечатаме цялото съдържание на отговора, ще получим цялото съдържание на цялата страница на URL адреса, който сме поискали.

## Beautiful Soup

Beautiful Soup е библиотека на Python за извличане на данни от HTML и XML файлове. Той работи с любимия ви анализатор, за да предостави идиоматични начини за навигация, търсене и модифициране на дървото за анализ. Обикновено спестява на програмистите часове или дни работа.

##### Видове обекти

Beautiful Soup трансформира сложен HTML документ в сложно дърво от обекти на Python. Но ще трябва да се справяте само с около четири вида обекти: Tag, NavigableString, BeautifulSoupи Comment.

##### Етикет

Обектът Tagсъответства на XML или HTML таг в оригиналния документ.

soup = BeautifulSoup('<b class="boldest">Extremely bold</b>', 'html.parser')

tag = soup.b

type(tag)

# <class 'bs4.element.Tag'>

##### Име

Всеки таг има име:

tag.name

# 'b'

Ако промените името на маркер, промяната ще бъде отразена във всяко маркиране, генерирано от Beautiful Soup надолу по линията:

tag.name = "blockquote"

tag

# <blockquote class="boldest">Extremely bold</blockquote>

##### Атрибути

Един HTML или XML таг може да има произволен брой атрибути. Тагът има атрибут “id”, чиято стойност е “boldest”. Можете да получите достъп до атрибутите на етикет, като третирате етикета като речник:<b id="boldest">

tag = BeautifulSoup('<b id="boldest">bold</b>', 'html.parser').b

tag['id']

# 'boldest'

Можете да получите директен достъп до речника на атрибутите като .attrs:

tag.attrs

# {'id': 'boldest'}

Можете да добавяте, премахвате и модифицирате атрибути на етикет. Отново, това се прави чрез третиране на етикета като речник:

tag['id'] = 'verybold'

tag['another-attribute'] = 1

tag

# <b another-attribute="1" id="verybold"></b>

del tag['id']

del tag['another-attribute']

tag

# <b>bold</b>

tag['id']

# KeyError: 'id'

tag.get('id')

# None

### Атрибути с множество стойности

HTML 4 дефинира няколко атрибута, които могат да имат множество стойности. HTML 5 премахва няколко от тях, но дефинира още няколко. Най-често срещаният атрибут с множество стойности е class(т.е. един таг може да има повече от един CSS клас). Други включват rel, rev, accept-charset, headersи accesskey. По подразбиране Beautiful Soup анализира стойността(ите) на многозначен атрибут в списък:

css\_soup = BeautifulSoup('<p class="body"></p>', 'html.parser')

css\_soup.p['class']

# ['body']

css\_soup = BeautifulSoup('<p class="body strikeout"></p>', 'html.parser')

css\_soup.p['class']

# ['body', 'strikeout']

Ако даден атрибут изглежда , че има повече от една стойност, но не е атрибут с много стойности, както е дефинирано от която и да е версия на стандарта HTML, Beautiful Soup ще остави атрибута сам:

id\_soup = BeautifulSoup('<p id="my id"></p>', 'html.parser')

id\_soup.p['id']

# 'my id'

Когато превърнете етикет обратно в низ, множество стойности на атрибути се консолидират

rel\_soup = BeautifulSoup('<p>Back to the <a rel="index first">homepage</a></p>', 'html.parser')

rel\_soup.a['rel']

# ['index', 'first']

rel\_soup.a['rel'] = ['index', 'contents']

print(rel\_soup.p)

# <p>Back to the <a rel="index contents">homepage</a></p>

Можете да принудите всички атрибути да бъдат анализирани като низове, като подадете multi\_valued\_attributes=Noneкато аргумент на ключова дума в BeautifulSoupконструктора:

no\_list\_soup = BeautifulSoup('<p class="body strikeout"></p>', 'html.parser', multi\_valued\_attributes=None)

no\_list\_soup.p['class']

# 'body strikeout'

Можете да използвате get\_attribute\_list, за да получите стойност, която винаги е списък, независимо дали е атрибут с много стойности или не:

id\_soup.p.get\_attribute\_list('id')

# ["my id"]

Ако анализирате документ като XML, няма многостойностни атрибути:

xml\_soup = BeautifulSoup('<p class="body strikeout"></p>', 'xml')

xml\_soup.p['class']

# 'body strikeout'

Отново можете да конфигурирате това с помощта на multi\_valued\_attributesаргумента:

class\_is\_multi= { '\*' : 'class'}

xml\_soup = BeautifulSoup('<p class="body strikeout"></p>', 'xml', multi\_valued\_attributes=class\_is\_multi)

xml\_soup.p['class']

# ['body', 'strikeout']

Вероятно няма да е необходимо да правите това, но ако го направите, използвайте настройките по подразбиране като ръководство. Те прилагат правилата, описани в HTML спецификацията:

from bs4.builder import builder\_registry

builder\_registry.lookup('html').DEFAULT\_CDATA\_LIST\_ATTRIBUTES

### Специални низове

Tag, NavigableString, и BeautifulSoupобхващат почти всичко, което ще видите в HTML или XML файл, но има няколко остатъчни бита. Основният, който вероятно ще срещнете, е Comment.

##### Comment

markup = "<b><!--Hey, buddy. Want to buy a used parser?--></b>"

soup = BeautifulSoup(markup, 'html.parser')

comment = soup.b.string

type(comment)

# <class 'bs4.element.Comment'>

Обектът Commentе просто специален тип на NavigableString:

comment

# 'Hey, buddy. Want to buy a used parser'

Но когато се появява като част от HTML документ, a Commentсе показва със специално форматиране:

print(soup.b.prettify())

# <b>

# <!--Hey, buddy. Want to buy a used parser?-->

# </b>

### HTML документи

Beautiful Soup дефинира няколко NavigableStringподкласа, които да съдържат низове, намерени в конкретни HTML тагове. Това улеснява избирането на основното тяло на страницата, като игнорира низове, които вероятно представляват програмни директиви, намерени в страницата. (Тези класове са нови в Beautiful Soup 4.9.0 и парсерът html5lib не ги използва.)

##### Стилове

Подклас NavigableString, който представлява вградени CSS таблици със стилове; т.е. всички низове, намерени вътре в <style>таг по време на анализ на документ.

##### Скрипт

Подклас NavigableString, който представлява вграден Javascript; т.е. всички низове, намерени вътре в <script>таг по време на анализ на документ.

##### Шаблон

Подклас NavigableString, който представлява вградени HTML шаблони; т.е. всички низове, намерени вътре в <template>таг по време на анализ на документ.

### Requests

Библиотеката с заявки предоставя прост API за взаимодействие с HTTP операции като GET, POSTи т.н.Методите, внедрени в библиотеката с заявки, изпълняват HTTP операции срещу конкретен уеб сървър, посочен от неговия URL адрес.Той също така поддържа изпращане на допълнителна информация към уеб сървър чрез параметри и заглавки, кодиране на отговорите на сървъра, откриване на грешки и обработка на пренасочвания.В допълнение към опростяването на начина, по който работим с HTTP операциите, библиотеката с заявки предоставя някои разширени функции, като обработка на HTTP изключения и удостоверяване.

## HTTP

Протоколът за трансфер на хипертекст (HTTP) е протокол за заявка/отговор, базиран на архитектурата клиент-сървър, който разчита на TCP/IP връзки за обмен на съобщения за заявка и отговор.HTTP клиенти като уеб браузъри или мобилни приложения изпращат заявки до HTTP сървър и сървърът им отговаря със съобщения, съдържащи ред за състояние, заглавка и тяло.

Протоколът за трансфер на хипертекст ( HTTP ) е протокол на приложния слой в модела на набора от интернет протоколи за разпределени, съвместни, хипермедийни информационни системи. HTTP е основата на комуникацията на данни за World Wide Web , където хипертекстовите документи включват хипервръзки към други ресурси, до които потребителят може лесно да получи достъп, например чрез щракване на мишката или чрез докосване на екрана в уеб браузър.

Разработването на HTTP беше инициирано от Тим ​​Бърнърс-Лий в CERN през 1989 г. и обобщено в прост документ, описващ поведението на клиент и сървър, използващи първата версия на HTTP протокола, наречена 0.9.

Тази първа версия на HTTP протокола скоро се превърна в по-разработена версия, която беше първата чернова към далечна бъдеща версия 1.0.

Разработването на ранни HTTP заявки за коментари (RFC) започна няколко години по-късно и беше координирано усилие от Internet Engineering Task Force (IETF) и World Wide Web Consortium (W3C), като работата по-късно се премести в IETF.

HTTP/1 беше финализиран и напълно документиран (като версия 1.0) през 1996 г. Той се разви (като версия 1.1) през 1997 г. и след това спецификациите му бяха актуализирани през 1999 г., 2014 г. и 2022 г. Негови HTTP/2 , публикуван през 2015 г., предоставя по-ефективен израз на семантиката на HTTP „по кабела“. Към април 2023 г. той се използва от 39% от уебсайтовете и се поддържа от почти всички уеб браузъри (над 97% от потребителите).Освен това се поддържа от големи уеб сървъри чрез защита на транспортния слой (TLS), използвайки разширение за договаряне на протокола на приложния слой (ALPN) , където се изисква TLS 1.2 или по-нова версия.

HTTP/3 , наследникът на HTTP/2, беше публикуван през 2022 г. Сега се използва от над 25% от уебсайтовете и се поддържа от много уеб браузъри (над 75% от потребителите). HTTP/3 използва QUIC вместо TCP за основния транспортен протокол. Подобно на HTTP/2, той не остарява предишните основни версии на протокола. Поддръжката за HTTP/3 беше добавена първо към Cloudflare и Google Chrome , и също е активирана във Firefox .HTTP/3 има по-ниска латентност за уеб страници от реалния свят, ако е активиран на сървъра, зарежда по-бързо от HTTP/2 и дори по-бързо от HTTP/1.1, в някои случаи над 3 пъти по-бързо от HTTP/1.1 (което все още е обикновено само активиран). Неговият защитен вариант, наречен HTTPS , се използва от повече от 80% от уебсайтовете.

HTTP дефинира методи (понякога наричани глаголи , но никъде в спецификацията не споменава глагол ), за да посочи желаното действие, което да бъде извършено върху идентифицирания ресурс. Какво представлява този ресурс, независимо дали вече съществуващи данни или данни, които се генерират динамично, зависи от внедряването на сървъра. Често ресурсът съответства на файл или резултат от изпълним файл, който се намира на сървъра. Спецификацията HTTP/1.0 дефинира методите GET, HEAD и POST, а спецификацията HTTP/1.1 добави пет нови метода: PUT, DELETE, CONNECT, OPTIONS и TRACE. Всеки клиент може да използва всеки метод и сървърът може да бъде конфигуриран да поддържа всяка комбинация от методи. Ако даден метод е непознат на междинния продукт, той ще се третира като несигурен и неидемпотентен метод. Няма ограничение за броя на методите, които могат да бъдат дефинирани, което позволява бъдещите методи да бъдат специфицирани, без да се нарушава съществуващата инфраструктура. Например WebDAV дефинира седем нови метода, а RFC 5789 посочи метода PATCH .

Имената на методите са чувствителни към регистъра.Това е за разлика от имената на полетата на HTTP заглавките, които не са чувствителни към главни и малки букви.

### Методи:

* ВЗЕМЕТЕ

Методът GET изисква целевият ресурс да прехвърли представяне на своето състояние. Заявките GET трябва само да извличат данни и не трябва да имат друг ефект. (Това важи и за някои други HTTP методи.)За извличане на ресурси без извършване на промени, GET е за предпочитане пред POST, тъй като те могат да бъдат адресирани чрез URL . Това позволява маркиране и споделяне и прави GET отговорите подходящи за кеширане , което може да спести честотна лента. W3C публикува насоки за това разграничение, като казва: „ Дизайнът на уеб приложенията трябва да бъде информиран от горните принципи, но също и от съответните ограничения.“Вижбезопасни методи по-долу.

* ГЛАВА

Методът HEAD изисква целевият ресурс да прехвърли представяне на своето състояние, както при GET заявка, но без данните за представяне, включени в тялото на отговора. Това е полезно за извличане на метаданните за представяне в заглавката на отговора, без да се налага да прехвърляте цялото представяне. Употребите включват проверка дали дадена страница е достъпна чрез кода на състоянието и бързо намиране на размера на файл ( Content-Length).

* ПУБЛИКУВАНЕ

Методът POST изисква целевият ресурс да обработи представянето, включено в заявката, според семантиката на целевия ресурс. Например, използва се за публикуване на съобщение в интернет форум , абониране за пощенски списък или завършване на транзакция за онлайн пазаруване .

* СЛАГАМ

Методът PUT изисква целевият ресурс да създаде или актуализира своето състояние със състоянието, дефинирано от представянето, включено в заявката. Разлика от POST е, че клиентът посочва целевото местоположение на сървъра.

* ИЗТРИЙ

Методът DELETE изисква целевият ресурс да изтрие своето състояние.

* СВЪРЗВАНЕ

Методът CONNECT изисква посредникът да установи TCP/IP тунел към първоначалния сървър, идентифициран от целта на заявката. Често се използва за защита на връзки през един или повече HTTP проксита с TLS . Вижте метод HTTP CONNECT .

* НАСТРОИКИ

Методът OPTIONS изисква целевият ресурс да прехвърли HTTP методите, които поддържа. Това може да се използва за проверка на функционалността на уеб сървър чрез заявка на '\*' вместо конкретен ресурс.

* СЛЕДИ

Методът TRACE изисква целевият ресурс да прехвърли получената заявка в тялото на отговора. По този начин клиентът може да види какви (ако има такива) промени или допълнения са направени от посредниците.

* КРЕПКА

Методът PATCH изисква целевият ресурс да промени състоянието си според частичната актуализация, дефинирана в представянето, приложено в заявката. Това може да спести честотна лента чрез актуализиране на част от файл или документ, без да се налага да го прехвърляте изцяло.

Всички уеб сървъри с общо предназначение трябва да прилагат поне методите GET и HEAD, а всички други методи се считат за незадължителни от спецификацията.

##### Безопасни методи

Метод на заявка е безопасен , ако заявка с този метод няма предвиден ефект върху сървъра. Методите GET, HEAD, OPTIONS и TRACE се определят като безопасни. С други думи, безопасните методи са предназначени да бъдат само за четене . Те обаче не изключват странични ефекти , като например добавяне на информация за заявка към лог файл или таксуване на рекламен акаунт , тъй като те не се изискват от клиента по дефиниция.

Обратно, методите POST, PUT, DELETE, CONNECT и PATCH не са безопасни. Те могат да променят състоянието на сървъра или да имат други ефекти като изпращане на имейл . Следователно такива методи обикновено не се използват от съобразени уеб роботи или уеб роботи; някои, които не отговарят, са склонни да правят искания без оглед на контекста или последствията.

Въпреки предписаната безопасност на GET заявките, на практика тяхната обработка от сървъра не е технически ограничена по никакъв начин. Небрежното или умишлено нередовно програмиране може да позволи GET заявките да причинят нетривиални промени на сървъра. Това не се препоръчва поради проблемите, които могат да възникнат, когато уеб кеширането , търсачките и други автоматизирани агенти правят нежелани промени на сървъра. Например, уебсайт може да позволи изтриване на ресурс чрез URL адрес като https://example.com/article/1234/delete , който, ако бъде извлечен произволно, дори с помощта на GET, просто ще изтрие статията.Правилно кодиран уебсайт би изисквал метод DELETE или POST за това действие, което незлонамерените ботове не биха направили.

Един пример за това, което се случва на практика, беше по време на краткотрайната бета версия на Google Web Accelerator , която предварително извличаше произволни URL адреси на страницата, която потребителят преглеждаше, което караше записите да бъдат автоматично променяни или масово изтривани . Бета бе спряна само седмици след първото си пускане, след широко разпространена критика.

##### Идемпотентни методи

Методът на заявка е идемпотентенако множество идентични заявки с този метод имат същия ефект като една такава заявка. Методите PUT и DELETE и безопасните методи се определят като идемпотентни. Безопасните методи са тривиално идемпотентни, тъй като са предназначени да нямат никакъв ефект върху сървъра; методите PUT и DELETE междувременно са идемпотентни, тъй като последователните идентични заявки ще бъдат игнорирани. Един уебсайт може например да настрои крайна точка PUT, за да промени записания имейл адрес на потребителя. Ако тази крайна точка е конфигурирана правилно, всички заявки, които изискват промяна на имейл адреса на потребител със същия имейл адрес, който вече е записан – например дублирани заявки след успешна заявка – няма да имат ефект. По същия начин, искане за ИЗТРИВАНЕ на определен потребител няма да има ефект, ако този потребител вече е бил изтрит.

Обратно, методите POST, CONNECT и PATCH не са непременно идемпотентни и следователно изпращането на идентична POST заявка многократно може допълнително да промени състоянието на сървъра или да има допълнителни ефекти, като например изпращане на множество имейли . В някои случаи това е желаният ефект, но в други случаи може да се получи случайно. Потребителят може, например, по невнимание да изпрати множество POST заявки, като щракне отново върху бутон, ако не получи ясна обратна връзка, че първото щракване се обработва. Въпреки че уеб браузърите могат да показват предупредителни диалогови прозорци , за да предупредят потребителите в някои случаи, когато презареждането на страница може да изпрати повторно POST заявка, обикновено уеб приложението трябва да обработва случаите, когато POST заявка не трябва да се изпраща повече от веднъж.

Обърнете внимание, че дали даден метод е идемпотентен или не, не се налага от протокола или уеб сървъра. Напълно възможно е да напишете уеб приложение, в което (например) вмъкване на база данни или друго неидемпотентно действие се задейства от GET или друга заявка. Да се ​​направи това срещу препоръките обаче може да доведе до нежелани последствия, ако потребителски агент приеме, че повтарянето на същата заявка е безопасно, когато не е така.

##### Кешируеми методи

Метод на заявка може да се кешира , ако отговорите на заявки с този метод могат да се съхраняват за бъдеща повторна употреба. Методите GET, HEAD и POST са определени като кеширащи.

Обратно, методите PUT, DELETE, CONNECT, OPTIONS, TRACE и PATCH не могат да се кешират.

### Използване на GET заявка

Използваме getметода, за да поискаме данни от конкретен уеб сървър. Нека да го изпробваме:

url = 'http://httpbin.org/json'

r = requests.get(url)

print('Response Code:', r.status\_code)

print('Response Headers:\n', r.headers)

print('Response Content:\n',r.text)

Изпълнението на кода по-горе извежда код на състояние на 200, което показва, че URL адресът е достъпен. След това връща данните за заглавката на страницата, последвани от съдържанието на страницата.

Свойството headersвръща специален речник, направен само за HTTP заглавки, така че можете да получите достъп до всеки елемент просто като използвате неговия ключ:

print(r.headers['Content-Type'])

В последния ред на кода можем да използваме свойството content, което връща съдържанието на страницата като поредица от байтове, но предпочитаме да използваме свойството, textкоето отпечатва съдържанието на страницата като декодиран текст във формат Unicode.

### Използване на GET параметри

Ние използваме параметрите GET, за да предадем информация във формат на двойка ключ-стойност към уеб сървър чрез URL адрес. Методът getни позволява да предадем речник от двойки ключ-стойност, използвайки аргумента params. Да пробваме.

url = 'http://httpbin.org/get'

payload = {

'website':'dataquest.io',

'courses':['Python','SQL']

}

r = requests.get(url, params=payload)

print('Response Content:\n',r.text)

Стартирайте горния код. Ще видите следния резултат:

Response Content:

{

"args": {

"courses": [

"Python",

"SQL"

],

"website": "dataquest.io"

},

"headers": {

"Accept": "\*/\*",

"Accept-Encoding": "gzip, deflate",

"Host": "httpbin.org",

"User-Agent": "python-requests/2.27.1",

"X-Amzn-Trace-Id": "Root=1-61e7e066-5d0cacfb49c3c1c3465bbfb2"

},

"origin": "121.122.65.155",

"url": "http://httpbin.org/get?website=dataquest.io&courses=Python&courses=SQL"

}

Съдържанието на отговора е във формат JSON и двойките ключ-стойност, които сме предали през аргумента, paramsсе появяват в argsраздела на отговора. Освен това urlразделът съдържа кодирания URL адрес заедно с параметрите, предадени на сървъра.

# Приложение за парсинг на сайтове

Python като език за разработка на парсинг python

**Python** е език за програмиране [на високо ниво](https://en.wikipedia.org/wiki/High-level_programming_language) с [общо предназначение](https://en.wikipedia.org/wiki/General-purpose_programming_language) . Неговата философия на дизайна набляга на [четливостта на кода](https://en.wikipedia.org/wiki/Code_readability) с използването на значителни отстъпи чрез [правилото извън страната](https://en.wikipedia.org/wiki/Off-side_rule) .

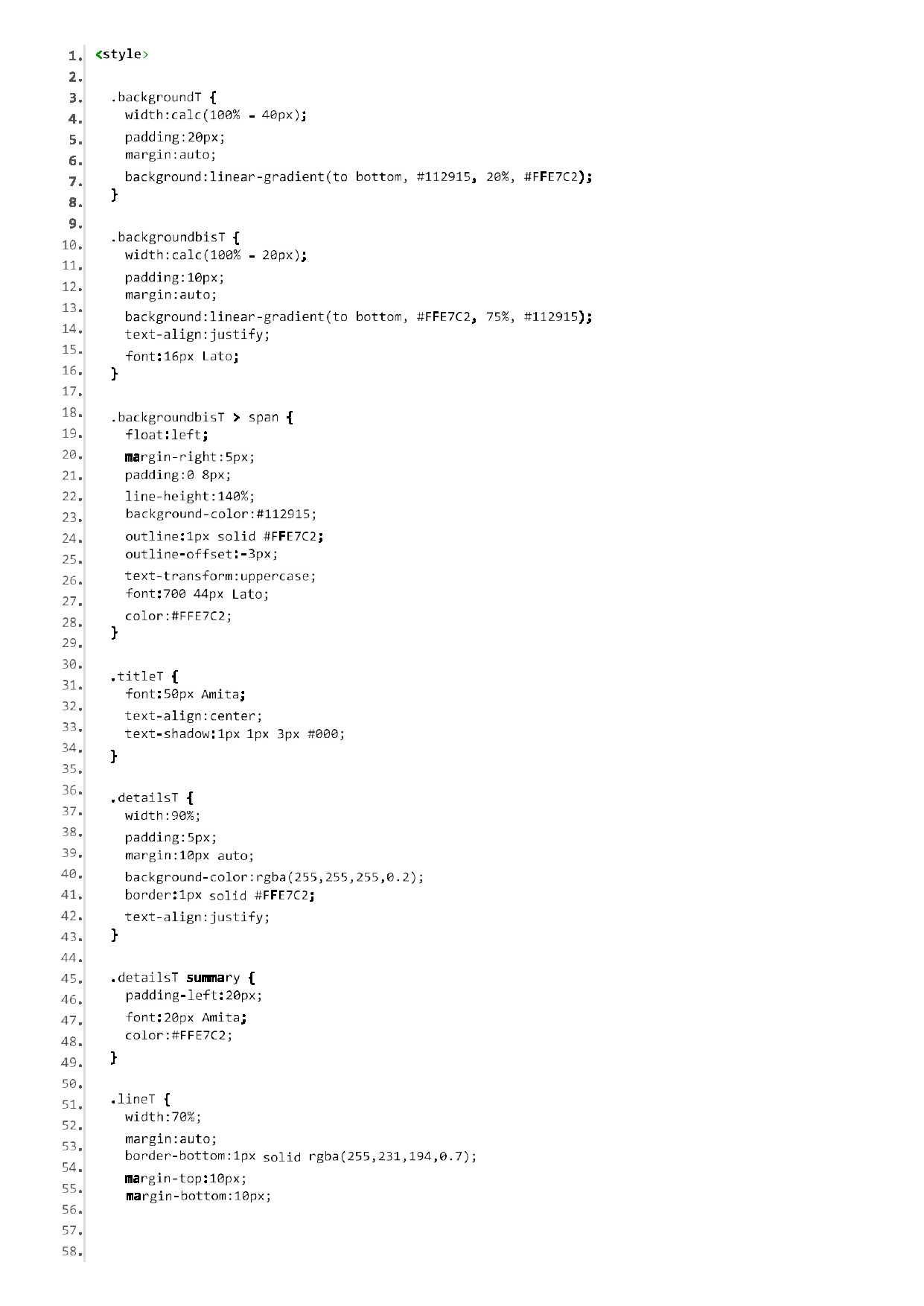
Python се [въвежда динамично](https://en.wikipedia.org/wiki/Type_system#DYNAMIC) и [се събира боклук](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) . Той поддържа множество [програмни парадигми](https://en.wikipedia.org/wiki/Programming_paradigm) , включително [структурирано](https://en.wikipedia.org/wiki/Structured_programming) (особено [процедурно](https://en.wikipedia.org/wiki/Procedural_programming) ), [обектно-ориентирано](https://en.wikipedia.org/wiki/Object-oriented_programming) и [функционално програмиране](https://en.wikipedia.org/wiki/Functional_programming) . Често се описва като език с „включени батерии“ поради изчерпателната си [стандартна библиотека](https://en.wikipedia.org/wiki/Standard_library) .

[Гуидо ван Росум](https://en.wikipedia.org/wiki/Guido_van_Rossum) започва да работи върху Python в края на 80-те години като наследник на [езика за програмиране ABC](https://en.wikipedia.org/wiki/ABC_(programming_language)) и го пуска за първи път през 1991 г. като Python 0.9.0.  Python 2.0 беше пуснат през 2000 г. Python 3.0, пуснат през 2008 г., беше основна ревизия, която не беше напълно [обратно съвместима](https://en.wikipedia.org/wiki/Backward_compatibility) с по-ранните версии. Python 2.7.18, издаден през 2020 г., беше последната версия на Python 2.

Python постоянно се нарежда като един от най-популярните езици за програмиране.

Анализ на изискванията

Изискванията, които определят какви услуги една система може да предостави на крайния потребител се наричат функционални изисквания. Функционалните изиск-вания са тясно свързани с изискванията на потребителските спецификации. Това може да включва изчисления, обработка на данни, технически операции и друга такава фун-кционалност, която има за цел да изпълни целите на приложението. Всички операции по проследяване, законови изисквания, подробности за интерфейса, нива на оториза-ция, актуализации на транзакции и административни функции отговарят на функцио-нални изисквания. Техническата архитектура на системата е определени от тези изис-квания.







Започваме с импортирането на двете библионеки request и bueatifusoup.

Въвеждаме модула csv който ще ни позволи да отваряме файла.

Създаваме host за да задедем конкретния сайт.

Създамаме константа url.

Вземаме заглавия от сайта за да не ни възприеме като робот сайта, от който извличаме информацията. За тази цел създаваме един речник headers във който поставяме страницата.

Създаваме друг речник, в който изграждаме accept достапа и user-agent.

Изработваме парсинг с функционално програмиrане, а не с обектно ориентирано програмиране. За тази цел създаваме няколко функции. Всички функции започват със def и името на функцията.

Първата get\_html функция е за получаване на html първият параметър е url вторият параметър е paramas това е препратката.

Изпращаме запитване към страницата чрез r=requests.get(url, headers=HEADERS,params=params). Функцията ни връща командата r

Започваме да извичаме информация от самия сайт със функцията def get\_content(html) зазавеме посака към самият сайт.

# Заключение:

Приложението ми е лесно достъпно, напълно безплатно така че всеки заинтересован да може да го използва. Направил съм така че да е лесно за използване, за да може без много усилия потребителя да намира информацията която търси. Мисля че ще е доста удобно за много хора, защото съм се съсредоточил върху най-големите футболни лиги и най-големите футболни отбори, които са най-гледаните и най-често търсените в световен мащаб. Това обаче е една от отрицателните страни на моето приложение - няма да мога да добавя всики отбори и няма да е интересно за феновете на по-малките отбори. Може би след време ще съм успял да добавя абсолютно всички отбори,обаче за жалост няма да мога да го направя в срока на проекта.

Темата е переспективна. Има много възможности за развитие

Използвана литература:

1. <https://en.wikipedia.org/wiki/Python_(programming_language)>
2. <https://www.youtube.com/watch?v=ykjBVT57r68&ab_channel=%D0%90%D0%BD%D0%B4%D1%80%D0%B5%D0%B9%D0%90%D0%BD%D0%B4%D1%80%D0%B8%D0%B5%D0%B2%D1%81%D0%BA%D0%B8%D0%B9>
3. <https://en.wikipedia.org/wiki/HTTP#Request_methods>
4. https://bg.wikipedia.org/wiki/HTTP