

باسمه تعالی



دانشگاه صنعتی شریف

دانشکده مهندسی برق

## گزارش کارآموزی

گرایش مخابرات

نگارش:

امیرحسین افشارراد

۹۵۱۰۱۰۷۷

استاد کارآموزی:

دکتر محمد معماریان

تابستان ۱۳۹۸

## چکیده

گزارشی که در دست دارید، مربوط به دوره‌ی کارآموزی تابستان سال ۱۳۹۸ در شرکت مهندسی تحول‌آفرین برنا (متاب) است. نگارنده‌ی این گزارش در این دوره دو پروژه‌ی جداگانه را با همکاری آقای احسان شریفیان و تحت سرپرستی جناب آقای محمدمهدی کیانی به انجام رسانده است. تمامی قسمت‌های هر دو پروژه با همکاری هر دو کارآموز صورت گرفته‌اند، با این حال در برخی موارد، یکی از دو کارآموز فعالیت بیشتری داشته و در برخی موارد دیگر، به صورت بالعکس. به همین دلیل در قسمت‌های مختلفی از این گزارش، برای توضیحات کامل‌تر و بیشتر به گزارش کارآموزی آقای احسان شریفیان ارجاعاتی داده شده است.

موضوع کلی هر دو پروژه استخراج اطلاعات از صفحات وب با استفاده از برنامه‌نویسی به زبان پایتون بوده است. پروژه‌ی اول استخراج متون چهار کتاب مذهبی قرآن، مفاتیح، نهج‌البلاغه، و صحیفه‌ی سجّادیه از پایگاه اطلاع‌رسانی دفتر استاد حسین انصاریان و طبقه‌بندی و مرتب‌سازی این اطلاعات در قالب پایگاه‌های داده‌ای از نوع SQLite بوده است.

پروژه‌ی دوم نیز در ارتباط استخراج اطلاعات و تحلیل داده‌های شبکه‌ی اجتماعی اینستاگرام بوده است. در این پروژه، ابزارهای لازم برای دستیابی خودکار به اطلاعات مورد نیاز از صفحات اینستاگرام طراحی شدند، و همچنین برخی ابزارهای مقدماتی برای تجزیه و تحلیل این اطلاعات نیز ایجاد شدند، اما فرصت محدود کارآموزی اجازه‌ی گسترش ابزارهای تحلیلی را نداد.

همچنین شایان ذکر است که بخشی از این دوره نیز تحت نظر سرپرست کارآموزی صرف یادگیری ابزارهای مورد نیاز برای انجام پروژه‌ها گردیده است.

این گزارش شرحی کامل از فرآیند انجام این دو پروژه و حواشی آن‌ها می‌باشد.

# فهرست مطالب

## چکیده

آ

۱	شرکت مهندسی تحوّل آفرین برنا (مَتَاب)	۱
۱	۱.۱ زمینه‌ی فعالیت شرکت و واحدهای اصلی آن	۱
۲	۲.۱ برخی فعالیت‌ها و محصولات شرکت	۲
۲	۳.۱ فعالیت انجام‌شده در دوره‌ی کارآموزی	۲
۴	۲ پروژه‌ی اوّل – دیتابیس کُتب مذهبی	۴
۴	۱.۲ مقدمه	۴
۵	۲.۲ یادگیری مقدّماتِ زبان پایتون	۵
۵	۳.۲ آشنایی با کتابخانه‌های مورد نیاز	۵
۶	۱.۳.۲ کتابخانه‌ی requests	۶
۶	۲.۳.۲ کتابخانه‌ی bs4	۶
۷	۴.۲ پایگاه‌داده‌ی SQLite و کتابخانه‌ی sqlite3	۷
۷	۵.۲ پایگاه داده‌ی نهج‌البلاغه	۷
۸	۱.۵.۲ ایجاد پایگاه داده و جدول فهرست	۸
۹	۲.۵.۲ دریافت فهرست خطبه‌ها از صفحه‌ی وب مورد نظر	۹
۱۳	۳.۵.۲ دریافت فهرست نامه‌ها از صفحه‌ی وب مورد نظر	۱۳

۴.۵.۲	تکمیل فهرست نام خطبه‌ها و نامه‌ها در پایگاه داده	۱۵
۵.۵.۲	ایجاد جدول خطبه‌ها	۱۷
۶.۵.۲	یافتن لیست آدرس صفحات خطبه‌ها	۱۹
۷.۵.۲	استخراج و ذخیره‌سازی متن عربی خطبه‌ها	۲۰
۸.۵.۲	استخراج و ذخیره‌سازی ترجمه‌ی خطبه‌ها	۲۲
۹.۵.۲	جدول نامه‌ها	۲۳
۱۰.۵.۲	استخراج متن عربی حکمت‌های نهج البلاغه	۲۵
۱۱.۵.۲	استخراج ترجمه‌های حکمت‌های نهج البلاغه	۲۸
۱۲.۵.۲	ذخیره‌ی حکمت‌های نهج البلاغه در پایگاه داده	۲۹
۱۳.۵.۲	جمع‌بندی	۳۱
۶.۲	پایگاه داده‌ی صحیفه‌ی سجّادیه	۳۱
۱.۶.۲	ایجاد پایگاه داده و جداول فهرست و دعاها	۳۱
۲.۶.۲	دریافت فهرست دعاها از صفحه‌ی وب مورد نظر	۳۲
۳.۶.۲	تکمیل جدول فهرست دعاها	۳۳
۴.۶.۲	ایجاد جدول دعاها و ذخیره‌سازی متون عربی دعاها	۳۴
۵.۶.۲	تکمیل جدول دعاها و افزودن ترجمه‌های فارسی	۳۵
۶.۶.۲	جمع‌بندی	۳۶
۷.۲	پایگاه داده‌ی قرآن کریم	۳۶
۸.۲	پایگاه داده‌ی مفاتیح الجنان	۴۰
۹.۲	جمع‌بندی	۴۳
۳	پروژه‌ی دوم - استخراج و تحلیل داده از شبکه‌ی اجتماعی اینستاگرام	۴۴
۱.۳	مقدمه	۴۴
۲.۳	آشنایی با کتاب‌خانه‌ی selenium	۴۵

۴۹	برنامه‌ی دریافت اطلاعات از اینستاگرام	۳.۳
۴۹	تابع <code>instagram_login</code>	۱.۳.۳
۵۳	تابع <code>get_posts_url_list</code>	۲.۳.۳
۵۵	تابع <code>get_comments</code>	۴.۳
۵۸	تابع <code>comments_to_csv</code>	۱.۴.۳
۵۸	تابع <code>comments_from_csv</code>	۲.۴.۳
۵۹	تابع <code>word_frequency</code>	۳.۴.۳
۶۰	تابع <code>total_word_count</code>	۴.۴.۳
۶۰	تابع <code>ignore_list_creator</code>	۵.۴.۳
۶۱	یک نمونه‌ی آزمایشی از عملکرد توابع	۵.۳
۶۳	جمع‌بندی	۶.۳
۶۷	جمع‌بندی، نتیجه‌گیری، و پیشنهادها	۴
۶۷	جمع‌بندی و نتیجه‌گیری	۱.۴
۶۸	پیشنهادها	۲.۴

## فهرست تصاویر

- ۱.۲ صفحه‌ی <http://www.erfan.ir/farsi/nahj/> برای استخراج فهرست نهج البلاغه . . . ۱۱
- ۲.۲ محتوای HTML صفحه‌ی <http://www.erfan.ir/farsi/nahj/> . . . . . ۱۲
- ۳.۲ بخشی از خطبه‌ی اوّل نهج البلاغه موجود در آدرس <http://www.erfan.ir/farsi/nahj1-1/> ۱۸
- ۴.۲ بخشی از محتوای متون عربی حکمت‌ها، بلافاصله پس از استخراج از کد HTML و بدون پردازش‌های بعدی . . . . . ۲۷
- ۵.۲ بخشی از لیست حکمت‌ها و شماره‌ی آن‌ها پس از پردازش اطلاعات استخراج‌شده . . . . . ۲۸
- ۱.۳ صفحه‌ی ورود به سامانه‌ی آموزش دانشگاه صنعتی شریف . . . . . ۴۶
- ۲.۳ صفحه‌ی ورود به سایت ترم‌ایناتور . . . . . ۴۷
- ۳.۳ اوّلین صفحه‌ی ظاهرشده پس از ورود به سایت ترم‌ایناتور . . . . . ۴۸
- ۴.۳ صفحه‌ی ورود به اینستاگرام . . . . . ۵۱
- ۵.۳ لیست کلمات پرتکرار صفحه‌ی ورزشی بدون حذف کلمات خنثی . . . . . ۶۲
- ۶.۳ لیست کلمات پرتکرار صفحه‌ی مربوط به فیلم و سریال بدون حذف کلمات خنثی . . . . . ۶۳
- ۷.۳ لیست کلمات پرتکرار صفحه‌ی سیاسی بدون حذف کلمات خنثی . . . . . ۶۴
- ۸.۳ لیست کلمات پرتکرار صفحه‌ی ورزشی پس از حذف کلمات خنثی . . . . . ۶۵
- ۹.۳ لیست کلمات پرتکرار صفحه‌ی مربوط به فیلم و سریال پس از حذف کلمات خنثی . . . . . ۶۵
- ۱۰.۳ لیست کلمات پرتکرار صفحه‌ی سیاسی پس از حذف کلمات خنثی . . . . . ۶۶

## فصل ۱

# شرکت مهندسی تحول آفرین برنا (مَتَاب)

## ۱.۱ زمینه‌ی فعالیت شرکت و واحدهای اصلی آن

مهندسی تحول آفرین برنا (مَتَاب) یک شرکت فعال در زمینه‌های فناوری اطلاعات<sup>۱</sup> و اینترنت اشیا<sup>۲</sup> است. این شرکت دارای چهار واحد کلّی با مسئولان مجرّاً به شرح زیر می‌باشد:

- واحد نرم افزار
- واحد سخت افزار
- واحد تولید و آماده سازی محصول
- واحد پردازش و تحلیل داده

البته واضح است که فرآیند تولید یک محصول در این شرکت، برآیندی از فعالیت برخی یا تمامی واحدهای فوق است و عملکرد این بخش‌ها، مستقل از یکدیگر نمی‌باشد.

مسئولین چهار واحد فوق به ترتیب آقایان احسان صادقی نژاد، حمیدرضا قهرمانی، حسام ثقفی، و محمد مهدی کیانی هستند. همچنین مدیرعامل شرکت نیز جناب آقای امیر شکوهی نیا می‌باشد.

در مورد جزئیاتی بیشتری نظیر ساختار جزئی بخش‌های شرکت و گردش مالی آن نیز اطلاعاتی در اختیار کارآموزان قرار داده نشده است.

---

<sup>1</sup>Information Technology (IT)

<sup>2</sup>Internet Of Things (IOT)

## ۲.۱ برخی فعالیت‌ها و محصولات شرکت

همان‌طور که در قسمت قبل اشاره شد، زمینه‌ی فعالیت شرکت متاب، فناوری اطلاعات و اینترنت اشیاء است. این شرکت به تولید محصولاتی در حوزه‌های مذکور و عرضه‌ی آن در بازار پرداخته است. این محصولات می‌توانند ایده‌ی خود شرکت یا سفارش کارفرما باشد. به عنوان نمونه، می‌توان به محصولاتی نظیر سیستم پایش سلامت نوزاد، سامانه‌ی پایش رانندگی خودرو، و گجت‌های خانه‌ی هوشمند اشاره کرد. صرفاً به عنوان یک مثال، در مورد یکی از این محصولات، یعنی سامانه‌ی پایش رانندگی خودرو در ادامه توضیحی به اختصار آورده شده است:

مخاطب این محصول به طور خاص، شرکت‌های بیمه هستند. عملکرد محصول به این صورت است که دستگاه برای مدتی معین (به عنوان مثال یک ماه) از طرف شرکت بیمه روی خودروی متقاضی بیمه نصب می‌شود و رانندگی فرد را مد نظر قرار می‌دهد. در پایان این بازه‌ی زمانی، این سیستم قادر به سنجش میزان سلامت و امنیت رانندگی فرد مورد نظر است، به گونه‌ای که می‌تواند به شرکت بیمه اعلام کند که با توجه به چگونگی رانندگی، این متقاضی چه میزان در معرض حوادث جاده‌ای است (آیا رانندگی او آرام و با پیروی از قوانین است، یا خطرناک و بعضاً با زیر پا گذاشتن قوانین رانندگی) و از این طریق، شرکت بیمه می‌تواند قیمت دریافتی از هر یک از متقاضیان را به عنوان تابعی از چگونگی رانندگی آن‌ها مشخص کند. همچنین این سیستم می‌تواند نه تنها برای مدتی معین بلکه به صورت دائمی رانندگی فرد را مورد پایش قرار دهد و در صورتی که به مرور زمان، شخص نحوه‌ی رانندگی خود را تغییر دهد، مبلغ تمدید بیمه‌ی وی نیز دستخوش تغییر شود. به این ترتیب، علاوه بر هوشمندی بیشتر و افزایش سود شرکت بیمه، افراد نیز به رانندگی سالم‌تر تشویق می‌شوند. پیاده‌سازی این سیستم مبتنی بر الگوریتم‌های یادگیری ماشین است تا بتواند الگوهای پرخطر رانندگی را بیاموزد و تشخیص دهد.

دو مورد دیگر از محصولات شرکت که نگارنده نیز در ساخت آن‌ها در دوره‌ی کارآموزی خود فعالیت داشته است، در ادامه‌ی این گزارش توصیف می‌شوند.

## ۳.۱ فعالیت انجام‌شده در دوره‌ی کارآموزی

نگارنده‌ی این گزارش در طول دوره‌ی کارآموزی خود، در این شرکت و تحت سرپرستی جناب آقای محمدمهدی کیانی و در واحد پردازش و تحلیل داده مشغول فعالیت بوده است. جناب آقای کیانی مسئول واحد پردازش و تحلیل داده در این شرکت بوده و علاوه بر آن، ریاست هیئت مدیره را نیز به عهده دارند. همچنین همان‌طور که در چکیده‌ی این گزارش نیز ذکر شد، آقای احسان شریفیان نیز در بخش‌های مختلف پروژه‌های محول‌شده از سوی شرکت در دوره‌ی کارآموزی، همکار نگارنده بوده است؛ لذا گزارش کارآموزی ایشان نیز محتوای مشابهی با این گزارش داشته، و علاوه بر آن، می‌تواند در برخی موارد مکمل مطالب مذکور در آن نیز باشد.

در طول مدت کارآموزی، نگارنده‌ی این گزارش در زیرمجموعه‌ی واحد پردازش و تحلیل داده، در آماده‌سازی دو



محصول فعالیت داشته است. به صورت کلی، فعالیت انجام‌شده در حوزه‌ی جمع‌آوری و تحلیل اطلاعات به صورت خودکار از صفحات وب و با استفاده از زبان برنامه‌نویسی پایتون بوده است. این فعالیت که با نام Web Scraping شناخته می‌شود، عبارت است از ساختن برنامه‌ای که بتواند مطابق خواست برنامه‌نویس، با اتصال به اینترنت، اطلاعات مطلوب را از صفحات وب بخواند و در ساختار مطلوب و مورد نیاز، ذخیره کند و تحویل دهد. در ادامه، می‌توان با استفاده از الگوریتم‌های تحلیل داده به بررسی این اطلاعات و دستیابی به نتایج مطلوب پرداخت.

همان‌طور که گفته شد، در حوزه‌ی دریافت و تحلیل اطلاعات از صفحات وب، نگارنده در تکمیل و آماده‌سازی دو محصول مختلف در این شرکت فعالیت داشته است. عناوین این دو محصول به شرح زیر است. توضیحات مبسوط در فصل‌های بعدی ذکر می‌شود:

۱. دیتابیس کامل چهار کتاب قرآن، نهج‌البلاغه، صحیفه‌ی سجّادیه، و مفاتیح‌الجنان که اطلاعات آن به صورت آنلاین از سایت <http://www.erfan.ir/> گرفته می‌شود و قابلیت آپدیت شدن دارد.

۲. سامانه‌ی دریافت و تحلیل اطلاعات از شبکه‌ی اجتماعی اینستاگرام<sup>۳</sup>

---

<sup>۳</sup>Instagram

## فصل ۲

# پروژه‌ی اول – دیتابیس کُتب مذهبی

## ۱.۲ مقدمه

همان طور که در قسمت چکیده و نیز در فصل اول گزارش ذکر شد، این کارآموزی شامل دو پروژه‌ی جداگانه (هر دو بر مبنای استخراج اطلاعات از صفحات وب – Web Scraping) بوده است. پروژه‌ی اول که در این فصل مورد بررسی قرار می‌گیرد، مربوط به استخراج محتوای چهار کتاب مذهبی قرآن، نهج البلاغه، صحیفه‌ی سجّادیه، و مفاتیح‌الجنان از صفحات وب و در ادامه، طبقه‌بندی و تحویل آن‌ها در قالب دیتابیس‌هایی از خانواده‌ی SQL بوده است. برنامه‌ای که این کار را به صورت خودکار انجام می‌دهد، موظف بوده تا بتواند همواره قابل اجرا باشد تا در صورتی که اطلاعات مرجع (صفحه‌ی وبی که اطلاعات از آن خوانده می‌شود) دستخوش تغییر گردد، دیتابیس‌ها نیز با آخرین تغییرات آن به‌روزرسانی شوند. به عبارت دیگر، وظیفه‌ی کارآموز تنها تحویل دیتابیس‌ها نبوده است، بلکه تحویل برنامه‌ای بوده است که در هر زمان بتواند دیتابیس‌ها را از ابتدا و با دسترسی به وب‌سایت مورد نظر بسازد. وب‌سایت مورد استفاده برای دریافت این اطلاعات، پایگاه اطلاع‌رسانی دفتر استاد حسین انصاریان به آدرس <http://www.erfan.ir/> بوده است.

شرکت متاب این محتوا را به عنوان یکی از محصولات رایگان خود (که هم دیتابیس‌ها و هم کد آن در اختیار عموم قرار می‌گیرد) مدّ نظر داشته است.

برای پیاده‌سازی این پروژه، مراحل زیر توسط نگارنده طی شده‌اند که در ادامه به تفصیل مورد بررسی قرار می‌گیرند:

- یادگیری مقدماتی زبان پایتون<sup>۱</sup>

- آشنایی با زبان HTML<sup>۲</sup>

- آشنایی با کتابخانه‌های requests و bs4<sup>۳</sup> به منظور دریافت اطلاعات از صفحات وب و پردازش کد HTML

---

<sup>۱</sup>Python

<sup>۲</sup>Hypertext Markup Language

<sup>۳</sup>Beautiful Soup

- آشنایی با ساختار برنامه‌نویسی سایت <http://www.erfan.ir/> به منظور استخراج اطلاعات از آن
- آشنایی با دیتابیس SQLite و چگونگی ارتباط برقرار کردن با این دیتابیس از طریق زبان برنامه‌نویسی پایتون به منظور ذخیره‌سازی اطلاعات استخراج‌شده از صفحات وب
- پیاده‌سازی پیکره‌ی اصلی برنامه با استفاده از موارد مذکور

## ۲.۲ یادگیری مقدماتی زبان پایتون

نظر به این که نگارنده پیش از ورود به دوره‌ی کارآموزی، تجربه‌ی چندانی در کار با زبان پایتون نداشت، نیاز بر آن بود که در ابتدا با مقدمات و اصول اولیه‌ی این زبان آشنا شود تا بتواند پروژه‌های محوّل از سوی شرکت را به نحو احسن پیاده‌سازی کرده و انجام دهد. به همین منظور، زمانی در حدود ۱/۵ هفته از دوره‌ی کارآموزی (حدود ۵۰ ساعت) به یادگیری مقدمات و اصول اولیه و نیز برخی کاربردهای پیشرفته‌ی زبان پایتون اختصاص داده شد. صرف این زمان با نظارت و تأیید سرپرست کارآموزی انجام شده است. برای این منظور، کارآموز در دو دوره‌ی آموزش زبان پایتون به صورت آنلاین شرکت نموده است. این دو دوره در سایت «مکتب‌خونه» به آدرس <https://maktabkhooneh.org/> تحت عناوین «برنامه‌نویسی پایتون مقدماتی» و «برنامه‌نویسی پایتون پیشرفته» قابل دسترسی می‌باشند.

همچنین لازم به ذکر است که این دوره‌های آموزشی رایگان نمی‌باشند و کارآموز برای دریافت آن‌ها به ترتیب مقادیر ۱۷۹۰۰۰ تومان و ۲۲۹۰۰۰ تومان به صورت شخصی و بدون حمایت مالی از جانب محلّ کارآموزی هزینه کرده است. (البته همان‌طور که در چکیده و فصل اوّل این گزارش نیز ذکر شد، نگارنده در تمامی مراحل کارآموزی با آقای احسان شریفیان همکاری کرده، و این هزینه‌ها نیز مشترکاً توسط این دو نفر و به مساوات صورت گرفته است.)

## ۳.۲ آشنایی با کتابخانه‌های مورد نیاز

مهم‌ترین عاملی که امروزه باعث شده است تا زبان برنامه‌نویسی پایتون به یکی از محبوب‌ترین و روبه‌رشدترین زبان‌های برنامه‌نویسی تبدیل شود، وجود کتابخانه‌های متعدّد برای این زبان است؛ به گونه‌ای امروزه تقریباً برای هر کاری که به ذهن برسد، کتابخانه‌ای در زبان پایتون موجود است. استخراج اطلاعات از صفحات وب نیز از این امر مستثنی نیست و کتابخانه‌های بسیاری مفیدی در زبان پایتون برای انجام این کار در دسترس است. در این پروژه، دو کتابخانه‌ی مهمّی که مورد بررسی و استفاده قرار گرفته‌اند، کتابخانه‌های `requests` و `bs4` می‌باشند. همچنین در پروژه‌ی بعدی که در فصل آینده مورد بررسی قرار خواهد گرفت نیز از یک کتابخانه‌ی بسیار مشهور و قدرتمند به نام `selenium` استفاده شده است که در همان فصل به بررسی آن پرداخته خواهد شد.

## ۱.۳.۲ کتابخانه‌ی requests

این کتابخانه قابلیت‌های متنوعی دارد که می‌توان با مراجعه به آدرس [https://pypi.org/project/](https://pypi.org/project/requests/) requests/ آن‌ها را بررسی نمود؛ اما قابلیتی که به طور خاص در این پروژه مورد نظر بود و از آن استفاده شد، آن است که با استفاده از این کتابخانه می‌توان به کمک یک برنامه به زبان پایتون، کد یک صفحه‌ی وب را به زبان HTML دریافت و ذخیره کرد و در ادامه اطلاعات لازم را از آن استخراج نمود. برای این کار کافی است از دستور `requests.get` استفاده شود. به عنوان مثال، قطعه کد زیر، کد HTML سایت <http://www.erfan.ir/> را دریافت کرده و در متغیر `code` ذخیره می‌کند:

```
import requests
code = requests.get('http://www.erfan.ir/')
```

در این حالت، می‌توان به کد HTML این صفحه دسترسی داشت و کافی است به متغیری که کد در آن ذخیره شده است، پسوند `.text` اضافه شود. به عنوان مثال، پس از اجرای کد فوق، برای آن که کد HTML مربوطه را بر روی صفحه‌ی نمایش چاپ کنیم، کافی است از دستور زیر استفاده نماییم:

```
print(code.text)
```

بنابراین طبق توضیحات فوق، می‌توانیم با دانستن آدرس صفحه‌ی وب مورد نظر، به کد HTML آن دست پیدا کنیم و در ادامه، اطلاعات مورد نیاز را از آن استخراج کنیم. در بخش‌های آتی، نمونه‌هایی از پردازش و استخراج اطلاعات از چنین کدهایی مورد بررسی قرار خواهند گرفت.

## ۲.۳.۲ کتابخانه‌ی bs4

کتابخانه‌ی Beautiful Soup که به طور خلاصه آن را با نام `bs` می‌شناسیم، به طور کلی قابلیت آن را در اختیار ما قرار می‌دهد که بتوانیم کدهای دریافتی از صفحات وب را به آسانی پردازش نماییم. این کتابخانه ساختارهای مورد استفاده در کدهای HTML و XML را می‌شناسد؛ و به همین دلیل می‌تواند پردازش این کدها و جست‌وجو در آن‌ها را بسیار ساده کند. برای استفاده از این قابلیت‌ها، ابتدا باید یک شیء از جنس `bs4.BeautifulSoup` ساخته شود. برای این کار، کافی است در ادامه‌ی کد قبلی (که محتوای یک سایت به زبان HTML را دریافت و ذخیره کردیم)، دستور زیر را اجرا کنیم:

```
soup = BeautifulSoup(code.text, 'html.parser')
```

ورودی اوّل تابع `BeautifulSoup` متن کد صفحه‌ی وب است و ورودی دوم آن نیز زبان کد را مشخص می‌کند

که در این جا، HTML است.

در ادامه می‌توانیم از امکانات کتابخانه‌ی bs4 استفاده کنیم. مهم‌ترین متد (تابع) مورد استفاده در این پروژه، متد `find_all` است که با دریافت تگ<sup>۴</sup> و ویژگی<sup>۵</sup> مورد نظر، تمامی مواردی که دارای آن تگ و ویژگی باشند را از کد HTML استخراج می‌کند و در اختیار قرار می‌دهد. نمونه‌هایی از کاربرد این متد را در ادامه خواهیم دید.

## ۴.۲ پایگاه داده‌ی SQLite و کتابخانه‌ی sqlite3

همان طور که پیش‌تر نیز بیان شد، هدف نهایی این پروژه تحویل محتوای کتب مذهبی مذکور در قالب دیتابیس بود. به همین منظور، در این پروژه (و به توصیه‌ی سرپرست کارآموزی) از دیتابیس SQLite که یک پایگاه داده‌ی سبک و ساده از خانواده‌ی دیتابیس‌های SQL است، استفاده شده است. ارتباط برقرار کردن با این پایگاه داده از نظر دستورات و زبان برنامه‌نویسی، مشابه با سایر دیتابیس‌های خانواده‌ی SQL است. تنها موردی که مضاف بر کار با دیتابیس لازم بود مورد توجه قرار گیرد، آن بود که در این پروژه لازم بود از طریق برنامه‌ای به زبان پایتون با پایگاه داده ارتباط برقرار شود و اطلاعات به آن فرستاده شده یا از آن خوانده شود. به همین منظور، از کتابخانه‌ی sqlite3 استفاده کردیم که به واسطه‌ی آن، می‌توان هر نوع ارتباط مورد نیاز را از طریق زبان پایتون با دیتابیس برقرار نمود. همان طور که پیش‌تر اشاره شد، وجود کتابخانه‌های متعدّد برای انجام تقریباً هر کاری در زبان پایتون، یکی از مهم‌ترین ویژگی‌های این زبان می‌باشد. یکی از این موارد، همین کتابخانه‌ی sqlite3 می‌باشد که این امکان را می‌دهد تا دقیقاً با همان زبانی که می‌توان مستقیماً با دیتابیس ارتباط برقرار کرد، قابلیت برقراری ارتباط از طریق برنامه‌ی پایتون نیز مهیا باشد تا بتوان بدون یادگیری دستورات جدید و صرفاً با استفاده از همان دستورات استاندارد SQL به پایگاه داده دسترسی پیدا نمود. جزئیات استفاده از این کتابخانه در بخش‌های آینده ذکر خواهد شد.

## ۵.۲ پایگاه داده‌ی نهج‌البلاغه

در این بخش به توضیح کامل برنامه‌ای می‌پردازیم که با مراجعه با سایت <http://www.erfan.ir/>، محتوای کتاب نهج‌البلاغه را دریافت می‌کند و آن را در یک دیتابیس SQLite ذخیره کرده و در خروجی تحویل می‌دهد. در این بخش، کد پایتون مورد استفاده را به صورت قطعه به قطعه مورد بررسی قرار می‌دهیم. در هر مرحله ابتدا کد مربوطه آورده می‌شود و در ادامه، توضیحات مربوط به آن ذکر می‌گردد.

<sup>۴</sup>tag

<sup>۵</sup>attribute

## ۱.۵.۲ ایجاد پایگاه داده و جدول فهرست

```
import sqlite3

conn = sqlite3.connect('Nahj.db')
c = conn.cursor()
c.execute("CREATE TABLE list (ID integer, category, catID integer, title)")

conn.commit()
conn.close()
```

در ابتدا، کتاب‌خانه‌ی `sqlite3` فراخوانده شده است. پیش‌تر ذکر شد که هدف از این کتاب‌خانه، برقراری ارتباط میان برنامه و دیتابیس‌ی از نوع `SQLite` است. ایجاد این ارتباط با استفاده از متد `connect` صورت می‌گیرد. متغیر `conn` که در کد تعریف شده است، در واقع همان اتّصالی است که بین برنامه و دیتابیس‌ی با نام `Nahj.db` برقرار شده است. در صورتی که چنین دیتابیس‌ی وجود نداشته باشد، اجرای دستور فوق موجب ایجاد آن نیز می‌شود.

در ادامه باید یک مکان‌نما<sup>۶</sup> نیز ساخته شود. به واسطه‌ی این مکان‌نما می‌توانیم دستورات مورد نیاز را به دیتابیس بدهیم. برای ایجاد آن نیز کافی است از متد `cursor` استفاده شود.

حال برنامه آماده است تا دستورات مورد نیاز را به دیتابیس بدهد. با استفاده از متد `execute` می‌توانیم دقیقاً با همان زبانی که در حالت عادی و بدون واسطه به دیتابیس وصل می‌شویم، به آن فرمان بدهیم و تغییرات لازم را در آن اعمال کنیم. در این جا با استفاده از متد `execute` یک جدول جدید به نام `list` ایجاد می‌کنیم. این جدول به نوعی مشابه فهرست نهج البلاغه عمل خواهد کرد که دارای چهار فیلد زیر می‌باشد:

- **ID:** این فیلد، یک شماره‌ی سراسری به همه‌ی خطبه‌ها، نامه‌ها و حکمت‌ها می‌دهد. به عبارت دیگر، این شماره تعداد عناوین موجود در فهرست را می‌شمارد.
- **category:** این فیلد، نوع هر عنوان را در فهرست مشخص می‌کند. عناوین موجود در نهج البلاغه سه نوع هستند: خطبه، نامه، و حکمت.
- **catID:** این فیلد، یک شماره‌گذاری دیگر است که داخل هر یک از سه نوع محتوا صورت می‌گیرد. یعنی خطبه‌ها به ترتیب از شماره‌ی ۱ شماره‌گذاری می‌شوند؛ و هنگامی که خطبه‌ها به پایان رسیده و به نامه‌ها می‌رسیم، این شماره (برعکس فیلد `ID`) مجدداً از ابتدا (از شماره‌ی یک) شروع به شمردن می‌کند.

<sup>۶</sup>cursor

● title: این فیلد، عنوان هر یک از موارد موجود در فهرست را ذخیره می‌کند. به عنوان مثال، اوّلین خطبه‌ی نهج‌البلاغه، «خطبه در ابتدای آفرینش آسمان و زمین» نام دارد. این نام و موارد مشابه آن برای سایر خطبه‌ها و نامه‌ها در فیلد title ذخیره می‌شوند.

پس از آن که همه‌ی دستورات مدّ نظر از طریق متد execute اعمال شدند، می‌توان به سادگی تحقیق کرد که تغییرات مربوطه هنوز در دیتابیس اعمال نشده‌اند. برای آن که این تغییرات در دیتابیس ذخیره و اعمال شوند، لازم است که آن‌ها را ثبت کنیم. برای این کار، از متد commit استفاده می‌شود. با اجرای دستور conn.commit، دستوراتی که در خطّ قبل داده شده بودند، در دیتابیس ثبت می‌شوند و در این مثال خاص، جدول مربوطه با چهار فیلد مذکور ساخته می‌شود.

در نهایت و هرگاه که کار ما (فعلاً) با دیتابیس تمام شود، خوب است با استفاده از متد close از دیتابیس خارج شویم و اتّصال برقرارشده با آن را از بین ببریم، هر چند عدم انجام این کار نیز مشکلی در فرآیند ثبت اطلاعات در دیتابیس ایجاد نمی‌کند.

## ۲.۵.۲ دریافت فهرست خطبه‌ها از صفحه‌ی وب مورد نظر

```
import requests
from bs4 import BeautifulSoup

req = requests.get('http://www.erfan.ir/farsi/nahj/')
soup = BeautifulSoup(req.text, 'html.parser')
```

در این بخش از کد، دو کتابخانه‌ی requests و BeautifulSoup فراخوانده شده‌اند. در مورد این دو کتابخانه پیش‌تر در بخش‌های ۱.۳.۲ و ۲.۳.۲ توضیحاتی داده شده است. در این جا متغیّر req اطلاعات صفحه‌ی وب به آدرس <http://www.erfan.ir/farsi/nahj/> را دریافت می‌کند. در ادامه متغیّر soup یک شیء از جنس BeautifulSoup.bs4 است که آماده است تا در ادامه، از امکانات کتابخانه‌ی BeautifulSoup برای جست‌وجو و یافتن محتوای مدّ نظر در آن استفاده کنیم.

```
import re
a = soup.find_all('a', href = lambda x: x and 'خطبه' in x)

lst = []
for i in a:
    s = i.text.strip()
```

```
s = re.sub('\d','',s)
s = re.sub('\( خطبه \)','',s)
lst.append(s)
```

در این بخش، در ابتدا کتابخانه‌ی re فراخوانده شده است. re مخفف Regular Expression است که بعضاً به آن regex نیز گفته می‌شود. regexها رشته‌هایی از علائم و کاراکترها هستند که برای جست‌وجو و جایگزینی یک عبارت با عبارت دیگر به کار می‌روند. به عبارت دیگر، می‌توان گفت که regexها یک زبان شناخته‌شده هستند که به کمک آن می‌توان الگوهای مدّ نظر را در یک متن جست‌وجو کرد. برای آشنایی بیشتر با این الگوها می‌توانید: [https://en.wikipedia.org/wiki/Regular\\_expression](https://en.wikipedia.org/wiki/Regular_expression) را بررسی کنید و نیز برای آزمون regexهای خود می‌توانید از وبسایت <https://regex101.com/> استفاده نمایید. در این جا بیش از این به توضیح این مبحث نمی‌پردازیم و در ادامه، هرگاه که به آن‌ها نیاز پیدا کنیم، توضیح مقتضی را در مورد آن‌ها ارائه می‌کنیم.

در ادامه، از متد find\_all برای یک شیء از جنس bs4.BeautifulSoup استفاده شده است. برای استفاده از این متد، در حالت کلی باید دو ورودی داده شود: اوّل تگ مورد نظر (که در این جا 'a' است) و دوم ویژگی مطلوب. در این مثال، ویژگی مورد نظر، آدرس لینک است که با کلمه‌ی href در زبان HTML شناخته می‌شود. به عنوان مثال، دستور `soup.find_all('a', href = 'https://google.com')` همه‌ی المان‌هایی در کد HTML را پیدا می‌کند که دارای تگ a بوده و لینک اتّصالی به صفحه‌ی اصلی گوگل دارند.

بنابراین واضح است که می‌توان به سادگی href مورد نظر را در ورودی متد find\_all نوشت و همه‌ی موارد دارای آن را پیدا کرد؛ با این حال در قطعه‌کدی که در این بخش به توضیح آن می‌پردازیم، کاربرد پیچیده‌تری از این متد استفاده شده است. مشاهده می‌شود که دستور زیر در کد آمده است:

```
a = soup.find_all('a', href = lambda x: x and 'خطبه' in x)
```

منظور از این خط از کد آن است که همه‌ی المان‌هایی از کد HTML پیدا شوند که تگ آن‌ها به صورت 'a' است و در href آن‌ها، کلمه‌ی «خطبه» موجود است.

برای آن که واضح شود چطور این کد باید نوشته شود، دو مورد را توضیح می‌دهیم: اوّل آن که در زبان پایتون، عبارت lambda برای تعریف توابعی به کار می‌رود که به صورت یک‌بار مصرف استفاده می‌شوند و نمی‌خواهیم مانند سایر توابعی که تعریف می‌کنیم، برای آن‌ها نامی در نظر بگیریم. به عنوان مثال، دستور `x+2` : `lambda x` اشاره به تابعی دارد که ورودی را می‌گیرد و به آن دو واحد می‌افزاید. البته می‌شد این تابع را به صورت رسمی تعریف کنیم و برای آن نامی در نظر بگیریم و هرگاه نیاز بود به آن تابع اشاره کنیم. از آن نام استفاده کنیم. با این حال، تعریف این تابع با دستور lambda باعث می‌شود که بدون تخصیص نام، تنها یک‌بار آن را تعریف کرده و استفاده کرده باشیم. این کار زمانی مفید است که بخواهیم خود تابع را به عنوان ورودی به تابعی دیگر بدهیم تا از آن استفاده کند. در این مثال خاص،



از این ساختار برای معرفی عباراتی استفاده شده است که در آن‌ها کلمه‌ی «خطبه» به کار گرفته شده است.

مورد دیگری که لازم است توضیح داده شود، آن است که چطور متوجه می‌شویم اگر به شکل مذکور جست‌وجو را انجام دهیم، به هدف مطلوب خود می‌رسیم. به عبارت دیگر، جست‌وجوی فوق در نتیجه نام تمامی خطبه‌های نهج البلاغه را در اختیار ما می‌دهد؛ اما سؤال این است که چرا چنین روشی را برای یافتن آن‌ها استفاده کرده‌ایم. چنین سؤال‌ی هیچ‌گاه پاسخ یکتا و مشخص ندارد؛ و روش یافتن پاسخ آن همواره در گرو بررسی صفحه‌ای است که می‌خواهیم اطلاعات را از آن استخراج کنیم. پیش‌تر مشاهده شد که در این قسمت از کد پایتون، در حال استخراج اطلاعات از صفحه به آدرس <http://www.erfan.ir/farsi/nahj/> هستیم. محتوای این صفحه در شکل ۱.۲ مشاهده می‌شود.



شکل ۱.۲: صفحه‌ی <http://www.erfan.ir/farsi/nahj/> برای استخراج فهرست نهج البلاغه

برای آن که بتوانیم به کد HTML این صفحه و ارتباط آن با آنچه در ظاهر صفحه مشاهده می‌شود، دست پیدا کنیم، با فشار دادن کلیدهای Ctrl+Shift+I، صفحه را به صورت شکل ۲.۲ در می‌آوریم.

مشاهده می‌شود که در این حالت، قسمتی به کنار صفحه اضافه می‌شود که در آن، محتوای همین صفحه به صورت HTML قابل مشاهده است. در این حالت اگر اشاره‌گر ماوس را روی هر یک از خطوط کد ببریم، قسمت متناظر با آن در صفحه‌ی اصلی رنگی و مشخص می‌شود. از طرف دیگر، با فشردن دکمه‌ای که در بالای پنل سمت راست با علامت اشاره‌گر ماوس ظاهر می‌شود، می‌توانیم برعکس نیز عمل کنیم؛ یعنی با حرکت دادن ماوس روی صفحه‌ی اصلی وب‌سایت، هرگاه ماوس روی المانی در صفحه‌ی اصلی قرار گیرد، کد HTML متناظر با آن در پنل سمت راست مشخص می‌شود. به این ترتیب می‌توانیم ماوس را روی قسمت‌هایی از صفحه که برای ما مهم هستند ببریم، و مشاهده کنیم که کد HTML متناظر با آن‌ها به چه صورت است. با این کار می‌توانیم بفهمیم که هنگام نوشتن کد پایتون، باید چه چیزی را در کد HTML جست‌وجو کنیم تا چیزی که مد نظر ماست، دقیقاً پیدا شود.

در این مثال، با بررسی عناوین خطبه‌ها که در شکل‌های ۱.۲ و ۲.۲ مشخص است، پی‌بردیم که همه‌ی این عناوین

شکل ۲.۲: محتوای HTML صفحه‌ی <http://www.erfan.ir/farsi/nahj/>

در کد HTML خود، دارای تگ 'a' بوده و ویژگی‌های href ای دارند که در نام آن‌ها، کلمه‌ی فارسی «خطبه» موجود است؛ و از طرف دیگر تنها این عناصر هستند که چنین ویژگی‌ای دارند. بنابراین مطابق آنچه پیش‌تر توضیح دادیم، پس از دریافت کد کامل HTML این صفحه، به دنبال عناصری در آن گشتیم که تگ 'a' داشته باشند و در ویژگی 'href' آن‌ها، کلمه‌ی «خطبه» موجود باشد.

این نحوه‌ی جست‌وجو که در این جا به تفصیل توضیح داده شد، در تمامی قسمت‌های دیگر این پروژه و نیز پروژه‌ی بعدی مورد استفاده قرار گرفته است و تنها راه مناسب برای دریافت اطلاعات به صورت خودکار از صفحات وب نیز همین رویکرد می‌باشد.

حال به ادامه‌ی توضیح کد برمی‌گردیم. پس از آن که جست‌وجوی مربوطه انجام شد، همه‌ی موارد یافت‌شده در متغیر a ذخیره می‌شوند. با این حال، دو مشکل جزئی در این مرحله وجود دارند که باید حل شوند؛ اوّل آن که جنس متغیر a یک لیست عادی پایتون نیست، بلکه وابسته به کتابخانه‌ی BeautifulSoup است و برای سادگی کار کردن در مراحل بعد، بهتر است آن را به یک لیست عادی پایتون تبدیل نماییم. برای این کار، یک لیست خالی به نام lst ایجاد کرده‌ایم که در ادامه، عناصر مطلوب را به آن می‌افزاییم. لازم به ذکر است که نوع لیست، یکی از انواع داده در زبان پایتون است که مشابه با آرایه‌هایی است که در زبان‌هایی نظیر C++ استفاده می‌شوند.

دومین مشکل، آن است که ما فهرستی از نام خطبه‌های نهج البلاغه می‌خواهیم، در حالی که آن‌چه تا کنون استخراج کرده‌ایم، کمی زوائد دارد. به عنوان مثال، یکی از عناوین استخراج شده به صورت زیر است:

» ( خطبه ۱ ) خطبه در ابتدای آفرینش آسمان و زمین و آدم «

توجه کنید که در عبارت فوق، علائم « » جزء متن استخراج‌شده نیستند و آن‌چه بین آن‌ها نوشته شده است، دقیقاً متنی

است که استخراج شده است. اوّلین موردی که مشاهده می‌شود، آن است که دو طرف آن مقادیری فاصله‌ی بی‌مورد وجود دارد. برای از بین بردن آن، از متد `strip` استفاده شده است. این متد، فاصله‌های اضافی را از ابتدا و انتهای یک رشته حذف می‌کند. مورد دیگری که می‌خواهیم در فهرست وجود نداشته باشد، عبارت « (خطبه ۱) » است؛ چرا که در دیتابیس مورد استفاده، فیلدهایی جهت شماره‌گذاری خطبه‌ها موجود هستند و نیازی نیست شماره‌ی آن‌ها در نام خطبه نیز موجود باشد. برای رفع این مشکل، از متد `sub` از کتاب‌خانه‌ی `re` استفاده می‌کنیم. در مورد `regex`ها توضیحی عمومی در همین قسمت داده شد. متد `sub` برای جایگزینی استفاده می‌شود. ابتدا در تمامی عبارات حاصل از جست‌وجو، کاراکترهای عددی را از بین می‌بریم. برای نشان دادن اعداد به زبان `regex`ها، از نماد `\d` استفاده می‌کنیم. کد استفاده شده به صورت `s = re.sub('\d', '', s)` است. این یعنی اعداد در متغیر `s` با عبارت تهی جایگزین شوند، که یعنی کاراکترهای عددی از بین بروند.

در نهایت در خطّ بعدی، دستوری به صورت `s = re.sub('\ (خطبه \)', '', s)` استفاده شده است. این بدان معناست که هرگاه در متغیر `s` ساختاری وجود داشته باشد که شروع و پایان آن پُرانتز باشد و درون آن کلمه‌ی «خطبه» نوشته شده باشد، آن را با کاراکتر تهی جایگزین می‌کند؛ یا به عبارت دیگر آن را حذف می‌کند. بنابراین برآیند دو دستور اخیر باعث می‌شود که از نام هر یک از خطبه‌ها، عبارت ابتدایی آن که در آن شماره‌ی خطبه نوشته شده بود حذف شود، و عبارتی نظیر

« (خطبه ۱) خطبه در ابتدای آفرینش آسمان و زمین و آدم »

به عبارت

«خطبه در ابتدای آفرینش آسمان و زمین و آدم»

تبدیل شود. در این جا، عنوان خطبه دقیقاً همان‌طور که مورد انتظار است، به دست می‌آید و با دستور `lst.append`، به انتهای لیست `lst` افزوده می‌شود. وقتی حلقه‌ی `for` به پایان برسد، لیست `lst` حاوی نام تمامی خطبه‌های نهج البلاغه خواهد بود و این مرحله از کار با موفقیت به اتمام می‌رسد.

## ۳.۵.۲ دریافت فهرست نامه‌ها از صفحه‌ی وب مورد نظر

```
import re

a = soup.find_all('a', href = lambda x: x and 'نامه' in x)

lst2 = []
for i in a:
    s = i.text.strip()
```

```
s = re.sub('\d',' ',s)
s = re.sub('\( نامه \)',' ',s)
s = re.sub('\s+',' ',s)
if ' نامه ' in i.text:
    lst2.append(s)
```

این قسمت، شباهت بسیار زیادی به قسمت ۲.۵.۲ دارد که در آن، لیست خطبه‌های نهج البلاغه را استخراج نمودیم. این بار، می‌خواهیم لیست نامه‌ها را بیابیم. دقیقاً به روش مشابه عمل می‌کنیم؛ ابتدا به دنبال المان‌هایی در کد HTML می‌گردیم که در ویژگی href خود، دارای کلمه‌ی «نامه» هستند. در ادامه، یک لیست خالی به نام lst2 ایجاد می‌کنیم و به ویرایش نتایج حاصل از جست‌وجو می‌پردازیم و در نهایت، نام هر یک از نامه‌ها را به آن لیست اضافه می‌کنیم. مراحل ویرایش نیز مشابه هستند؛ ابتدا با استفاده از متد strip، فاصله‌های اضافی را از ابتدا و انتهای اسم نامه‌ها حذف می‌کنیم؛ سپس در دو مرحله و با استفاده از متد sub از کتابخانه‌ی re (که مربوط به کار با regexها بود)، خود عبارت «(نامه)» را نیز از ابتدای اسم نامه‌ها حذف می‌کنیم (چرا که در فیلد category در دیتابیس، به صورت جداگانه بین نامه‌ها و خطبه‌ها تمایز قائل می‌شویم، و نیازی نیست که به صورت جداگانه در نام‌ها نیز نوع نامه یا خطبه مشخص باشد).

در این مرحله، تفاوتی با قسمت ۲.۵.۲ مشاهده می‌شود. قابل مشاهده است که یک بار دیگر نیز از دستور sub استفاده شده است و ساختار \s+ با یک فاصله (کاراکتر space) جایگزین شده است. منظور از \s در ادبیات regexها، همان فاصله یا space است و منظور از کاراکتر + که در انتهای آن اضافه شده است، آن است که هر تعداد ناصفری از کاراکتر قبلی که به صورت متوالی در متن ظاهر شده باشد در نظر گرفته شود. برای فهم بهتر، یک مثال می‌زنیم: اگر در یک regex کاراکتر a استفاده شود، در جست‌وجو، هر گاه با این کاراکتر مواجه شویم، یک بار محاسبه می‌شود. بنابراین اگر به عنوان مثال دستور دهیم که چنین کاراکتری را با b جایگزین کند، عبارت aaa به bbb تبدیل می‌شود. حال اگر یک کاراکتر + نیز به a بیفزاییم و a+ را جست‌وجو کنیم، همه‌ی توالی‌هایی از کاراکتر a به طول بزرگ‌تر یا مساوی یک را می‌یابد و آن‌ها را با b جایگزین می‌کند. لذا در این حالت، عبارت aaa تبدیل به b می‌شود؛ چرا که عبارت اوّلیه تنها یک تحقق از چیزی بود که به دنبال آن گشته بودیم، و نه سه تحقق از آن.

مطابق توضیح فوق، آن خط از کد که مشغول توضیح آن بودیم، همه‌ی توالی‌های با طول بزرگ‌تر یا مساوی یک از فاصله (space) را با یک فاصله جایگزین می‌کند، تا مواردی که در میان جملات و عناوین، فاصله‌های بی‌مورد استفاده شده است؛ از بین بروند. علّت آن که این کد را اضافه کردیم، آن بود که چنین مواردی را در میان عناوین نامه‌ها مشاهده کرده بودیم؛ در حالی که چنین مشکلی در مورد خطبه‌ها نبود.

در نهایت یک عبارت دیگر نیز در قطعه کد این بخش مشاهده می‌شود، و آن نیز یک عبارت شرطی با دستور if است که بررسی می‌کند که آیا عنوان مدّ نظر، در نام خود کلمه‌ی «نامه» را به یک فاصله قبل و یک فاصله بعد از خود دارد یا نه. علّت قرار دادن این عبارت نیز آن است که در یکی از خطبه‌ها، کلمه‌ی «برنامه» وجود داشت؛ و چون ما به دنبال

کلمات «نامه» گشته بودیم، این خطبه نیز در میان نامه‌ها انتخاب شده بود. با این حال، این شرط باعث می‌شود این خطبه از لیست نامه‌ها حذف شود و نهایتاً لیستی که از نامه‌ها به دست می‌آید؛ دقیقاً همان چیزی است که مورد انتظار است و این بخش نیز کامل می‌شود.

## ۴.۵.۲ تکمیل فهرست نام خطبه‌ها و نامه‌ها در پایگاه داده

```
import sqlite3
conn = sqlite3.connect('Nahj.db')
c = conn.cursor()
count = 0
sub_count = 0
for text in lst:
    count += 1
    sub_count += 1
    c.execute(f'INSERT INTO list VALUES ({count}, "khotbe", {sub_count},
        "{text}") ')

sub_count = 0
for text in lst:
    count += 1
    sub_count += 1
    c.execute(f'INSERT INTO list VALUES ({count}, "nameh", {sub_count},
        "{text}") ')

conn.commit()
conn.close()
```

تا به این جا، فهرست نام خطبه‌ها و نامه‌ها در دو لیست به نام‌های `lst` و `lst2` ذخیره شده‌اند. مرحله‌ی بعدی آن است که آن‌ها را در جدول `list` در دیتابیس اصلی ذخیره کنیم. برای این کار، مجدداً به سراغ کتابخانه‌ی `sqlite3` می‌رویم و با ایجاد یک اتصال و یک مکان‌نما، شروع به نوشتن در دیتابیس می‌کنیم.

ابتدا به کمک یک حلقه‌ی `for`، روی لیست `lst` که نام خطبه‌ها در آن ذخیره شده است، پیمایش می‌کنیم. در این

جا از دو متغیر با نام‌های `count` و `sub_count` نیز استفاده شده است. یکی از این دو متغیر، همه‌ی عناوینی را که در دیتابیس قرار می‌گیرند می‌شمارد و دیگری، خطبه‌ها را می‌شمارد و هنگامی که خطبه‌ها تمام شوند و به نامه‌ها برسیم، این متغیر دوباره از نو شروع می‌کند.

برای تکمیل کار، از متد `execute` که پیش‌تر در بخش ۱.۵.۲ معرفی شده بود، استفاده می‌کنیم. برای آن که بتوانیم یک عبارت کلّی در داخل حلقه‌ی `for` بنویسیم که هر یک از عناوین را به درستی در دیتابیس جایگذاری کند، از روشی در پایتون استفاده می‌کنیم که اصطلاحاً به `f-string` معروف است. رشته‌هایی که با این روش تشکیل می‌شوند، ترکیبی از یک رشته‌ی ازپیش‌تعیین‌شده و یک یا چند متغیر هستند. به عنوان مثال، فرض کنید یک متغیر به اسم `x` داریم که مقدار آن برابر ۲ است و می‌خواهیم در یک رشته، مقدار آن را اعلام کنیم. اگر عبارت زیر را در پایتون چاپ کنیم:

```
f'The value in x is {x}'
```

چیزی که روی صفحه‌ی نمایش چاپ می‌شود به شکل زیر خواهد بود:

```
The value in x is 2
```

و این دقیقاً همان چیزی است که به آن نیاز داشتیم. به عبارت دیگر، در ساختار `f-string`ها، یک حرف `f` قبل از رشته قرار می‌گیرد و در داخل رشته، هر گاه بخواهیم به مقدار یک متغیر اشاره کنیم، آن متغیر را داخل یک جفت `{ }` قرار می‌دهیم.

آنچه توضیح داده شد، دقیقاً همان کاری است که برای وارد کردن اطلاعات در دیتابیس کرده‌ایم. با اجرای کد

```
c.execute(f'INSERT INTO list VALUES ({count}, "khotbe", {sub_count},
"{text}")')
```

مقادیر متغیرهای `count` و `sub_count` در فیلدهای `ID` و `catID` قرار گرفته، در فیلد `category` کلمه‌ی `khotbe` قرار می‌گیرد، و نهایتاً در فیلد `title` نام خطبه قرار داده می‌شود. با اتمام این حلقه‌ی `for`، اسامی همه‌ی خطبه‌ها همراه با شماره‌گذاری کلّی و نیز شماره‌ی خطبه (که در این جا یکی هستند، چرا که خطبه‌ها اوّلین گروهی هستند که وارد دیتابیس شده‌اند و اوّلین خطبه، اوّلین ورودی دیتابیس نیز هست) وارد پایگاه داده می‌شوند.

در ادامه، حلقه‌ی `for` مشابهی را برای نامه‌ها نیز اجرا می‌کنیم و در میان این دو حلقه، تنها کاری که لازم است انجام شود آن است که مقدار متغیر `sub_count` را مجدداً برابر صفر قرار دهیم، چرا که این متغیر قرار است شماره‌ی نامه‌ها را اعلام کند و باید دوباره از نو شروع به شماره‌گذاری کند.

در نهایت همان‌طور که در بخش ۱.۵.۲ بیان شد، برای آن که تغییرات در دیتابیس ثبت شوند از متد `commit` استفاده کرده و نهایتاً هنگامی که کار دیگری با دیتابیس نداریم، با استفاده از دستور `conn.close()` ارتباط برنامه را با دیتابیس قطع می‌کنیم.

در این جا جدول list در دیتابیس که فهرست عناوین خطبه‌ها و نامه‌ها است، کامل می‌شود. نکته‌ی دیگری که لازم است ذکر شود آن است که همان طور که پیش‌تر بیان شد، در نهج البلاغه علاوه بر خطبه‌ها و نامه‌ها، گونه‌ی سومی نیز وجود دارد و آن، حکمت‌ها هستند. با این حال، به دلیل آن که هر یک از حکمت‌ها در حدّ یک یا چند جمله‌ی کوتاه هستند؛ نامی جداگانه برای هر حکمت در نظر گرفته نشده و به همین دلیل، آن‌ها را به صورت جداگانه در جدول list ذکر نکرده‌ایم. در ادامه و پس از استخراج محتوای هر یک از خطبه‌ها و نامه‌ها، به سراغ حکمت‌ها نیز خواهیم رفت.

## ۵.۵.۲ ایجاد جدول خطبه‌ها

```
import sqlite3
conn = sqlite3.connect('Nahj.db')
c = conn.cursor()

c.execute('CREATE TABLE khotbe (ID integer, khotbeID integer, partID
integer, text, translation)')

conn.commit()
conn.close()
```


تا به این جا، دیتابیس ما تنها یک جدول به نام list دارد که به نوعی، فهرست عناوین خطبه‌ها و نامه‌های نهج البلاغه است؛ با این حال، ما هنوز متن هیچ‌یک از خطبه‌ها یا نامه‌ها را استخراج نکرده‌ایم و تنها عناوین آن‌ها را به دست آورده‌ایم. در ادامه، می‌خواهیم متن‌ها را نیز از سایت مورد نظر استخراج کنیم. ابتدا، یک جدول دیگر با نام khotbe به دیتابیس اضافه می‌کنیم. ساختار کد مورد استفاده دقیقاً مانند قسمت ۱.۵.۲ است، لذا از توضیح دوباره‌ی آن پرهیز می‌کنیم. مطابق کد فوق، مشاهده می‌شود که این جدول پنج فیلد دارد:

- ID: همان شماره‌ی کلی است که همه‌ی بخش‌های همه‌ی خطبه‌ها را می‌شمارد. به عبارت دیگر، این شماره روی تمامی سطرهای جدول khotbe به ترتیب و یک به یک زیاد می‌شود.


- khotbeID: همان شماره‌ی خطبه است و متناظر با فیلد catID برای خطبه‌ها در جدول list است.

- partID: با توجه به این که خطبه‌ها به نسبت طولانی هستند، هر خطبه به بخش‌های کوچک شکسته می‌شود و سپس در دیتابیس قرار می‌گیرد. رابطه‌ی هر خطبه و بخش‌های آن، مانند رابطه‌ی سوره‌های قرآن کریم با آیه‌های آن است. با توجه به این که این بخش‌بندی یکتا نیست، از بخش‌بندی موجود در خود سایت مرجع استفاده می‌کنیم. به عنوان مثال در شکل ۳.۲ بخشی از خطبه‌ی اوّل نهج البلاغه که در این سایت موجود است را مشاهده می‌کنید. مطابق این شکل، در هر خط بخشی از خطبه نوشته شده و در ادامه، ترجمه‌ی آن بخش قابل مشاهده است. پس از


آن، به بخش بعدی می‌رسیم و این روال تا انتهای خطبه ادامه دارد. همین بخش‌بندی را برای قرار دادن خطبه در دیتابیس استفاده می‌کنیم و فیلد partID شماره‌ی بخش را مشخص می‌کند که به وضوح، با رسیدن به خطبه‌ی جدید، این فیلد مجدداً از یک شروع به شمارش می‌نماید.





صفحه اصلی < نهج البلاغه (ترجمه استاد حسین انصاریان) < خطبه ۱



## خطبه ۱ - خطبه در ابتدای آفرینش آسمان و زمین و آدم




 متن عربی


 متن ترجمه

### فَمِنْ خُطْبَةٍ لَهُ عَلَيْهِ السَّلَامُ

از خطبه‌های آن حضرت است

## يَذْكُرُ فِيهَا ابْتِدَاءَ خَلْقِ السَّمَاءِ وَالْأَرْضِ وَخَلْقِ آدَمَ، وَفِيهَا ذِكْرُ الْحَجِّ

که در آن ابتدای آفرینش آسمان و زمین و آدم را توضیح می‌دهد و یادی از حج می‌کند

## الْحَمْدُ لِلَّهِ الَّذِي لَا يَبْلُغُ مَدْحَتَهُ الْقَائِلُونَ، وَلَا يُحْصَى نِعَمَاهُ

خدای را سپاس که گویندگان به عرصه ستایشش نمی‌رسند، و شماره گران از عهده شمردن

## الْعَادُونَ، وَلَا يُؤَدِّي حَقَّهُ الْمُجْتَهِدُونَ، الَّذِي لَا يُدْرِكُهُ بَعْدُ الْهَمَمُ،

نعمتهایش برنمایند، و کوشندگان حقش را ادا نکنند، خدایی که اندیشه‌های بلند او را درک ننمایند،

## وَلَا يَنَالُهُ غَوْصُ الْفِطْنِ، الَّذِي لَيْسَ لِصِفَتِهِ حَدٌّ مُحَدَّدٌ، وَلَا نَعْتُ

و هوش‌های ژرف به حقیقتش دست نیابند، خدایی که اوصافش در چهارچوب حدود نگنجد، و به ظرف وصف

شکل ۳.۲: بخشی از خطبه‌ی اوّل نهج البلاغه موجود در آدرس <http://www.erfan.ir/farsi/nahj1-1/>

• text: این فیلد حاوی متن عربی هر بخش از خطبه می‌باشد.

• translation: این فیلد حاوی ترجمه‌ی هر بخش از خطبه می‌باشد.

جدول khotbe با پنج فیلد مذکور در این بخش ساخته می‌شود. در بخش‌های بعدی به پر کردن این جدول

می‌پردازیم.



## ۶.۵.۲ یافتن لیست آدرس صفحات خطبه‌ها

```

address = []
for i in range(len(lst)):
    if i < 231:
        t = f'خطبه-{i+1}-'+'.join(lst[i].split())
        t = f'http://www.erfan.ir/farsi/nahj1-{i+1}/' + t
    else:
        t = f'خطبه-{i}-'+'.join(lst[i].split())
        t = f'http://www.erfan.ir/farsi/nahj1-{i}/' + t
    address.append(t)

```

به خاطر داریم که در حال حاضر، هدف آن است که محتوای همه‌ی خطبه‌ها را به دست آوریم. می‌دانیم که آدرس صفحه‌ی وب مربوط به هر خطبه با خطبه‌های دیگر متفاوت است. برای آن که بتوانیم اطلاعات خطبه‌ها را بیابیم، لازم است که آدرس تمامی صفحات خطبه‌ها را داشته باشیم. ابتدا به صورت دستی آدرس صفحه‌ی چند مورد از خطبه‌ها را مشاهده می‌کنیم تا بتوانیم الگوی کلی نام‌گذاری این صفحات را بیابیم. با اندکی مشاهده به این نتیجه می‌رسیم که نام‌گذاری این صفحات به این صورت است که ابتدای آدرس به صورت `http://www.erfan.ir/farsi/nahj1-n/` است که در آن، `n` شماره‌ی خطبه است. در ادامه‌ی این آدرس، نام خطبه به فارسی می‌آید، به طوری که کلمات آن با کاراکتر - از یک‌دیگر جدا شده‌اند. به عنوان مثال، آدرس صفحه‌ی اولین خطبه به صورت زیر است:

خطبه-۱- خطبه-در-ابتدای-آفرینش-آسمان-و-زمین-و-آدم `http://www.erfan.ir/farsi/nahj1-1/`

کد این بخش، وظیفه‌ی آن را به عهده دارد تا در لیستی به نام `address`، آدرس تمامی صفحات مربوط به خطبه‌ها را با الگوی فوق بسازد و ذخیره کند، تا در مراحل بعدی، بتوانیم به ترتیب به هر یک از این آدرس‌ها مراجعه کنیم و محتوای خطبه‌ی متناظر را دریافت نمائیم.

حال به توضیح کد فوق می‌پردازیم. عبارت `lst[i].split()`، عضو `i`ام لیست `lst` را در محل‌هایی که کاراکتر فاصله قرار دارد، جدا می‌کند و آن را تبدیل به لیستی از چند کلمه می‌کند. به عبارت دیگر، متد `split` رشته‌هایی از کلمات را که با فاصله از یک‌دیگر جدا شده‌اند، تبدیل به کلمات مجزایی می‌کند که در یک لیست کنار هم قرار گرفته‌اند. به این ترتیب می‌توان به هر یک از این کلمات به صورت جداگانه دسترسی یافت. در ادامه، دستور `join(lst[i].split())` آن کلمات را با کاراکتر - به یک‌دیگر متصل می‌کند و بار دیگر یک رشته می‌سازد. بنابراین تا به این جا، کاری که انجام شده آن است که نام خطبه‌ها که رشته‌هایی از کلمات بودند که با فاصله از یک‌دیگر جدا شده بودند، تبدیل به همان رشته‌ها شدند با این تفاوت که با - از یک‌دیگر جدا می‌شوند. علت این امر آن است که

در آدرس‌دهی صفحات، لازم است از چنین الگویی استفاده شود.

در ادامه شماره‌ی خطبه همراه با خود کلمه‌ی «خطبه» نیز به این رشته‌ها افزوده می‌شود و در آخرین مرحله، به ابتدای هر یک از آن‌ها، آدرس `http://www.erfan.ir/farsi/nahj1-n/` نیز افزوده می‌شود که منظور از `n`، شماره‌ی خطبه است.

این کار باعث می‌شود که آدرس صفحه‌ی هر یک از خطبه‌ها دقیقاً با همان الگویی که نام‌گذاری این صفحات در سایت انجام شده به دست آید؛ تا در ادامه بتوانیم با مراجعه به هر یک از این آدرس‌ها، به محتوای خطبه‌ها دست پیدا کنیم. شایان ذکر است که در ساختار این کد از روش آدرس‌دهی `f-string` استفاده شده که توضیحات مربوط به آن در بخش ۴.۵.۲ بیان شده است.

تنها یک نکته‌ی دیگر در این کد نیاز به توضیح دارد، و آن نیز عبارت `if/else` ای است که در آن موجود است. علت آن که مجبور شدیم از چنین ساختاری استفاده کنیم، آن بود که در نام‌گذاری آدرس صفحات خطبه‌ها در خطبه‌ی شماره‌ی ۲۳۱ اشتباهی رخ داده بود و در واقع در این خطبه، شماره‌گذاری‌های آدرس‌های صفحات یکی جابه‌جا می‌شوند. این اشتباه از طرف طراح سایت است و ما برای آن که بتوانیم کد خود را در برابر آن مقاوم سازیم، آن را برای خطبه‌های قبل و بعد از این شماره جدا کرده‌ایم.

نهایتاً با این کار، آدرس صفحات تمامی خطبه‌ها به دست می‌آیند و با متد `append` در لیست `address` اضافه می‌شوند. با اجرای این قطعه کد، لیست `address` حاوی آدرس تمامی صفحات خطبه‌ها خواهد بود. در ادامه با استفاده از این لیست به تک‌تک این صفحات مراجعه خواهیم کرد و متن و ترجمه‌ی خطبه‌ها را دریافت خواهیم نمود.

## ۷.۵.۲ استخراج و ذخیره‌سازی متن عربی خطبه‌ها

```
conn = sqlite3.connect('Nahj.db')
c = conn.cursor()
count = 0
sub_count = 0
for khotbe in range(len(address)):
    print(f'{khotbe} out of {len(address)}')
    req = requests.get(address[khotbe])
    soup = BeautifulSoup(req.text, 'html.parser')
    x = soup.find_all('div', attrs ={'class': ['MyArText', 'js_MafatidText', 'MafatidText']})
    sub_count = 0
```

```

for line in x:
    print(line.text.strip())
    count += 1
    sub_count += 1
    c.execute(f'INSERT INTO khotbe VALUES ({count}, {khotbe+1}, {
        sub_count}, "{line.text.strip()}", "")')
    conn.commit()

```

در این بخش، شروع به کامل کردن جدول khotbe در دیتابیس خود می‌کنیم. برای این کار، به آدرس هر یک از خطبه‌ها در سایت مورد نظر مراجعه می‌کنیم و متن عربی آن خطبه را دریافت می‌نماییم. سپس محتوای آن را در دیتابیس وارد می‌کنیم.

حلقه‌ی for بیرونی که در کد مشاهده می‌شود، وظیفه‌ی پیمایش روی خطبه‌ها را دارد. هر بار با استفاده از کتابخانه‌ی requests که در بخش ۱.۳.۲ به توضیح آن پرداخته شد، کد HTML صفحه‌ی مربوط به یکی از خطبه‌ها را دریافت می‌نماییم. در ادامه به کمک کتابخانه‌ی BeautifulSoup که در بخش ۲.۳.۲ مورد بررسی قرار گرفت، در این کد به جست‌وجو می‌پردازیم. هدف از این جست‌وجو، استخراج بخش به بخش متن عربی این خطبه است. منظور از هر بخش از خطبه را در قسمت ۵.۵.۲ به تفصیل بیان کردیم.

همان طور که در قسمت ۲.۵.۲ توضیح داده شد، برای آن که بفهمیم چگونه باید به دنبال هدف مورد نظر خود در کد HTML بگردیم؛ باید ابتدا صفحه‌ی وب مورد نظر را واریسی نماییم. پس از انجام این کار متوجه می‌شویم که متون عربی بخش‌های هر خطبه، دارای تگ div بوده و ویژگی class آن‌ها به صورت 'MyArText', ['js\_MafatidText', 'MafatidText'] است. بنابراین دقیقاً همین یافته‌ها را با استفاده از متد find\_all مورد جست‌وجو قرار می‌دهیم و نتایج را در متغیری به نام x ذخیره می‌کنیم.

در این جا به دومین حلقه‌ی for می‌رسیم که داخل حلقه‌ی اوّل قرار دارد. این حلقه روی نتایج حاصل از جست‌وجوی اخیر پیمایش می‌کند، یا به عبارت دیگر در هر بار گردش این حلقه، به متن عربی یکی از بخش‌های خطبه‌ی فعلی دسترسی می‌یابیم. (به خاطر داریم که حلقه‌ی for بیرونی نیز دارد روی خطبه‌ها پیمایش می‌کند.)

در این جا متغیر sub\_count در حال شمارش بخش‌های هر خطبه می‌باشد. بنابراین درون حلقه‌ی داخلی (که روی بخش‌ها جلو می‌رود) یکی یکی زیاد می‌شود و هرگاه خطبه عوض می‌شود (داخل حلقه‌ی بیرونی و خارج حلقه‌ی داخلی) مقدار آن دوباره برابر صفر می‌شود تا بخش‌های خطبه‌ی جدید را شمارش کند.

متغیر count شمارش‌گر کلی همه‌ی بخش‌های همه‌ی خطبه‌هاست و متناظر با فیلد ID است که در بخش ۵.۵.۲ توضیح داده شد.

در نهایت به کمک متد `execute`، اطلاعات مورد نظر را در جدول `khotbe` می‌نویسیم. فیلدهای `ID` و `partID` به ترتیب مطابق توضیحات فوق با متغیرهای `count` و `sub_count` پر می‌شوند. همچنین فیلد `khotbeID` نیز با متغیر `khotbe` (که شمارش‌گر حلقه‌ی بیرونی است) پر می‌شود. (البته چون این شمارش‌گر از صفر شروع به شمردن می‌کند، برای قرار دادن مقدار آن در دیتابیس باید یکی به آن اضافه کنیم).

نهایتاً فیلد `text` نیز با محتویات حاصل از جست‌وجو که در متغیر `x` ذخیره کرده بودیم پر می‌شود.

دقت کنید که یکی از فیلدها، یعنی فیلد `translation` هنوز خالی است و ما هنوز متون ترجمه‌ها را استخراج نکرده‌ایم. این کار در قسمت بعدی انجام خواهد شد.

## ۸.۵.۲ استخراج و ذخیره‌سازی ترجمه‌ی خطبه‌ها

```
conn = sqlite3.connect('Nahj.db')
c = conn.cursor()
count = 0
sub_count = 0
for khotbe in range(len(address)):
    print(f'{khotbe} out of {len(address)}')
    req = requests.get(address[khotbe])
    soup = BeautifulSoup(req.text, 'html.parser')
    y = soup.find_all('div', attrs ={'class': ['MyFaText', '
        js_TranslateText', 'TranslateText']})
    sub_count = 0
    for line in y:
        print(line.text.strip())
        count += 1
        sub_count += 1
        c.execute(f'UPDATE khotbe SET translation = "{line.text.strip
            ()}" WHERE ID = {count}')
    conn.commit()
```

کد این بخش بسیار مشابه کد قسمت قبل است. تفاوت اصلی، آن است که این بار به جای آن که به دنبال متون عربی بگردیم، به دنبال ترجمه‌ها می‌گردیم. با واریسی صفحات سایت، مشاهده می‌کنیم که ترجمه‌های هر بخش از هر

خطبه، المان‌هایی در کد HTML هستند که تگ آن‌ها div بود و ویژگی class آن‌ها به صورت 'MyFaText', ['TranslateText', 'js\_TranslateText'] می‌باشد. این بار، به کمک متد find\_all به دنبال این المان‌ها می‌گردیم و آن‌ها را در متغیر y ذخیره می‌کنیم. از این جا به بعد نیز بسیار شبیه قسمت قبل است؛ حلقه‌ی for دیگری را روی ترجمه‌ی بخش‌های هر خطبه قرار می‌دهیم تا هر یک را درون دیتابیس قرار دهد. تنها تفاوت آن است که این بار، باید سطرهای دیتابیس را آپدیت کنیم، چرا که چهار فیلد از پنج فیلد هر سطر قبلاً پر شده و کافی است تنها ترجمه را به آن اضافه کنیم تا جدول کامل شود. با انجام این کار از طریق متد execute، جدول خطبه‌ها کامل می‌گردد.

## ۹.۵.۲ جدول نامه‌ها

پس از ساخت و تکمیل جدول خطبه‌ها، جدول دیگری در دیتابیس خود ایجاد کرده و آن را به نامه‌های نهج البلاغه اختصاص می‌دهیم. فرآیندی که برای ساخت و تکمیل این جدول پیش می‌گیریم، بسیار شبیه به کاری است که برای جدول خطبه‌ها کردیم، بنابراین دیگر به صورت جزئی به توضیح آن نمی‌پردازیم. در ابتدا این جدول را با پنج فیلد ایجاد می‌کنیم. فیلدهای این جدول نظیر به نظیر مشابه فیلدهای جدول خطبه‌ها هستند که در بخش ۵.۵.۲ مورد توضیح و بررسی قرار گرفتند.

پس از ایجاد جدول، به سراغ پر کردن آن می‌رویم. یک نکته که در این بخش از کد نسبت به قسمت قبلی متفاوت است، آن است که برای ایجاد لیستی از آدرس صفحات مربوط به نامه‌ها (که نمونه‌ی مشابه آن برای خطبه‌ها در قسمت ۶.۵.۲ بررسی شد) دیگر به اندازه‌ی قسمت ۶.۵.۲ کار را سخت و پیچیده نمی‌کنیم و نام فارسی نامه‌ها را در آدرس صفحات قرار نمی‌دهیم. در واقع کافی است برای دستیابی به صفحه‌ی مربوط به نامه‌ی nام، به آدرس <http://www.erfan.ir/farsi/nahj2-n> مراجعه کنیم. لازم به ذکر است که می‌توانستیم این کار را برای خطبه‌ها نیز انجام دهیم؛ اما زمانی که کد آن بخش از برنامه را می‌نوشتیم، هنوز به وجود چنین امکانی پی نبرده بودیم؛ لذا مسیر سخت‌تری را برای دستیابی به آدرس صفحات خطبه‌ها پیمودیم (در واقع نکته‌ای که پیش‌تر نمی‌دانستیم آن بود که اگر آدرس صفحات را به صورت ناقص و بدون متن فارسی آن وارد کنیم هم صفحه به درستی لود می‌شود).

نهایتاً کدهای مربوط به یافتن متون عربی و ترجمه‌های نامه‌ها نیز کاملاً مشابه قسمت‌های متناظر مربوط به خطبه‌ها (یعنی قسمت‌های ۷.۵.۲ و ۸.۵.۲) می‌باشد و تنها تفاوت میان آن‌ها، در آدرس صفحه‌ی وبی است که برای جست‌وجو بارگذاری می‌شود؛ بنابراین دیگر از ارائه‌ی توضیحات تکراری پرهیز می‌کنیم. در صورت نیاز، می‌توانید برای بررسی توضیحات به بخش‌های ۷.۵.۲ و ۸.۵.۲ مراجعه نمایید.

کدهای مربوط به ساخت و تکمیل جدول نامه‌ها در ادامه قابل مشاهده هستند.

```
import sqlite3

conn = sqlite3.connect('Nahj.db')

c = conn.cursor()
```

```
c.execute('CREATE TABLE nameh (ID integer, namehID integer, partID integer,
    text, translation)')

conn.commit()

conn.close()
```

```
conn = sqlite3.connect('Nahj.db')
c = conn.cursor()
count = 0
sub_count = 0
N = 79
url = 'http://www.erfan.ir/farsi/nahj2-'
for nameh in range(N):
    print(f'{nameh} out of {N}')
    req = requests.get(url+str(nameh+1))
    soup = BeautifulSoup(req.text, 'html.parser')
    x = soup.find_all('div', attrs ={'class': ['MyArText', '
        js_MafatidText', 'MafatidText']})
    sub_count = 0
    for line in x:
        print(line.text.strip())
        count += 1
        sub_count += 1
        c.execute(f'INSERT INTO nameh VALUES ({count}, {nameh+1}, {
            sub_count}, "{line.text.strip()}", "")')
    conn.commit()
```

```
conn = sqlite3.connect('Nahj.db')
c = conn.cursor()
count = 0
sub_count = 0
```

```

N = 79
url = 'http://www.erfan.ir/farsi/nahj2-'
for nameh in range(N):
    print(f'{nameh} out of {N}')
    req = requests.get(url+str(nameh+1))
    soup = BeautifulSoup(req.text, 'html.parser')
    y = soup.find_all('div', attrs ={'class': ['MyFaText', '
        js_TranslateText', 'TranslateText']})
    sub_count = 0
    for line in y:
        print(line.text.strip())
        count += 1
        sub_count += 1
        c.execute(f'UPDATE nameh SET translation = "{line.text.strip
            ()}" WHERE ID = {count}')
        conn.commit()

```

## ۱۰.۵.۲ استخراج متن عربی حکمت‌های نهج البلاغه

همان‌طور که پیش‌تر گفته شد، حکمت‌های نهج البلاغه اغلب بسیار کوتاه و در حدّ یک تا چند جمله هستند؛ لذا بر خلاف خطبه‌ها و نامه‌ها اسامی مجزّا ندارند و صرفاً به ترتیب شماره‌گذاری می‌شوند. همچنین به همین دلیل، این حکمت‌ها در صفحات وب مختلف نیز وجود ندارند، و تنها در یک صفحه به آدرس <http://www.erfan.ir/farsi/nahj3-1> می‌توان تمامی آن‌ها را یافت. این مسأله کار ما را برای استخراج آن‌ها ساده‌تر می‌کند و دیگر لازم نیست آدرس تمامی صفحات آن‌ها را بیابیم.

در ادامه کد مربوط به استخراج متن عربی این حکمت‌ها را مشاهده می‌نمایید:

```

url = 'http://www.erfan.ir/farsi/nahj3-1'
req = requests.get(url)
soup = BeautifulSoup(req.text, 'html.parser')
x = soup.find_all('div', attrs ={'class': ['MyArText', 'js_MafatidText', '
    MafatidText']})

```

```

hek_list = []
count = 0
for line in x:
    match = re.match('\d+\s*-*', line.text.strip())
    if type(match).__name__ != 'NoneType':
        count += 1
        hek_list.append((count, line.text.strip().strip(match.group(0))))
    else:
        hek_list.append((count, line.text.strip()))

```

در ابتدا محتوای صفحه‌ی حاوی حکمت‌ها را با استفاده از کتاب‌خانه‌ی requests دریافت می‌کنیم. سپس مطابق توضیحاتی که به صورت متعّدّ پیش‌تر در این گزارش مورد اشاره قرار گرفته‌اند، به واریسی صفحه‌ی وب مربوطه می‌پردازیم و متوجّه می‌شویم که متون عربی حکمت‌ها از نظر تگ و ویژگی class در کد HTML دقیقاً مشابه متون عربی خطبه‌ها و نامه‌ها هستند؛ بنابراین مشابه قبل، به کمک متد find\_all این متون عربی را استخراج کرده و در متغیّری به نام x ذخیره می‌کنیم. در این جا اگر محتویات متغیّر x را چاپ کنیم، لیستی طولانی از متون عربی حکمت‌ها را مشاهده می‌کنیم. بخشی از این لیست را به عنوان نمونه در شکل ۴.۲ ملاحظه می‌نمایید.

با مشاهده‌ی شکل ۴.۲، واضح است که هر یک از حکمت‌ها بسته به این که چه میزان طولانی یا کوتاه باشد، به چند بخش تقسیم شده است. (هر یک از المان‌های حاصل از جست‌وجو، در یک خط در شکل ۴.۲ چاپ شده است، بنابراین به عنوان مثال حکمت شماره ۳ به دو بخش تقسیم شده، در حالی که حکمت شماره ۶ به ۵ بخش تقسیم شده است.)

همچنین الگوی مهمّی که با مشاهده‌ی شکل ۴.۲ به دست می‌آید، آن است که هرگاه حکمت جدیدی شروع می‌شود، در نتایج جست‌وجو شاهد شماره‌ی آن حکمت هستیم که با تعداد زیادی کاراکتر فاصله (space) از متن حکمت جدا شده است. باید از این الگو استفاده کنیم تا بتوانیم حکمت‌ها را از یک‌دیگر جدا کنیم و در دیتابیس خود، آن‌ها را به تفکیک شماره‌ی حکمت ذخیره کنیم.

برای این منظور، لازم است روی بخش‌های مختلف حکمت‌ها (عناصر مختلف متغیّر x) یکی یکی پیمایش کنیم و ببینیم که آیا این بخش، بخش اوّل یک حکمت جدید است (یعنی در ابتدای خود عدد دارد) یا ادامه‌ی بخش قبلی است و وارد حکمت جدیدی نشده است. همان طور که پیش‌تر بارها به regexها اشاره شد، برای انجام این جست‌وجو نیز از regexها کمک می‌گیریم. به دنبال ساختاری می‌گردیم که در ابتدای آن کاراکتر عددی باشد (در ادبیات regexها، عدد را با \d نشان می‌دهیم). پس از نماد کاراکتر عددی، نماد + قرار داده شده است. در مورد این نماد در بخش ۳.۵.۲



3	و قَالَ عَلَيْهِ السَّلَامُ: - الْبُخْلُ عَارٌ، وَ الْجُبْنُ مُنْقَصَةٌ، وَالْقَفْرُ - يُخْرِشُ الْقَطَنَ عَنْ حُجَّتِهِ، وَ الْفَقْلُ غَرِيبٌ فِي بَلَدِيهِ.
4	و قَالَ عَلَيْهِ السَّلَامُ: - الْعَجْزُ آفَةٌ، وَ الصَّبْرُ شَجَاعَةٌ، وَ الزُّهْدُ - تَرَوْهُ، وَ الْوَرَعُ خُبَّةٌ، وَ يَغْمُ الْقَرِينُ الرِّضَى.
5	و قَالَ عَلَيْهِ السَّلَامُ: - الْعِلْمُ وَرَاءُ كَرِيمَةٍ، وَ الْأَدَابُ خُلٌّ - مُجَدِّدَةٌ، وَ الْفِكْرُ مِرْآةٌ صَافِيَةٌ.
6	و قَالَ عَلَيْهِ السَّلَامُ: - صَدْرُ الْعَاقِلِ مُنْدُوقٌ سِرِّهِ، وَ الْبَشَاشَةُ - جِبَالَةُ الْمَوَدَّةِ، وَ الْأَحْتِمَالُ قَبْرُ الْغُيُوبِ. -: وَ رَوَى أَنَّهُ عَلَيْهِ السَّلَامُ قَالَ فِي الْعِبَارَةِ عَنْ هَذَا الْمَعْنَى أَيْضاً - وَالْمُسَالَمَةُ جِبَاءُ الْغُيُوبِ، وَ مَنْ رَضِيَ عَنْ نَفْسِهِ كَثُرَ السَّاجِدُ عَلَيْهِ.
7	و قَالَ عَلَيْهِ السَّلَامُ: - الصَّدَقَةُ دَوَاءٌ مُنْجِحٌ، وَ أَعْمَالُ الْعِبَادِ - فِي عَاجِلِهِمْ نُصَبُ أَعْيُنِهِمْ فِي آجِلِهِمْ.
8	و قَالَ عَلَيْهِ السَّلَامُ: - إِعْجَبُوا لِهَذَا الْإِنْسَانِ، يَنْظُرُ بِشَحْمٍ - وَيَتَكَلَّمُ بِلَحْمٍ، وَ يَسْمَعُ بِعَظْمٍ، وَ يَتَنَفَّسُ مِنْ حَرَمٍ.
9	و قَالَ عَلَيْهِ السَّلَامُ: - إِذَا أَقْبَلَتِ الدُّنْيَا عَلَى قَوْمٍ أَعَارَتْهُمْ -

شکل ۴.۲: بخشی از محتوای متون عربی حکمت‌ها، بلافاصله پس از استخراج از کد HTML و بدون پردازش‌های بعدی

به تفصیل توضیح دادیم و گفتیم که منظور از آن، این است که کارکتر قبلی (که در این جا هر کاراکتر عددی است) به تعداد یک یا بیشتر از آن تکرار شده باشد (زیرا ممکن است عدد یک‌رقمی یا دورقمی باشد). تا به این جا، regex مربوطه اعداد یک یا چندرقمی را شناسایی می‌کند. پس از آن، نماد \s قرار داده شده که نماد فاصله است و بعد از آن نماد \* که مانند + است با این تفاوت که به جای آن که تعداد را بزرگ‌تر یا مساوی یک در نظر بگیرد، آن را بزرگ‌تر یا مساوی صفر در نظر می‌گیرد. بنابراین تا به این جا الگوهایی را تعریف کرده‌ایم که در ابتدای آن یک عدد است، و پس از آن تعدادی فاصله (که ممکن است تعداد آن صفر نیز باشد) قرار دارد. در ادامه نیز نماد \*- قرار دارد که یعنی تعدادی (صفر یا بیشتر) نماد -. پس توانستیم ساختاری را که در شکل ۴.۲ برای بخش‌های شروع‌کننده‌ی حکمت جدید مشاهده می‌شود، به زبان regexها بیان کنیم؛ یعنی ساختاری که با عدد شروع می‌شود، تعدادی فاصله وجود دارد، و بعد از آن نیز نماد -

نهایتاً با اجرای کد فوق، متغیر `hek_list` کامل می‌شود و همه‌ی حکمت‌های نهج‌البلاغه همراه با شماره‌ی حکمت‌ها در آن ذخیره می‌شوند. بخشی از نتیجه‌ی این لیست در شکل ۵.۲ قابل مشاهده است. همان طور که این شکل نشان می‌دهد، حکمت‌ها بسته به طول آن‌ها به چند بخش تقسیم شده‌اند، و شماره‌ی هر حکمت در کنار تمامی بخش‌های آن وجود دارد. مرحله‌ی بعدی آن است که ترجمه‌های این متون را نیز بیابیم و نهایتاً همگی را در دیتابیس ذخیره کنیم.

- (1, 'كُنْ فِي الْفِتْنَةِ كَأَبْنِ الْبُؤْسِ; لَا تَهْزُقْ فِيزَكَبْ :\xa0-\xa0السَّلَامُ قَالَ عَلَيْهِ \xa0')  
(1, 'وَلَا تَصْرَعْ فَنُحَلِّتْ')  
(2, 'أَرَى يَنْفَسِيهِ مَنِ اسْتَشَعَرَ الطَّمَعِ :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(2, 'أَوْ رَضِيَ بِالذَّلِّ مَنْ كَشَفَ عَنْ صُرِّهِ، وَ هَاتَتْ عَلَيْهِ نَفْسُهُ')  
(2, 'مَنْ أَهْرَ عَلَيْهَا لِسَانَهُ')  
(3, 'الْبُخْلُ عَارٌ، وَ الْجُبْنُ مَنَقَصَةٌ، وَالْقَفْرُ :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(3, 'يُخْرِشُ الْقَطِيقَ عَنْ حَجَّتِهِ، وَ الْفِعْلُ غَرِيبٌ فِي بَلَدِيهِ')  
(4, 'الْعَجْزُ آفَةٌ، وَ الصَّبْرُ شَجَاعَةٌ، وَ الزَّفْدُ :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(4, 'نَرُوهُ، وَ الْوَرَعُ حَيَّةٌ، وَ يَغْمُ الْقَرِيبُ الرِّصَى')  
(5, 'الْعِلْمُ وَرَاءَةُ كَرِيْمَةٍ، وَ الْأَدَابُ خُلِّل :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(5, 'مُحَدَّدَةٌ، وَ الْفِكْرُ مِرَاةٌ صَافِيَةٌ')  
(6, 'صَدْرُ الْعَاقِلِ صُنْدُوقُ سِرِّهِ، وَ الْبَشَاشَةُ :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(6, 'وَ الْأَحْتِمَالُ قَبْرِ الْعُيُوبِ، \xa0-\xa0جِبَالَةُ الْقَوَدَةِ')  
(6, ':- الْعِبَارَةُ عَنْ هَذَا الْمَعْنَى أَيْضًا \xa0-\xa0السَّلَامُ قَالَ فِي \xa0-\xa0وَ رَوَى أَنَّهُ عَلَيْهِ \xa0-\xa0')  
(6, 'وَ الْفُسَالَةُ حِبَاءُ الْعُيُوبِ، وَ مَنْ رَضِيَ عَنْ نَفْسِهِ كَثُرَ')  
(6, 'السَّخَاطُ عَلَيْهِ')  
(7, 'الصَّدَقَةُ دَوَاءٌ مُنْجِحٌ، وَ أَعْمَالُ الْعِبَادِ :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(7, 'فِي عَاجِلِهِمْ نُصِبَ أَعْيُنُهُمْ فِي آجِلِهِمْ')  
(8, 'إِعْجَبُوا لِهَذَا الْإِنْسَانِ، يَنْطُرُ يَسْتَحِمُ :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(8, 'وَ يَتَكَلَّمُ بِلُحْمٍ، وَ يَسْمَعُ بِعَظْمٍ، وَ يَنْتَفِسُ مِنْ خَرْمٍ')  
(9, 'إِذَا أَقْبَلَتِ الدُّنْيَا عَلَى قَوْمٍ أَعَارَتْهُمْ :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(9, 'مَحَاسِنَ غَيْرِهِمْ، وَ إِذَا أَدْبَرَتْ عَنْهُمْ سَلَبَتْهُمْ مَحَاسِنَ')  
(9, 'أَنْفُسِهِمْ')  
(10, 'حَاطِلُوا النَّاسِ مُحَاطَلَةٌ إِنْ مِثْمَ مَعَهَا بَكْوَا :\xa0-\xa0السَّلَامُ وَ قَالَ عَلَيْهِ \xa0')  
(10, 'عَلَيْكُمْ، وَ إِنْ عَشِمْتُمْ حَتُّوا إِلَيْكُمْ')

### ۱۱.۵.۲ استخراج ترجمه‌های حکمت‌های نهج البلاغه

```

req = requests.get(url)
soup = BeautifulSoup(req.text, 'html.parser')
x = soup.find_all('div', attrs ={'class': ['MyFaText', 'js_TranslateText',
    'TranslateText']})
hek_tar_list = []
count = 0
for line in x:
    hek_tar_list.append(line.text.strip())

```

در این بخش به استخراج ترجمه‌های حکمت‌های نهج البلاغه می‌پردازیم. برای این کار، دیگر مشکلات و سختی‌های قسمت قبل را پیش‌رو نداریم، چرا که اوّل ترجمه‌ها دیگر در ابتدای برخی بخش‌های خود مقادیر عددی ندارند، و تنها خود ترجمه‌های فارسی در سایت حاضر هستند. دومین دلیل سادگی این بخش آن است که لازم نیست در کنار ترجمه‌ی هر بخش، شماره‌ی حکمت مربوط به آن را بنویسیم، چرا که ترجمه‌ها دقیقاً به ترتیب متناظر با متون عربی هستند، و از آن جا که برای متون عربی، شماره‌ی حکمت‌ها (مطابق توضیحات قسمت قبل) موجود است، کافی است که به ترتیب ترجمه‌ها را در کنار متون عربی قرار دهیم و نتایج کاملاً سازگار خواهند بود.

برای به دست آوردن ترجمه‌ها نیز فرآیند جست‌وجو دقیقاً مشابه با فرآیندی است که در قسمت‌های ۷.۵.۲ و ۸.۵.۲ انجام شد، بنابراین لیستی از ترجمه‌ها به آسانی در متغیری به نام hek\_tar\_list ذخیره می‌شود. همچنین متد strip نیز که در هنگام ذخیره‌سازی روی نتایج جست‌وجو اعمال شده است (مطابق توضیحات قسمت قبل) برای حذف کاراکترهای فاصله‌ی اضافی از طرفین عبارات است.

## ۱۲.۵.۲ ذخیره‌ی حکمت‌های نهج البلاغه در پایگاه داده

آخرین مرحله‌ای که لازم است انجام شود، آن است که متون عربی و ترجمه‌های حکمت‌های نهج البلاغه را به پایگاه داده‌ی خود اضافه کنیم. برای این کار ابتدا مطابق کد زیر، جدولی جدید به نام hekmat در دیتابیس ایجاد می‌کنیم. این جدول ۵ فیلد دارد که دقیقاً مشابه و متناظر با فیلدهای جدول khotbe است و توضیحات مربوط به آن در بخش ۵.۵.۲ موجود است.

```

conn = sqlite3.connect('Nahj.db')
c = conn.cursor()
c.execute('CREATE TABLE hekmat (ID integer, hekmat_ID integer, part_ID ,
    text, translation)')
conn.commit()

```

```
conn.close()
```

در ادامه، لازم است اطلاعات ذخیره‌شده در دو لیست `hek_list` و `hek_tar_list` را وارد دیتابیس کنیم. کد زیر برای انجام این کار طراحی شده است.

```
conn = sqlite3.connect('Nahj.db')
c = conn.cursor()
N = len(hek_list)
partID = 1
for count in range(N):
    if count != 0 and hek_list[count][0] == hek_list[count-1][0]:
        partID += 1
    else:
        partID = 1
    c.execute(f'INSERT INTO hekmat VALUES ({count+1}, {hek_list[count]
        ][0]}, {partID}, "{hek_list[count][1]}"\'
        , "{hek_tar_list[count]}"')
    conn.commit()

conn.close()
```

با یک حلقه‌ی `for` روی تک‌تک بخش‌های همه‌ی حکمت‌ها پیمایش می‌کنیم. شمارنده‌ی این حلقه می‌تواند فیلد `ID` از دیتابیس را پر کند، چرا که این فیلد شمارنده‌ی عمومی همه‌ی بخش‌های همه‌ی حکمت‌هاست. (تنها باید به متغیر `count` که شمارنده‌ی حلقه است یک واحد اضافه کنیم، چرا که از صفر شروع به شمارش می‌کند، نه از یک.)

متغیر `partID` نیز در هر حکمت، تعداد بخش‌ها را می‌شمارد و هرگاه حکمتی تمام شود و حکمت دیگر شروع شود، این متغیر نیز دوباره به مقدار یک برمی‌گردد. نحوه‌ی بررسی این موضوع نیز با استفاده از اطلاعات متغیر `hek_list` است. از بخش ۱۰.۵.۲ به خاطر دارید که اعضای این متغیر، دوتایی‌هایی بودند که حاوی متن عربی هر بخش و شمارهی حکمت آن بخش بودند؛ بنابراین برای آن که بفهمیم یک حکمت تمام شده و حکمت دیگر آغاز شده، کافی است بررسی کنیم که شمارهی دو عضو متوالی از `hek_list` با یک‌دیگر یکسان است (دو بخش مربوط به یک حکمت) یا با هم متفاوت است (یعنی حکمت جدید شروع شده). بنابراین متغیر `partID` دقیقاً باید در فیلد `part_ID` وارد شود.

فیلد `hekmat_ID` نیز کافی است با اعدادی که در عناصر `hek_list` ذخیره شده‌اند پر شود، چرا که آن اعداد شمارهی حکمت هستند و این فیلد نیز دقیقاً به همین شماره تخصیص یافته است.

نهایتاً فیلدهای text و translation نیز به وضوح با متون عربی و ترجمه‌های آن‌ها پر می‌شوند و به این ترتیب، جدول hekmat نیز کامل می‌شود.

## ۱۳.۵.۲ جمع‌بندی

مطابق آنچه در قسمت‌های پیشین توضیح داده شد، تمامی محتوای نهج‌البلاغه، اعم از خطبه‌ها، نامه‌ها، و حکمت‌ها از سایت <http://www.erfan.ir> استخراج شدند و در یک دیتابیس ذخیره شدند. این دیتابیس حاوی چهار جدول است، جدول اوّل در واقع فهرست عناوین خطبه‌ها و نامه‌ها است، و سه جدول دیگر حاوی متون و ترجمه‌های خطبه‌ها، حکمت‌ها، و نامه‌ها هستند. دیتابیس در قالب SQLite است و با کنار هم قرار دادن کدهای بخش‌های قبل، چنین دیتابیزی به طور کامل ساخته می‌شود.

## ۶.۲ پایگاه داده‌ی صحیفه‌ی سجّادیه

استخراج اطلاعات و ساخت دیتابیس مربوط به صحیفه‌ی سجّادیه، بسیار مشابه است با آنچه در مورد نهج‌البلاغه انجام شد، و به مراتب ساده‌تر نیز هست؛ چرا که در صحیفه‌ی سجّادیه دیگر تفکیکی مانند خطبه‌ها و نامه‌های نهج‌البلاغه وجود ندارد و همه‌ی عناصر این کتاب، دعاها هستند. در ادامه مراحل ساخت این دیتابیس را بررسی می‌نماییم؛ امّا با توجه به شباهت بسیار زیاد آن به فرآیندهای طی شده برای ساخت پایگاه داده‌ی نهج‌البلاغه، بیشتر توضیحات را به قسمت‌های قبلی ارجاع می‌دهیم و به منظور جلوگیری از طولانی شدن این گزارش، از بازگویی جزئیات تکراری خودداری می‌کنیم.

### ۱.۶.۲ ایجاد پایگاه داده و جداول فهرست و دعاها

```
import sqlite3

conn = sqlite3.connect('Sahife.db')
c = conn.cursor()

c.execute("CREATE TABLE list (ID integer, title)")
c.execute("CREATE TABLE prayer (ID integer, Doa_ID integer, part_ID integer
    , text, translation)")

conn.commit()
conn.close()
```

در این بخش مشابه با آنچه در قسمت‌های قبلی بارها بیان شد، با استفاده از کتاب‌خانه‌ی sqlite3، یک دیتابیس جدید از نوع SQLite ایجاد کرده و در آن دو جدول جدید می‌سازیم؛ جدول اوّل list نام دارد که به نوعی فهرست دعاهاست؛ و جدول دوم نیز prayer نام دارد که حاوی متن عربی و ترجمه‌ی فارسی دعاها خواهد بود.

جدول list دو فیلد دارد، اوّل ID که شماره‌ی دعاست، و دوم title که نام دعای مربوطه می‌باشد.

جدول prayer دارای پنج فیلد به شرح زیر است:

- ID: شمارنده‌ی همه‌ی بخش‌های همه‌ی دعاها به صورت متوالی
- Doa\_ID: شماره‌ی دعا که متناظر با فیلد ID در جدول list است.
- part\_ID: شماره‌ی بخش در هر دعا که با ورود به دعای جدید دوباره از شماره‌ی یک شروع به شمردن می‌کند.
- text: متن عربی هر بخش از دعا
- translation: ترجمه‌ی فارسی هر بخش از دعا

همچنین لازم به ذکر است، منظور از هر بخش از هر دعا، همان چیزی است که در مورد نهج‌البلاغه نیز ذکر شد؛ یعنی هر دعا بسته به طولانی یا کوتاه بودنش، به چندین بخش تقسیم می‌شود که رابطه‌ی بخش‌ها و دعاها مانند رابطه‌ی آیات و سوره‌های قرآن است. تعیین چگونگی بخش‌بندی نیز دقیقاً به تقلید از بخش‌بندی موجود در سایت <http://www.erfan.ir> است.

## ۲.۶.۲ دریافت فهرست دعاها از صفحه‌ی وب مورد نظر

```
import requests
from bs4 import BeautifulSoup

req = requests.get('http://www.erfan.ir/farsi/sahifeh')
soup = BeautifulSoup(req.text, 'html.parser')
```

```
import re
a = soup.find_all('a', href = lambda x: x and 'دعا' in x)
count = 0
lst = []
for i in a:
    s = i.text.strip()
```

```
s = re.sub('\d','',s)
s = re.sub('\( دعا \)','',s)
lst.append(s)

lst = lst[1:]
```

این بخش دقیقاً عملکردی مشابه با بخش ۲.۵.۲ دارد؛ در ابتدا کد HTML صفحه‌ای که فهرست دعاها در آن قرار دارد (به آدرس <http://www.erfan.ir/farsi/sahifeh>) دریافت می‌شود؛ سپس المان‌هایی از این کد که تگ a دارند و ویژگی class آن‌ها دارای کلمه‌ی «دعا» است، به کمک متد find\_all استخراج می‌گردند. نهایتاً با حذف زوائد، فهرست کامل دعاها پیدا شده و در متغیر lst ذخیره می‌گردد. برای توضیحات بیشتر به قسمت ۲.۵.۲ مراجعه نمایید.

### ۳.۶.۲ تکمیل جدول فهرست دعاها

```
import sqlite3
conn = sqlite3.connect('Sahife.db')
c = conn.cursor()
count = 0
for text in lst:
    count += 1
    c.execute(f'INSERT INTO list VALUES ({count}, "{text}") ')

conn.commit()
conn.close()
```

این قسمت نیز متناظر با بخش ۴.۵.۲ در فرآیند تولید دیتابیس نهج‌البلاغه است. پس از آن که لیست دعاها در قسمت قبل به دست آمد؛ محتویات این لیست را در جدول list در دیتابیس وارد می‌کنیم. برای این کار از یک حلقه‌ی for که متغیر lst را پیمایش می‌کند استفاده می‌کنیم. متغیر count نیز تعداد دعاها را می‌شمارد. در هر گردش از این حلقه، نام دعای مربوطه را همراه با شماره‌ی آن (یعنی متغیر count) در فیلدهای ID و title از دیتابیس وارد می‌کنیم. با تکمیل این فرآیند، جدول list از دیتابیس کامل می‌شود و نام همه‌ی دعاها را همراه با شماره‌ی آن‌ها در بر خواهد داشت. برای مشاهده‌ی توضیحات بیشتر، به قسمت متناظر در مورد نهج‌البلاغه، یعنی بخش ۴.۵.۲ مراجعه نمایید.

## ۴.۶.۲ ایجاد جدول دعاها و ذخیره‌سازی متون عربی دعاها

```

conn = sqlite3.connect('Sahife.db')
c = conn.cursor()
count = 0
sub_count = 0
url = 'http://www.erfan.ir/farsi/sahifeh'
for doa in range(len(lst)):
    print(f'{doa+1} out of {len(lst)}')
    req = requests.get(url+str(doa+1))
    soup = BeautifulSoup(req.text, 'html.parser')
    x = soup.find_all('div', attrs ={'class': ['MyArText', '
        js_MafatidText', 'MafatidText']})
    sub_count = 0
    for line in x:
        print(line.text.strip())
        count += 1
        sub_count += 1
        c.execute(f'INSERT INTO prayer VALUES ({count}, {doa+1}, {
            sub_count}, "{line.text.strip()}", "")')
    conn.commit()

```

این بخش دقیقاً متناظر با قسمت ۷.۵.۲ در بخش مربوط به نهج البلاغه است. آدرس صفحات حاوی دعاهاى مختلف به سادگى به دست مى‌آید. كافى است از الگوی <http://www.erfan.ir/farsi/sahifehN/> استفاده شود که در آن، N شماره‌ی دعای مورد نظر است.

با دسترسی به صفحه‌ی دعای مورد نظر، به دنبال المان‌هایی می‌گردیم که دارای تگ div بوده و ویژگی class آن‌ها به صورت ['MyArText', 'js\_MafatidText', 'MafatidText'] باشد.

روی عناصر حاصل از این جست‌وجو پیمایش کرده و با حذف کاراکترهای فاصله‌ی اضافی از ابتدا و انتهای آن‌ها، بخش‌های مختلف هر دعا را در جدول prayer وارد می‌کنیم. متغیر count شمارنده‌ی همه‌ی بخش‌های همه‌ی دعاها، و متغیر sub\_count شمارنده‌ی بخش‌های هر دعا هستند که به ترتیب در فیلدهای ID و Part\_ID وارد می‌شوند. فیلد Doa\_ID نیز به کمک شمارنده‌ی حلقه (که تعداد دعاها را می‌شمارد) پر می‌شود و نهایتاً فیلد text که متن عربی



حاصل از جست‌وجوی فوق را به خود اختصاص می‌دهد.

برای مشاهده‌ی جزئیات بیشتر، به نمونه‌ی مشابه مربوط به دیتابیس نهج‌البلاغه در قسمت ۷.۵.۲ مراجعه نمایید.

## ۵.۶.۲ تکمیل جدول دعاها و افزودن ترجمه‌های فارسی

```
conn = sqlite3.connect('Sahife.db')
c = conn.cursor()
count = 0
sub_count = 0
url = 'http://www.erfan.ir/farsi/sahifeh'
for doa in range(len(lst)):
    print(f'{doa+1} out of {len(lst)}')
    req = requests.get(url+str(doa+1))
    soup = BeautifulSoup(req.text, 'html.parser')
    y = soup.find_all('div', attrs ={'class': ['MyFaText', '
        js_TranslateText', 'TranslateText']})
    sub_count = 0
    for line in y:
        print(line.text.strip())
        count += 1
        sub_count += 1
        c.execute(f'UPDATE prayer SET translation = "{line.text.strip
            ()}" WHERE ID = {count}')
    conn.commit()
```

نهایتاً این بخش نیز دقیقاً متناظر با قسمت ۸.۵.۲ از دیتابیس نهج‌البلاغه می‌باشد. در قسمت قبل، جدول prayer تقریباً کامل شد و تنها فیلد translation از آن ناقص ماند. در این قسمت، ترجمه‌های بخش‌های مختلف دعاها را به این جدول می‌افزاییم. برای این کار، مشابه قسمت قبل عمل می‌کنیم، تنها با این تفاوت که هنگام جست‌وجو، به دنبال المان‌هایی می‌گردیم که تگ div دارند و ویژگی class آن‌ها به صورت ['MyFaText', 'js\_TranslateText', 'TranslateText'] است. با یافتن این عناصر، به واسطه‌ی یک حلقه‌ی for روی آن‌ها پیمایش می‌کنیم و هر یک را (پس از حذف فواصل اضافی از طرفین رشته) در فیلد translation جدول قرار می‌دهیم. در این جا دقت داریم

که باید سطرهای جدول prayer را به‌روزرسانی کنیم، چرا که چهار مورد از فیلدهای آن‌ها قبلاً پر شده‌اند.

برای بررسی دقیق‌تر این فرآیند، می‌توانید به قسمت متناظر آن در مورد دیتابیس نهج‌البلاغه، یعنی بخش ۸.۵.۲ مراجعه نمایید.

## ۶.۶.۲ جمع‌بندی

مشابه با نهج‌البلاغه، این بار دیتابیس برای صحیفه‌ی سجّادیه ساختیم که حاوی دو جدول است؛ جدول اوّل به نام list که به نوعی فهرست دعاهاست و حاوی نام و شماره‌ی آن‌هاست؛ و جدول دوم به نام prayer که حاوی متون عربی و ترجمه‌ی تمامی دعاهاست. دعاها مطابق با الگوی سایت مرجع به بخش‌های کوچک تقسیم شده‌اند که این بخش‌ها به ترتیب و به صورت منظم در این دیتابیس شماره‌گذاری و مرتّب شده‌اند. با اجرای کدهای بخش‌های قبل به صورت مرتّب، این دیتابیس به صورت کامل ایجاد می‌گردد.

## ۷.۲ پایگاه داده‌ی قرآن کریم

تا به این جا، توضیحات مفصّلی در مورد چگونگی ایجاد دیتابیس‌های نهج‌البلاغه و صحیفه‌ی سجّادیه در بخش‌های قبلی ارائه شده‌اند. چنان‌چه این توضیحات را به دقت بررسی کرده باشید، چگونگی ساخت این دیتابیس‌ها و روش کلی‌ای که لازم است برای تولید دیتابیس‌های مشابه اتخاذ شود را به خوبی درک نموده‌اید. این دیتابیس‌ها می‌توانند از هر وب‌سایتی استخراج شوند و هر موضوع و ساختاری داشته باشند. الگوهای منطقی عملکردی که در قسمت‌های قبلی توصیف شده‌اند، کاملاً کلی هستند و نهج‌البلاغه و صحیفه‌ی سجّادیه نیز به عنوان دو مثال عملی از این موارد، به دقت مورد بررسی قرار گرفته‌اند. به طور خلاصه می‌توان برای ساخت هر دیتابیس دیگری، به بررسی وب‌سایت مورد نظر پرداخت؛ با فشردن کلیدهای Ctrl+Shift+I محتوای HTML آن صفحه را مشاهده کرد، و با پیمایش عناصر آن صفحه و مشاهده‌ی کد HTML متناظر با آن‌ها در پنلی که به واسطه‌ی فشردن کلیدهای مذکور ظاهر شده، بررسی کرد که با جست‌وجوی چه عناصری در کد HTML می‌توان به اطلاعات مورد نظر در صفحه‌ی مربوطه دست یافت. سپس با دریافت کد HTML از طریق کتاب‌خانه‌ی requests، و به کمک کتاب‌خانه‌ی BeautifulSoup به جست‌وجو و تحلیل و پردازش در کد HTML پرداخت.

با توجه به این که این روال برای هر دیتابیس به صورت کلی ثابت است، و چیزی که تفاوت می‌کند پیچیدگی‌هایی است که در نوع داده‌های HTML مختلف ممکن است به چشم بخورد. دیتابیس قرآن نیز نمونه‌ی دیگری از این موارد است که اولاً به دلیل شباهت کلی آن با موارد قبلی و ثانیاً به دلیل آن که نگارنده در حین اجرای این پروژه، تمرکز کمتری بر روی انجام آن داشته است (پیش‌تر بیان شد که تمامی پروژه‌ها از سوی سرپرست کارآموزی به نگارنده و آقای احسان شریفیان به صورت مشترک سپرده شدند، و هر دو نفر در تمامی پروژه‌ها همکاری داشته و دخیل بوده‌اند، با این حال

در برخی موارد یک نفر و در برخی موارد نفر دیگر دخالت بیشتری داشته است) از ارائه‌ی توضیح مفصّل در مورد این دیتابیس پرهیز می‌کنیم و تنها مروری بر کلیّت کد آن خواهیم داشت. در مورد دیتابیس قرآن، توضیحات مفصّل‌تری در گزارش آقای احسان شریفیان موجود خواهد بود.

بدنه‌ی اصلی کد مربوط به دیتابیس قرآن کریم را در ادامه مشاهده می‌نمایید.

```
import requests
import sqlite3
from bs4 import BeautifulSoup

conn = sqlite3.connect('Quran.db')
c = conn.cursor()
c.execute('''CREATE TABLE Quran
(soore_name, jose, aye_number, aye_text, tarjome)''')

for page in range(1,605):
    print(page)
    res = requests.get('http://www.erfan.ir/quran/?page=%s' %page)
    soup = BeautifulSoup(res.text, 'html.parser')
    soore = soup.find_all('span', attrs={'style': 'cursor:pointer'})
    ayat_numbers = soup.find_all('span', attrs = {'class': ['ayaNumber',
        '']})
    soore_features = soup.find_all('h2', attrs= {'class': ['SorehTitle',
        'lh1-5', 'fl']})
    soore_name = soore_features[2]
    jose = soore_features[-1]
    tarjome = soup.find_all('a', attrs = {'class': ['aye', 'relative', '
        inline']})
    for i in range(len(soore)):
        aye = soore[i]
        aye_number_first = ayat_numbers[i]
        aye_number = aye_number_first.text
        tarjome_aye = tarjome[i+2]
```

```

c.execute("INSERT INTO Quran VALUES ('%s', '%s', '%s', '%s',
    '%s')" %(soore_name.text, jose.text, aye_number[1:-1], aye
    .text, tarjome_aye.text.strip()))

conn.commit()

conn.close()

```

مشابه نهج البلاغه و صحیفه‌ی سجّادیه، ابتدا دیتابیس را با جدولی شامل پنج فیلد ایجاد کرده‌ایم. این پنج فیلد به ترتیب مربوط به نام سوره، شماره‌ی جزء، شماره‌ی آیه، متن عربی آیه، و ترجمه‌ی آن می‌باشند. کوچک‌ترین بخشی از قرآن که در یک سطر از جدول ذخیره می‌شود، یک آیه است.

در ادامه، به سراغ ساختار سایت مربوطه در قسمت قرآن می‌رویم. مشاهده می‌شود که کلّ قرآن در ۶۰۵ صفحه از این سایت موجود است که آدرس صفحات آن با الگویی ثابت و بر اساس شماره‌ی صفحه تغییر می‌کند؛ بنابراین می‌توان به راحتی و با استفاده از یک حلقه‌ی for آدرس تمامی این صفحات را ساخت و با استفاده از کتابخانه‌ی requests به محتوای آن‌ها دسترسی یافت.

در ادامه و با بررسی محتوای چند مورد از این صفحات با روش‌های گفته‌شده، و با استفاده از کتابخانه‌ی BeautifulSoup ساختارهایی که برای جست‌وجو مناسب هستند را مورد جست‌وجو قرار داده و از صفحه‌ی وب، مشخصات هر آیه و سوره را به دست می‌آوریم و نهایتاً در سطرهای جدولی که در دیتابیس ساخته‌ایم، قرار می‌دهیم.

تنها یک نکته‌ی جالب در مورد دیتابیس قرآن باقی می‌ماند و آن این است که با اجرای کد فوق، مشکلی در سوره‌های پایانی قرآن به وجود می‌آید و آن مشکل این است که روش یافتن شماره و نام سوره به این صورت بوده که در هر یک از صفحات وب سایت، نام سوره‌ای که در بالای آن نوشته شده بود را به عنوان نام سوره انتخاب کرده بودیم. در صفحات پایانی قرآن، به دلیل آن که سوره‌ها بسیار کوچک هستند، در بعضی موارد در یک صفحه دو یا حتی سه سوره موجود بودند، و این الگوریتم باعث می‌شد که برای آیات تمامی این سوره‌ها نام یک سوره در دیتابیس ذخیره شود که این اشتباه بود. این مشکل همچنین در آیات پایانی سوره‌های بزرگ نیز مشاهده می‌شد، چرا که در یک صفحه یک سوره تمام می‌شد و سوره‌ی دیگر آغاز می‌شد و این باعث می‌شد آیات ابتدایی سوره‌ی جدید با نام سوره‌ی قبلی ذخیره شوند. برای حلّ این مشکل، از کد زیر استفاده شده است:

```

import requests

import sqlite3

from bs4 import BeautifulSoup

```

```

conn = sqlite3.connect('Quran.db')
c = conn.cursor()

c.execute('SELECT ID, soore_name, aye_number FROM Quran')
row = c.fetchall()

conn.close()

num = {
    '۱': '1',
    '۲': '2',
    '۳': '3',
    '۴': '4',
    '۵': '5',
    '۶': '6',
    '۷': '7',
    '۸': '8',
    '۹': '9',
    '۰': '0'
}

n = len(row)

for i in range(n):
    row[i] = list(row[i])
    row[i][2] = list(map(lambda x : x, row[i][2]))
    for j in range(len(row[i][2])):
        row[i][2][j] = num[row[i][2][j]]
    row[i][2] = ''.join(row[i][2])
    row[i][2] = int(row[i][2])

```

```

count = 1
for i in range(n-1):
    row[i][1] = soore_list[count]
    if row[i][2] > row[i+1][2]:
        count = count + 1

import sqlite3
conn = sqlite3.connect('Quran.db')
c = conn.cursor()
for i in range(0):

    c.execute(f'UPDATE Quran Set soore_name = "{row[i][1]}" where ID = {i+1}')
    c.execute(f'UPDATE Quran Set aye_number = {row[i][2]} where ID = {i+1}')

conn.commit()

conn.close()

```

در این کد، دیتابیس اوّلی‌ی قرآن گرفته می‌شود، همه‌ی محتویات سوره‌ها و آیات آن در لیستی به نام row ذخیره می‌شود؛ سپس سطر به سطر بر روی این لیست پیمایش می‌شود و هرگاه شماره‌ی آیه از آیه‌ای به آیه‌ی بعد زیاد نشد، بلکه تبدیل به عدد یک شد، این نتیجه گرفته می‌شود که سوره عوض شده است و وارد سوره‌ی جدیدی شده‌ایم. با این نشانه‌گذاری، مرزبندی سوره‌ها تصحیح می‌شود و مشکلی که پیش‌تر ذکر شد حل می‌گردد. نهایتاً تغییرات جدید را در دیتابیس نیز اعمال می‌کنیم تا این مشکل به طور کامل مرتفع گردد.

## ۸.۲ پایگاه داده‌ی مفاتیح‌الجنان

در مورد پایگاه داده‌ی مفاتیح نیز، دو موردی که در مورد پایگاه داده‌ی قرآن ذکر شد برقرار است، اوّلًا روال کلی کار کاملاً مشابه با موارد قبلی است، و ثانیاً به دلیل تقسیم وظایف در انجام پروژه‌ها، توضیحات بیشتر در مورد آن را

می‌توانید در گزارش آقای احسان شریفیان بررسی کنید.

با این حال ذکر این نکته در مورد دیتابیس مفاتیح لازم است که این مورد، تنهای موردی است که به صورت کامل در طول دوره‌ی کارآموزی به سرانجام رسانده نشد؛ چرا که در حین اجرای آن در ابتدا به مشکلاتی برخوردیم که ناشی از ضعف و بی‌نظم و قاعده بودن کد HTML موجود بر روی سایت <http://www.erfan.ir/> بود. البته به جز این موارد، مشکلات حاصل به ساختار کتاب مفاتیح نیز برمی‌گردد؛ چرا که این کتاب علاوه بر متون عربی و ترجمه‌های فارسی، مقدار قابل توجهی توضیحات و دستورالعمل‌های به زبان فارسی نیز دارد که باعث می‌شود جداسازی آن‌ها از متون عربی و ترجمه‌ها سخت‌تر از سه دیتابیس قبلی باشد. بی‌نظم بودن کد سایت نیز مزید بر علت شد تا علی‌رغم پیشرفت در این کار، پیش از به سرانجام رسیدن این پایگاه داده از جانب سرپرست کارآموزی مأمور به آغاز پروژه‌ی دوم شویم و این کار را با هماهنگی ایشان به آینده موکول کنیم. با این حال، صرفاً برای کامل بودن گزارش، کد مربوط به بخشی از کار دیتابیس مفاتیح که انجام شده بود را نیز در ادامه می‌آوریم. توضیحات بیشتر در این مورد در گزارش آقای احسان شریفیان قابل مشاهده است.

```
import requests
from bs4 import BeautifulSoup
import re
res = requests.get('http://www.erfan.ir/mafatih218')
soup = BeautifulSoup(res.text, 'html.parser')
whole_matn = soup.find_all('article')
count = 0
# Arabi_Text = list()
# Trans_Text = list()
# About_Text = list()
# New_farsi = list()
# New_arabi = list()
for part in whole_matn:
    nextNode = part
    if part.attrs == {'class': ['js_AboutText', 'AboutText']}:
        print('*****')
        print(part.text.strip(), '\n')
        # About_Text.append(part.text.strip())
        # New_farsi.append(part.text.strip())
        # New_arabi.append('...')
```

```

count += 1

nextNode = nextNode.nextSibling

if re.sub('\s','',nextNode.string.strip()) != '':
    print(nextNode.string.strip(), '\n')
    # nextNode = nextNode.nextSibling
    # Arabi_Text.append(nextNode.string.strip())
    # New_arabi.append(nextNode.string.strip())

if part.attrs == {'class': ['js_TranslateText', 'TranslateText']}:
    if count == 0:
        nextNode = nextNode.previousSibling
        if re.sub('\s','',nextNode.string.strip()) != '':
            # nextNode = nextNode.previousSibling
            # About_Text.append('...')
            # Arabi_Text.append(nextNode.string.strip())
            print(nextNode.string.strip(), '\n')
            count += 1
            print(part.text.strip(), '\n')
            # New_arabi.append(nextNode.string.strip())
            # Trans_Text.append(part.text.strip())
            # New_farsi.append(part.text.strip())
    else:
        print(part.text.strip(), '\n')
        # Trans_Text.append(part.text.strip())
        # New_farsi.append(part.text.strip())
        count += 1
        nextNode = nextNode.nextSibling
        if re.sub('\s','',nextNode.string.strip()) != '':
            print(nextNode.string.strip(), '\n')
            # nextNode = nextNode.nextSibling
            # About_Text.append('...')
            # Arabi_Text.append(nextNode.string.strip())

```



```
# New_arabi.append(nextNode.string.strip())

# for i in range(min(len(New_arabi),len(New_farsi))):
# print(New_arabi[i], '\n', New_farsi[i])
```

## ۹.۲ جمع‌بندی

در این فصل، به گزارش تهیه‌ی چهار دیتابیس از چهار کتاب مذهبی قرآن، نهج‌البلاغه، صحیفه‌ی سجّادیه، و مفاتیح پرداختیم. مراحل طی‌شده جهت تولید هر یک از این چهار پایگاه داده شرح داده شد و همچنین به دلایل ناتمام ماندن پروژه‌ی مفاتیح نیز اشاره شد.

برخی موارد برای توضیحات بیشتر به گزارش آقای احسان شریفیان ارجاع داده شد و کدهای پایتون مورد استفاده نیز به صورت کامل (البته به صورت قطعه‌قطعه و نه به صورت یکپارچه) در گزارش آورده شدند.

همچنین در بسیاری از موارد به دلیل تکراری یا مشابه بودن توضیحات، از ذکر نکات تکراری پرهیز شد، با این حال در صورتی که هر یک از بخش‌ها گنگ هستند و نیاز به توضیح بیشتری دارند، می‌توانید از طریق آدرس ایمیل [afsharrad.a@gmail.com](mailto:afsharrad.a@gmail.com) پیگیری نمایید.

## فصل ۳

# پروژه‌ی دوم - استخراج و تحلیل داده از شبکه‌ی اجتماعی اینستاگرام

### ۱.۳ مقدمه

دومین پروژه‌ای که در زمان باقی‌مانده از دوره‌ی کارآموزی به نگارنده و همکار وی (آقای احسان شریفیان) محوّل شد، پروژه‌ی استخراج و تحلیل داده از شبکه‌ی اجتماعی اینستاگرام بود. لازم به ذکر است که این پروژه به صورت کامل بسیار حجیم، گسترده، زمان‌بر و پیچیده است و از ابتدا نیز شرکت محلّ کارآموزی انتظار آن را نداشت تا کارآموزان از ابتدا تا انتهای آن را انجام دهند.

هدف کلّی این پروژه آن است که برنامه‌ای نوشته شود که بتواند به شبکه‌ی اجتماعی اینستاگرام دسترسی یابد و در ادامه، برخی صفحات اینستاگرام را که از پیش تعیین شده‌اند در نظر گرفته و با بررسی کامل محتوای آن‌ها، به خصوص لایک‌ها و کامنت‌های آن‌ها، به چگونگی فعالیت این صفحات پی ببرد. به طور خاص فرض کنید شخصی می‌خواهد صفحه‌ای در اینستاگرام را انتخاب کند تا با پرداخت هزینه، در آن صفحه به تبلیغ فعالیت یا محصول خود بپردازد. سؤال این است که با توجه به موضوعی که این فرد می‌خواهد در مورد آن تبلیغ کند، کدام صفحه‌ی اینستاگرامی برای تبلیغ آن مفیدتر است. به عنوان مثال، فرض کنید موضوعی که فرد می‌خواهد در مورد آن تبلیغ کند ورزشی است. در این صورت ممکن است یک صفحه‌ی اینستاگرامی با میلیون‌ها دنبال‌کننده اصلاً برای این تبلیغ مناسب نباشد، چرا که موضوع آن صفحه اصلاً ورزشی نیست و دنبال‌کننده‌های آن صفحه اصلاً علاقه‌ای به موضوعات ورزشی ندارند. در این حالت ممکن است این فرد در صفحه‌ای که ده درصد صفحه‌ی اوّل دنبال‌کننده دارد، امّا دنبال‌کننده‌های آن ورزشی هستند تبلیغ کند و به مراتب افراد بسیار بیشتری را به خود جذب کند. به طور کلّی هدف آن است که برنامه‌ای نوشته شود که بتواند با تحلیل محتوای کامنت‌های مخاطبین یک صفحه که در زیر پست‌های آن صفحه نوشته می‌شوند؛ تشخیص دهد که مخاطبان این صفحه چه مقدار به هر یک از موضوعات مختلف مانند ورزش، سیاست، تغذیه، موسیقی، و... علاقه‌مند هستند تا

بتواند در نهایت برای کسانی که به دنبال تبلیغ محصولات خود هستند، بهترین گزینه یا گزینه‌ها را جهت تبلیغات معرفی کند.

واضح است که چنین برنامه‌ای باید از ابزارهای پیشرفته‌ی پردازش متن و تحلیل داده و نیز هوش مصنوعی و یادگیری ماشین یا یادگیری عمیق بهره ببرد. با این حال، گام ابتدایی برای ساخت چنین برنامه‌ای آن است که ابزار لازم برای دریافت اطلاعات از اینستاگرام مهیا گردد. کاری که نگارنده و همکار وی در فرصت باقی‌مانده از زمان کارآموزی (پس از پروژه‌ی اول) انجام دادند، نوشتن ابزارهایی به زبان پایتون بود که اولاً بتواند اطلاعات لازم را از شبکه‌ی اجتماعی اینستاگرام دریافت کند، و در مرحله‌ی بعد مقدماتی از امکانات تحلیل داده را فراهم آورد؛ هرچند که تقریباً زمانی برای ورود به فرآیند تحلیل داده در دوره‌ی کارآموزی باقی نماند و این کار – در قالب کارآموزی تابستانی – در همان مرحله متوقف گردید. (البته کارآموزان و شرکت هر دو علاقه‌مند بودند که این پروژه پس از دوره‌ی کارآموزی نیز ادامه یابد).

## ۲.۳ آشنایی با کتابخانه‌ی selenium

در پروژه‌ی اول، با کتابخانه‌ی requests در پایتون آشنا شدیم که ابزاری برای دریافت محتوای HTML صفحات وب از طریق زبان برنامه‌نویسی پایتون بود. این کتابخانه در کنار کتابخانه‌هایی نظیر BeautifulSoup (که آن نیز در فصل قبل مورد بررسی قرار گرفت) از جمله کتابخانه‌های مشهور و پرکاربرد در زمینه‌ی استخراج اطلاعات از صفحات وب (Web Scraping) هستند. با این حال نکته‌ای که در مورد این کتابخانه‌ها به سادگی به ذهن می‌رسد آن است که این کتابخانه‌ها تنها برای دریافت اطلاعات از صفحات وب استفاده می‌شوند؛ و قادر نیستند در صورت لزوم عملی از جنس وارد کردن اطلاعات در صفحه را انجام دهند. به عنوان یک مثال خیلی ساده فرض کنید می‌خواهیم برنامه‌ای بنویسیم که با داشتن رمز ورودی، وارد یک سایت شود. برای این کار لازم است که برنامه بتواند رمز را در صفحه‌ی وب وارد کند. در چنین موقعیتی، نیاز به یک ابزار قدرتمندتر احساس می‌شود.

یکی از قدرتمندترین و مشهورترین کتابخانه‌های موجود در زبان پایتون که مشکل فوق را حل کرده و قابلیت ارسال اطلاعات به صفحات وب را از طریق برنامه‌های پایتون مهیا می‌کند، کتابخانه‌ی selenium است. به کمک این کتابخانه می‌توان برنامه‌ای نوشت که گویی هنگام بررسی صفحات وب به ماوس و صفحه‌کلید کامپیوتر دسترسی دارد، لذا می‌تواند روی دکمه‌ها کلیک کند یا با استفاده از کیبورد متنی را تایپ کند.

برای شروع این بخش از پروژه، کارآموزان زمانی در حدود ۳۰ ساعت را صرف آشنایی کامل با کتابخانه‌ی selenium و نوشتن برنامه‌های تمرینی برای تسلط بر این کتابخانه کردند. اگرچه موضوع این برنامه‌های تمرینی به طور مشخص از سوی سرپرست کارآموزی تعیین نشده بود، اما چون نوشتن آن‌ها بخشی از فرآیند یادگیری بود در این جا به طور مختصر به عناوین آن‌ها اشاره می‌شود:

- اولین برنامه‌ی تمرینی‌ای که کارآموز اقدام به نوشتن آن نمود، برنامه‌ای جهت وارد شدن به سامانه‌ی آموزش دانشگاه صنعتی شریف بود. می‌دانید که صفحه‌ی ورود به این سامانه به صورت شکل ۱.۳ است.

### دانشگاه صنعتی شریف - معاونت آموزشی و تحصیلات تکمیلی - واحد انفورماتیک

رمز خود را فراموش کردم

دریافت شماره دانشجویی با ارائه شماره ملی با شماره داوطلب - مخصوص دانشجویان کارشناسی نیمسال اول ۱۳۹۸-۹۹

شکل ۱.۳: صفحه‌ی ورود به سامانه‌ی آموزش دانشگاه صنعتی شریف

اولین گام، آن است که دو فیلد مربوط به شناسه‌ی کاربر و رمز عبور شناسایی شود. کتابخانه‌ی selenium این امکان را در اختیار قرار می‌دهد که از طریق راه‌های مختلفی بتوان این فیلدها را شناسایی کرد. دقت کنید که این راه‌ها همگی بستگی به کد HTML آن صفحه دارند، لذا مشابه آن چه در فصل قبل به کرات بیان شد، در این جا نیز لازم است با بررسی کد HTML و مشاهده‌ی تگ‌ها و ویژگی‌های المان‌های موجود در صفحه، بتوانیم نحوه‌ی ارجاع‌دهی مناسب آن‌ها به کتابخانه‌ی selenium را بیابیم. این کتابخانه امکان آن را در اختیار قرار می‌دهد که المان‌ها را با توجه به ویژگی‌های مختلف آن‌ها نظیر id، name، class\_name، و... بیابیم. در این مثال خاص، دو فیلد شناسه‌ی کاربر و رمز عبور هر دو از طریق id قابل شناسایی هستند و با استفاده از دستوراتی نظیر `find_element_by_id('password')` و `find_element_by_id('username')` می‌توان به آن‌ها دسترسی یافت. در ادامه نیز این کتابخانه امکان آن را در اختیار قرار می‌دهد که با استفاده از متد `send_keys`، در المان‌هایی که پیش‌تر یافتیم، عباراتی را وارد کنیم. (این عبارات همان شناسه‌ی کاربری و رمز عبور خواهند بود.) در صورتی که کد احراز هویت نبود، کافی بود تا کلید `Enter` از طریق صفحه کلید فشار داده شود تا عملیات وارد شدن به سامانه انجام شود. این کار نیز با استفاده از زیرکتابخانه‌ی `Keys` و دستور `send_keys(Keys.RETURN)` قابل انجام است.

با این حال، در این مثال خاص مشکل کد احراز هویت نیز وجود دارد. برای حل این مشکل، از تکنیک‌های یادگیری ماشین استفاده کردیم. حدود ۱۰۰ نمونه از تصاویر عددی مربوط به احراز هویت را ذخیره کردیم، در هر کدام چهار رقم را از هم جدا کردیم و به صورت دستی، اعداد این رقم‌ها را نیز ذخیره کردیم. سپس یک مدل SVM<sup>۱</sup> بر روی این دیتاست آموزش دادیم که با گرفتن نمونه‌های ارقام و رقم واقعی آن در ورودی، بتواند شناسایی ارقام را یاد بگیرد. درصد پاسخ‌گویی این مدل بسیار بالا بود و در عمل، تقریباً در تمامی موارد می‌توانست پس از یادگیری و

<sup>۱</sup>Support Vector Machine

در مرحله‌ی آزمودن نیز درست عمل کند و کد را تشخیص بدهد. به این ترتیب برنامه قادر بود با آمیزه‌ای از Web Scraping و یادگیری ماشین به سامانه‌ی آموزش وارد شود.

- دومین برنامه‌ی تمرینی‌ای که نوشته شد، برنامه‌ای جهت استخراج اطلاعات از سایت <http://term.inator.ir> بود. این سایت محلّی است که دانشجویان دانشگاه صنعتی شریف می‌توانند قبل از انجام انتخاب واحد، برنامه‌های احتمالی خود را مرتّب کنند. صفحه‌ی ورود به این سایت به صورت شکل ۲.۳ است.

شکل ۲.۳: صفحه‌ی ورود به سایت ترم ایناتور

مشاهده می‌شود که دو روش برای ورود به این سایت وجود دارد، اول با نام کاربری و رمز و عبور، و دوم با شماره‌ی دانشجویی. بسیاری از دانشجویان با شماره‌های دانشجویی اصلی خود و از طریق فیلد شماره‌ی دانشجویی وارد این سایت می‌شوند و این یعنی دیگران نیز می‌توانند در صورت داشتن شماره‌ی دانشجویی آن‌ها، وارد شده و برنامه‌ی آن‌ها را مشاهده کنند. برنامه‌ی تمرینی‌ای که نوشته‌ایم نیز بر همین مبنا عمل می‌کند. این برنامه ابتدا با مراجعه به سایت [http://ee.sharif.ir/bs\\_email.php](http://ee.sharif.ir/bs_email.php)، شماره‌ی دانشجویی و نام تمامی دانشجویان دانشکده‌ی مهندسی برق را دریافت و ذخیره می‌کند. این کار بدون نیاز به کتابخانه‌ی selenium و تنها با استفاده از تکنیک‌هایی که در فصل قبل مورد بررسی قرار دادیم، امکان‌پذیر است.

در ادامه، برنامه به عنوان ورودی شماره‌ی یک یا چند درس از دروس ارائه‌شده را دریافت می‌کند؛ سپس با تک‌تک شماره دانشجویی‌های ذخیره‌شده وارد سایت ترم‌ایناتور شده و بررسی می‌کند که آیا فرد مورد نظر در برنامه‌اش درس مذکور را دارد یا نه. خروجی این برنامه، لیستی از افرادی خواهد بود که در برنامه‌های احتمالی خود درس(های) مورد نظر را قرار داده‌اند.

برای ورود به سایت، عملی کاملاً مشابه با ورود به سامانه‌ی آموزش دانشگاه انجام می‌شود؛ ابتدا فیلد شماره‌ی دانشجویی پیدا شده، سپس شماره‌ی مورد نظر وارد آن می‌شود و کلید Enter فشرده می‌شود. پس از آن، صفحه‌ای به صورت شکل ۳.۳ مشاهده می‌شود.



شکل ۳.۳: اولین صفحه‌ی ظاهرشده پس از ورود به سایت ترم‌ایناتور

در این جا هنوز یک مرحله تا دستیابی به برنامه‌ی مورد نظر فاصله داریم. در شکل ۳.۳ مشاهده می‌شود که در بالای این صفحه سه دکمه وجود دارد. لازم است دکمه‌ی دوم فشار داده شود تا وارد صفحه‌ی برنامه‌ی درسی شویم. برنامه‌ای که نوشته شده قادر است مشابه با عملی که برای پیدا کردن فیلدهای ورود اطلاعات انجام می‌داد، در این جا نیز به دنبال آن دکمه بگردد (پیش‌تر توضیح داده شد که برای این کار لازم است ابتدا کد HTML صفحه بررسی شود) و با یافتن آن، بر روی آن کلیک کند. کلیک کردن نیز به سادگی و با متد click قابل انجام است.

پس از این کار صفحه‌ی برنامه ظاهر می‌شود و می‌توان در محتوای آن به دنبال شماره‌ی درس‌هایی که در ورودی برنامه داده شده بودند گشت. به این ترتیب چنان‌چه شماره‌ی آن درس در محتوای HTML صفحه‌ی برنامه موجود باشد، شخص آن درس را اخذ کرده است. لازم به ذکر است که با کمک متد current\_url می‌توان آدرس صفحه‌ی فعلی را از طریق selenium دریافت کرد، سپس به کمک کتابخانه‌ی requests محتوای آن را دریافت نمود.

با ذکر این دو نمونه از برنامه‌های تمرینی که در فرآیند یادگیری کتابخانه‌ی selenium نوشته شدند، به سراغ برنامه‌ی اصلی دریافت اطلاعات از اینستاگرام می‌رویم.

## ۳.۳ برنامه‌ی دریافت اطلاعات از اینستاگرام

همان طور که پیش‌تر گفته شد، برنامه‌ای که برای دریافت اطلاعات از اینستاگرام نوشته‌ایم، مجموعه‌ای از ابزارهاست. هدف اصلی این ابزارها آن است که کامنت‌های موجود در زیر پست‌های اینستاگرام یک صفحه‌ی مشخص را به طور کامل دریافت و ذخیره کند تا در ادامه بتوان بر روی آن‌ها، تحلیل داده و استنتاج آماری انجام داد. برای این کار، مجموعه‌ای از توابع نوشته شده‌اند که در ادامه، به توضیح هر کدام از آن‌ها می‌پردازیم.

### ۱.۳.۳ تابع `instagram_login`

```
def instagram_login(username, password):  
  
    from selenium import webdriver  
    from selenium.webdriver.common.keys import Keys  
    import time  
  
    driver = webdriver.Firefox()  
    driver.get('https://www.instagram.com/accounts/login/?source=  
        auth_switcher')  
    while True:  
        try:  
            elem = driver.find_element_by_name('username')  
            elem.clear()  
            elem.send_keys(username)  
            break  
        except:  
            pass  
  
    elem = driver.find_element_by_name('password')  
    elem.clear()  
    elem.send_keys(password)
```

```

elem.send_keys(Keys.RETURN)

while True:
    try:
        driver.find_element_by_xpath("//button[@class='a00lW HoLwm
        ']').click()
        break
    except:
        pass

```

این تابع در ورودی نام کاربری و رمز عبور را دریافت می‌کند و با این مشخصات، وارد اکانت اینستاگرام می‌شود.

لازم به ذکر است که همه‌ی اعمال کتابخانه‌ی selenium در یک صفحه‌ی مرورگر اینترنت انجام می‌شود، به گونه‌ای که این صفحه در حین اجرای برنامه باز می‌شود و کاربر می‌تواند تک‌تک مراحل که در برنامه نوشته است را مشاهده کند. به عنوان مثال، می‌بیند که مرورگر باز می‌شود، صفحه‌ی ورود بارگذاری می‌شود، رمزها ناگهان در فیلدها وارد می‌شوند، و نهایتاً کلید Enter فشار داده شده و عملیات وارد شدن انجام می‌شود.

در ادامه به بررسی کد می‌پردازیم. ابتدا از کتابخانه‌ی اصلی selenium، زیرکتابخانه‌های webdriver و Keys مطابق کد فراخوانده می‌شوند.

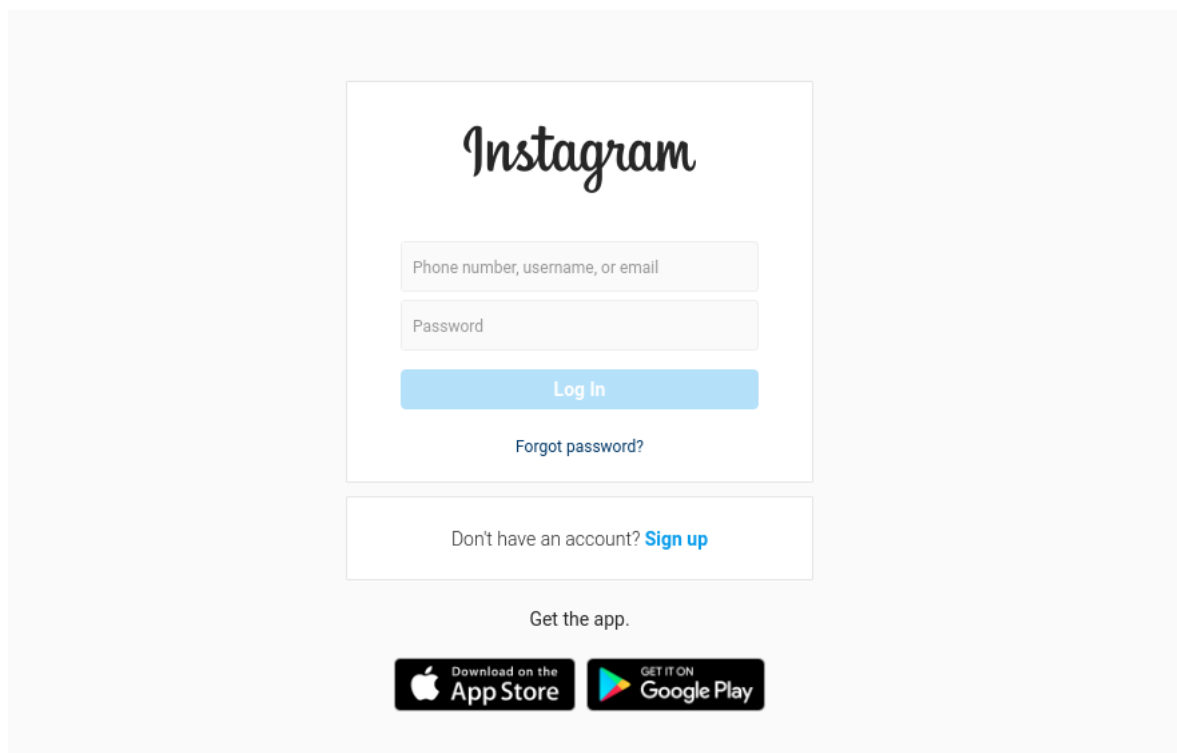
در ادامه لازم است یک driver تعریف شود که در واقع، کنترل‌کننده‌ی مرورگری است که قرار است باز شود و کارها در آن انجام شود. در این برنامه، از طریق کتابخانه‌ی webdriver، مرورگر Firefox را فراخوانی کرده‌ایم. می‌توان این کار را با مرورگرهای دیگر نیز انجام داد.

سپس لازم است صفحه‌ی مورد نظر بارگذاری شود. آدرس صفحه‌ی ورود به اینستاگرام از طریق متد get و به کمک دستور driver.get() به برنامه داده می‌شود. با این کار، برنامه در مرورگر فایرفاکس که آن را باز می‌کند، صفحه‌ی ورود به اینستاگرام را بارگذاری می‌کند.

مرحله‌ی بعدی آن است که فیلدهای لازم برای وارد کردن نام کاربری و رمز عبور را بیابیم. شکل ۴.۳، صفحه‌ی ورود به اینستاگرام را نشان می‌دهد که دارای همان دو فیلد مورد نظر است. در این جا لازم است به کمک کد HTML این صفحه، این دو فیلد را بیابیم.

همان‌طور که پیش‌تر گفته شد، کتابخانه‌ی selenium این امکان را در اختیار ما قرار می‌دهد که با استفاده از ویژگی‌های مختلفی که المان‌های صفحات وب دارند، آن‌ها را بیابیم. متدهای مختلفی برای یافتن المان‌ها وجود دارد که در ادامه لیست آن‌ها را مشاهده می‌کنید:





شکل ۴.۳: صفحه‌ی ورود به اینستاگرام

- `find_element_by_id`
- `find_element_by_name`
- `find_element_by_xpath`
- `find_element_by_link_text`
- `find_element_by_partial_link_text`
- `find_element_by_tag_name`
- `find_element_by_class_name`
- `find_element_by_css_selector`

در این جا و برای یافتن دو فیلد مربوطه، از ویژگی `name` یا به عبارتی، از متد `find_element_by_name` استفاده شده است. با بررسی کد HTML صفحه‌ی ورود به اینستاگرام مشاهده می‌کنیم که ویژگی `name` این دو فیلد، دقیقاً همان `username` و `password` است، لذا می‌توانیم آن‌ها را از این طریق بر روی صفحه بیابیم. ابتدا به سراغ فیلد نام کاربری می‌رویم. `elem` حاصل پیدا کردن این المان است. به عبارتی، با اجرای کد `elem = driver.find_element_by_name('username')`، از نظر برنامه `elem` همان فیلد ورود نام کاربری است. در ادامه با فراخواندن متد `clear` بر روی آن، این فیلد را پاک می‌کنیم (البته به صورت پیش فرض چیزی داخل آن نوشته

نشده، با این حال این کار به نوعی برای اطمینان حاصل کردن انجام می‌شود). سپس با استفاده از متد `send_keys` می‌توانیم در این فیلد هر عبارتی که مد نظر داریم را وارد کنیم. متغیر `username` که از ورودی تابع به عنوان نام کاربری دریافت شده است را وارد می‌کنیم.

در این جا لازم است یک ساختار را توضیح دهیم. ساختار زیر را در نظر بگیرید:

```
while True:
    try:
        CODE
        break
    except:
        pass
```

این ساختار را دفعات زیادی در ادامه در کدهای مربوط به کار با `selenium` استفاده می‌کنیم. این کد از یک حلقه‌ی همواره درست تشکیل شده که تا ابد به گردش ادامه می‌دهد، مگر آن که به دستور `break` برسد. داخل این حلقه، ساختار `try/except` وجود دارد. این ساختار برای مواجه شدن با خطاهای برنامه استفاده می‌شود، به گونه‌ای که ابتدا هر چه در زیرمجموعه‌ی `try` باشد انجام می‌شود. اگر در این قسمت خطایی اتفاق نیفتد، برنامه به صورت عادی جلو می‌رود و اصلاً وارد بخش `except` نمی‌شود. اما اگر در بخش `try` برنامه با خطا مواجه شود، به جای آن که برنامه کلاً متوقف شود، به بخش `except` می‌رود. به این شکل می‌توان از متوقف شدن برنامه در اثر خطاها جلوگیری کرد و خطاها را مدیریت نمود. این کار را `exception handling` نیز می‌نامند.

اما علت استفاده از این ساختار در این جا این است که در بسیاری از موارد، ممکن است به دلیل پایین بودن سرعت اینترنت یا سایت مورد نظر، زمانی طول بکشد تا صفحه‌ای که از طریق `selenium` می‌خواهیم به آن دسترسی پیدا کنیم، به صورت کامل بارگذاری شود. در این حالت ممکن است به دلیل آن که صفحه هنوز ناقص است و اطلاعات آن به صورت کامل دریافت نشده، دستورات بعدی کد که با فرض کامل بودن صفحه نوشته شده‌اند به خطا بخورند. در این حالت از ساختار فوق استفاده می‌کنیم و کد مد نظر را درون یک حلقه‌ی بی‌نهایت و داخل عبارت `try` قرار می‌دهیم و پس از آن نیز دستور `break` را می‌نویسیم. همچنین در قسمت `except` نیز عبارت `pass` را می‌نویسیم که یعنی برنامه هیچ کار خاصی نکند و صرفاً به اجرا ادامه دهد. به این ترتیب برنامه تا زمانی که کد مورد نظر بدون خطا اجرا شود در حلقه گیر می‌کند و سعی می‌کند آن کد را اجرا کند. به محض آن که این کد بدون خطا اجرا شد، به دستور `break` می‌رسیم و برنامه از این حلقه خارج می‌شود و ادامه می‌یابد.

این ساختار را از این پس `while-try-except` می‌نامیم و در قسمت‌های آینده نیز به همین اسم از آن یاد می‌کنیم. علت استفاده از این ساختار در این جا نیز همین است، ممکن است زمانی که می‌خواهیم نام کاربری را وارد کنیم، صفحه هنوز به درستی بارگذاری نشده باشد، بنابراین با استفاده از ساختار `while-try-except` منتظر می‌مانیم تا اطلاعات

صفحه کامل شود، سپس نام کاربری را وارد می‌کنیم.

در ادامه عمل مشابه را برای رمز عبور (password) نیز انجام می‌دهیم و کلید Enter را فشار می‌دهیم. کلید Enter از طریق `Keys.RETURN` قابل دسترسی است.

با این کار، عملیات ورود به اینستاگرام انجام می‌شود، با این حال مشاهده می‌کنید که قطعه کد دیگری نیز در انتهای این تابع موجود است. علت این امر آن است که پس از ورود به اینستاگرام، ممکن است پیغام دیگری ظاهر شود که لازم است صرفاً با کلیک کردن بر روی آن، آن را از بین برد.

به این ترتیب کار این تابع کامل می‌شود و با اکانت مورد نظر، وارد اینستاگرام می‌شویم.

### ۲.۳.۳ تابع `get_posts_url_list`

```
def get_posts_url_list(page_name, number_of_posts):
    from selenium import webdriver
    from selenium.webdriver.common.keys import Keys
    import time

    driver = webdriver.Firefox()
    driver.get('https://www.instagram.com/' + page_name)

    posts_url_list = []

    while len(posts_url_list) < number_of_posts:

        posts = driver.find_elements_by_xpath("//div[@class='v1Nh3 kIKUG _bz0w']")

        for post in posts:
            post_url = post.find_element_by_css_selector('a').
                get_attribute("href")
```

```

if post_url not in posts_url_list and len(
    posts_url_list) < number_of_posts:
    posts_url_list.append(post_url)

elem = driver.find_element_by_tag_name('html')
elem.send_keys(Keys.END)
time.sleep(2)

driver.close()

return posts_url_list

```

این تابع در ورودی نام یک صفحه‌ی اینستاگرام را به همراه تعداد پست‌های مورد نظر (مثلاً  $n$ ) می‌گیرد و در خروجی، لیستی از آدرس‌های صفحات مربوط به  $n$  پست آخر آن صفحه را می‌دهد. این لیست، مقدمه‌ای خواهد بود برای آن که بتوانیم اطلاعات مربوط به  $n$  پست آخر یک صفحه‌ی اینستاگرام را استخراج کنیم.

پس از فراخوانی کتابخانه‌های مورد نیاز، ایجاد درایور، و ارسال آدرس صفحه‌ی مورد نظر (که در بخش ۱.۳.۳ بررسی شده‌اند)، ابتدا یک لیست خالی به نام `posts_url_list` تولید می‌کنیم. قرار است آدرس  $n$  پست آخر صفحه‌ی مورد نظر را به ترتیب در این لیست قرار دهیم.

در یک حلقه‌ی `while`، تا زمانی که به تعداد مطلوب از آدرس‌های پست‌های صفحه‌ی مورد نظر دست نیافته‌ایم، شروع می‌کنیم به پیدا کردن آدرس‌های بیشتر. برای این کار، به کمک متد `find_elements_by_xpath`، المان‌های مربوط به پست‌های اینستاگرام را می‌یابیم. این که چرا باید چنین جست‌وجو کنیم، به آن برمی‌گردد که قبل از نوشتن کد به بررسی کد HTML صفحه‌ی اینستاگرام پردازیم و مشاهده کنیم که پست‌های آن با چه تگ‌ها و ویژگی‌هایی مشخص می‌شوند.

حاصل این جست‌وجو را در متغیر `posts` ذخیره می‌کنیم، سپس روی آن‌ها به کمک یک حلقه‌ی `for` پیمایش می‌نماییم و از میان نتایج، آدرس‌های آن‌ها را که در ویژگی `href` آن‌ها قرار دارد، به کمک متد `get_attribute` استخراج می‌کنیم و در صورتی که هنوز پست‌ها به تعداد کافی نرسیده باشند، به لیست `posts_url_list` اضافه می‌کنیم.

اگر پس از پایان این حلقه‌ی `for` هنوز به تعداد کافی از پست‌ها نرسیده باشیم، یعنی در اولین باری که صفحه‌ی اینستاگرام ظاهر شده است، در آن صفحه تعداد کافی پست وجود نداشته است. در حالت عادی کافی است در مرورگر به سمت پایین حرکت کنیم تا پست‌های بیشتری لود شوند. این کار را از طریق `selenium` با فشار دادن دکمه‌ی `Keys.END`

انجام می‌دهیم. این دکمه در واقع همان دکمه‌ی end موجود بر روی صفحه کلید است. این کار باعث می‌شود پست‌های قدیمی‌تر بیشتری در روی صفحه لود شوند و با تکرار حلقه‌ی while، لیست پست‌ها بیشتر و بیشتر می‌شود و این روال تا زمانی ادامه می‌یابد که به تعداد مورد نظر برسیم. ضمناً هنگام اضافه کردن آدرس پست جدید به لیست آدرس‌ها، این شرط را گذاشته‌ایم که آدرس قبلاً در لیست قرار نگرفته باشد. علت این امر آن است که با فشار دادن کلید end، اگر چه پست‌های جدیدی ظاهر می‌شوند، اما همه‌ی پست‌های قبلی از بین نمی‌روند؛ و اگر این شرط وجود نداشته باشد، آدرس‌های تکراری درون لیست قرار می‌گیرند که نامطلوب است.

به این ترتیب، این تابع کار خود را تمام می‌کند و در خروجی، لیستی از آدرس  $n$  پست آخر صفحه‌ی مورد نظر بر می‌گرداند.

## ۴.۳ تابع get\_comments

```
def get_comments(posts_url_list):
    from selenium import webdriver
    from selenium.webdriver.common.keys import Keys
    import time

    driver = webdriver.Firefox()

    comments_list, users_list = [], []

    for post in posts_url_list:
        driver.get(post)
        count = 0
        while True:
            try:
                elem = driver.find_element_by_xpath("//span[
                    @aria-label='Load more comments']")
                elem.click()
            except:
                try:
```

```

        elem = driver.find_element_by_xpath("//
            button[@class='Z4IfV _Omzm- sqdOP
                yWX7d ']")
        elem.click()
    except:
        count += 1
        pass
    if count == 1000:
        break

while True:
    try:
        comments = driver.find_elements_by_class_name("
            C4VMK")
        break
    except:
        pass

for c in comments:
    count = 0
    while count < 10:
        try:
            comment = c.find_element_by_css_selector('span')
                .get_attribute("textContent")
            user = c.find_element_by_class_name("TlrDj").
                get_attribute("textContent")
            comments_list.append(comment)
            users_list.append(user)
            break
        except:
            count += 1

```

```
pass
```

```
driver.close()
```

```
return users_list, comments_list
```

این تابع در ورودی لیستی از پست‌های یک صفحه‌ی اینستاگرام را دریافت می‌کند (که این لیست می‌تواند خروجی تابع قسمت قبل باشد) و در خروجی، لیست تمام کامنت‌های این پست‌ها و نیز نام تمام کاربرانی که این کامنت‌ها را گذاشته‌اند را در دو لیست مجزاً تحویل می‌دهد.

پس از فراخوانی کتابخانه‌های مورد نیاز، ایجاد درایور، و ارسال آدرس صفحه‌ی مورد نظر (که در بخش ۱.۳.۳ بررسی شده‌اند)، ابتدا دو لیست خالی با نام‌های `users_list` و `comments_list` ایجاد می‌کند. سپس با یک حلقه‌ی `for`، روی تک‌تک صفحاتی که آدرس آن‌ها را در ورودی گرفته است، پیمایش می‌نماید. ابتدا آن صفحه را فرا می‌خواند. در ادامه باید کامنت‌های این صفحه استخراج شوند.

هنگامی که یک پست اینستاگرام باز می‌شود، تعداد کمی از کامنت‌ها در همان لحظه قابل مشاهده هستند. برای آن که کامنت‌های بیشتری مشاهده شوند، لازم است دکمه‌ای که علامت + دارد فشار داده شود. این کار باعث می‌شود تعدادی کامنت دیگر در صفحه ظاهر شوند؛ با این حال باز هم احتمالاً تعداد زیادی کامنت باقی مانده‌اند که هنوز نشان داده نشده‌اند. برای آن که تمامی کامنت‌ها در صفحه ظاهر شوند، لازم است که دکمه‌ی + آن قدر فشار داده شود تا دیگر این دکمه در صفحه موجود نباشد و تمامی کامنت‌ها در صفحه ظاهر شده باشند.

ابتدا با بررسی کد HTML صفحه‌ی اینستاگرام، بررسی می‌کنیم که چطور باید دکمه‌ی + را فشرد. سپس از ساختار `while-try-except` که در قسمت ۱.۳.۳ به تفصیل توضیح داده شد استفاده می‌کنیم تا این دکمه به صورت متوالی فشرده شود. با توجه به این که زمانی که همه‌ی کامنت‌ها نمایش داده شده‌اند، دیگر لازم نیست این دکمه فشار داده شود، یک حد آستانه برای آن تعیین می‌کنیم. این حد آستانه در کد فوق برابر ۱۰۰۰ در نظر گرفته شده است، یعنی اگر ۱۰۰۰ بار تلاش برای فشردن دکمه‌ی + انجام شود و هیچ‌کدام موفق نباشند، برنامه فرض می‌کند که دیگر کامنتی وجود ندارد.

یک نکته‌ی دیگر که در کد برنامه قابل مشاهده است، آن است که در واقع به جای یک ساختار `while-try-except` ساده، از دو ساختار `while-try-except` تودرتو استفاده شده است. علت این امر آن است که مشاهده می‌شود که بعضی اوقات، علامت + در صفحه‌ی اینستاگرام جای خود را به یک لینک با عبارت `load more comments` می‌دهد. این برنامه برای فشردن هر دو دکمه تلاش می‌کند، و اگر هیچ یک از این دو را در صفحه پیدا نکند، فرض می‌کند که موفق به زدن دکمه‌ی ظاهر شدن کامنت‌های بیشتر نشده است.

پس از آن که این فرآیند به پایان رسید، انتظار داریم که تمامی کامنت‌های پست مورد نظر در صفحه وجود داشته باشد. مجدداً با بررسی کد HTML در می‌یابیم که برای یافتن کامنت‌ها باید به دنبال المان‌هایی بگردیم که ویژگی class آن‌ها به صورت C4VMK است. همه‌ی این المان‌ها را در متغیر comments ذخیره می‌کنیم، سپس به کمک یک حلقه‌ی for آن‌ها را پیمایش می‌کنیم. با استفاده از امکانات جست‌وجوی کتابخانه‌ی selenium، می‌توانیم از هر یک از این المان‌ها، متن کامنت و نیز نام اکانت نویسنده‌ی آن را جدا کنیم. این دو را در متغیرهای user و comment ذخیره می‌کنیم؛ سپس این دو متغیر را به لیست‌های comments\_list و users\_list می‌افزاییم. به این ترتیب تمام کامنت‌های پست مورد نظر و نیز نام نویسنده‌های آن‌ها در لیست‌های مربوطه ذخیره می‌شود. با اتمام بررسی این پست، در حلقه‌ی for بیرونی به سراغ پست بعدی می‌رویم و لیست کامنت‌ها به همین منوال، کامل و کامل‌تر می‌شود. نهایتاً همه‌ی پست‌ها تمام می‌شوند و تابع در خروجی، لیست کامنت‌ها و نویسنده‌های آن‌ها را در اختیار قرار می‌دهد.

### ۱.۴.۳ تابع comments\_to\_csv

```
def comments_to_csv(users_list, comments_list, file_name):
    import pandas as pd

    df = pd.DataFrame({"User": users_list, "comment": comments_list})
    df.to_csv(file_name + ".csv", index = False)
```

این تابع دو خروجی تابع قسمت قبل را همراه با نام انتخابی برای فایل دریافت می‌کند، و این اطلاعات را در یک فایل به فرمت csv<sup>۲</sup> ذخیره می‌کند. برای انجام این کار از کتابخانه‌ی pandas که یکی از کتابخانه‌های بسیار مشهور و قدرتمند در زمینه‌ی طبقه‌بندی و تحلیل داده است، استفاده می‌کند.

### ۲.۴.۳ تابع comments\_from\_csv

```
def comments_from_csv(file_name):
    import pandas as pd

    data = pd.read_csv(file_name + ".csv")
    data = data.values.tolist()

    users_list = list(x[0] for x in data)
```

<sup>۲</sup>Comma-Separated Values



```

comments_list = list(x[1] for x in data)

return users_list, comments_list

```

این تابع دقیقاً عمل برعکس تابع `comments_to_csv` را انجام می‌دهد؛ یعنی با دریافت فایل `csv` که توسط تابع `comments_to_csv` تولید شده، دو لیست در خروجی می‌دهد، یکی لیست کامنت‌ها و دیگری لیست اکانت‌هایی که این کامنت‌ها را نوشته‌اند. مجدداً برای این کار نیز از کتابخانه‌ی `pandas` استفاده شده است.

### ۳.۴.۳ تابع `word_frequency`

```

def word_frequency(string, min_freq, ignore_list):
    if(str(type(string)) == "<class 'list'>"):
        string = ' '.join(string)

    word_freq = []

    str1 = string.split()
    str2 = list(set(str1))

    for word in str2:
        if str1.count(word) >= min_freq and word not in ignore_list:
            word_freq.append((word, str1.count(word)))

    word_freq = sorted(word_freq, key = lambda x: x[1], reverse = True)
    return word_freq

```

این تابع در ورودی خود، رشته‌ای از کلمات (یا لیستی از رشته‌ها، که در واقع همان لیست کامنت‌ها است) را همراه با یک عدد که آن را فرکانس کمینه می‌نامیم و نیز یک لیست که به آن لیست صرف‌نظرشونده می‌گوییم، دریافت می‌کند. وظیفه‌ی این تابع این است که کلمات را در ورودی بر حسب تعداد دفعات تکرار از زیاد به کم مرتب کند. دومین ورودی تابع که آن را فرکانس کمینه نامیدیم، عددی است که برای مقادیر کمتر از آن، از تکرارهای کلمات صرف‌نظر می‌شود. مثلاً اگر این عدد برابر با ۵ باشد، تابع تنها در خروجی کلماتی را لیست می‌کند که در ورودی بیشتر از ۵ بار تکرار شده باشند. همچنین ورودی سوم نیز لیست صرف‌نظرشونده نام داشت. چنانچه کلمه‌ای در این لیست وجود داشته باشد،

این تابع مستقل از تعداد دفعات تکرار این کلمه، از آن صرف نظر می‌کند و آن را در لیست خروجی قرار نمی‌دهد. خروجی این تابع لیستی از دوتایی‌هاست که کلمات را در کنار تعداد دفعات تکرار آن‌ها در لیست قرار داده است.

هدف از این تابع آن است که لیست کامنت‌ها را دریافت کند، و در خروجی کلماتی که در این کامنت‌ها دفعات زیادی تکرار شده‌اند را تحویل دهد. این کار باعث می‌شود بتوانیم در راستای هدف اصلی این پروژه که در ۱.۳ ذکر شد، موضوع فعالیت هر صفحه را تشخیص دهیم.

فرآیند پیاده‌سازی این تابع ساده است و دیگر از توضیح دادن جزئیات کد پایتون صرف نظر می‌کنیم.

### ۴.۴.۳ تابع total\_word\_count

```
def total_word_count(list_of_strings):
    count = 0
    for string in list_of_strings:
        count += len(string)

    return count
```

این تابع نیز یک ابزار کمکی کوچک است که در ورودی لیستی از رشته‌ها را دریافت می‌کند و در خروجی، تعداد کل کلمات را برمی‌گرداند. از این تابع در قسمت بعدی (تابع ignore\_list\_creator) استفاده می‌شود.

### ۵.۴.۳ تابع ignore\_list\_creator

```
def ignore_list_creator(csv_list, pre_address, ignore_list):
    total_list = []
    for item in csv_list:
        users_list, comments_list = comments_from_csv(pre_address +
            item)
        list_of_word_freq = word_frequency(comments_list,
            total_word_count(comments_list)/10000, [])
        total_list += list(map(lambda x: x[0], list_of_word_freq))

    new_ignore_list = word_frequency(total_list, 2, [])
```

```
ignore_list += list(map(lambda x: x[0], new_ignore_list))

return ignore_list
```

این تابع لیستی از آدرس فایل‌های csv را دریافت می‌کند. این فایل‌ها باید به فرمت خروجی‌های تابع `comments_to_csv` باشند. همچنین ورودی دیگر این تابع، لیستی به نام `ignore_list` است. (تابع ورودی دیگری نیز به اسم `pre_address` نیز دارد که آدرسی است که باید قبل از نام فایل‌های csv قرار دهد تا به آن‌ها دسترسی یابد. اگر فایل‌های csv در دایرکتوری خود برنامه باشند، کافی است ورودی دوم تهی داده شود).

هدف این تابع آن است که کامنت‌هایی که از صفحات مختلف جمع‌آوری شده‌اند را با یک‌دیگر مقایسه کند و کلماتی را که در چند مورد از این صفحات بوده‌اند، در لیست صرف‌نظرشونده قرار دهد. فلسفه‌ی این کار آن است که به عنوان مثال، اگر کلمه‌ای در کامنت‌های یک صفحه‌ی ورزشی بسیار تکرار شده، و در عین حال در کامنت‌های یک صفحه‌ی سیاسی نیز پرتکرار است؛ احتمالاً این کلمه خنثی است؛ یعنی با استفاده از آن نمی‌توان موضوعات صفحات مختلف را از هم جدا کرد. بنابراین این کلمات باید در لیست صرف‌نظرشونده قرار بگیرند و این لیست را تکمیل کنند تا بعداً در سنجش موضوعات صفحات از آن‌ها استفاده نشود.

عملکرد تابع نیز دقیقاً در همین راستا است. این تابع ابتدا با استفاده از تابع `word_frequency` کلمات پرتکرار هر یک از صفحات را می‌یابد؛ سپس در صورتی که در میان این کلمات پرتکرار، کلمه‌ی مشترکی پیدا شود؛ آن را در لیست صرف‌نظرشونده قرار می‌دهد. ضمناً حدّ آستانه‌ی کلمات پرتکرار (که یکی از ورودی‌های تابع `word_frequency` بود) نیز برابر با یک‌هزارم تعداد کلّ کلمات موجود در کامنت‌های آن صفحه (که این تعداد به کمک تابع `total_word_count` محاسبه می‌شود) در نظر گرفته شده است.

## ۵.۳ یک نمونه‌ی آزمایشی از عملکرد توابع

تا به این‌جا، ابزارهایی که برای استخراج اطلاعات از اینستاگرام طراحی شده بودند را معرفی نمودیم. در این مرحله، به عنوان نمونه این ابزارها را روی ۱۰ صفحه‌ی مختلف اینستاگرامی آزمایش می‌کنیم؛ سپس با استفاده از تابع `ignore_list_creator` کلمات مشترک آن‌ها را حذف می‌کنیم، و نهایتاً با حذف این کلمات، سعی می‌کنیم نمونه‌ای از کلمات کلیدی حوزه‌های ورزش، فیلم و سینما، و سیاست را مشاهده کنیم.

برای این کار، کد زیر را اجرا می‌کنیم.

```
page_list = ['90tv.official', 'hassan_reyvandi', 'masih.alinejad', 'ebi', 'chikhobe', 'sooriland', 'film_bazzan', 'mr.taster', 'art.fatemehebad',
```

```
'homayounshajarian', 'rambodjavan1']

for page in page_list:
    posts_url_list = get_posts_url_list(page, 30)
    users_list, comments_list = get_comments(posts_url_list)
    comments_to_csv(users_list, comments_list, 'InstagramData/' + page)

ignore_list = ignore_list_creator(page_list, 'InstagramData/', [])
```

موضوعات صفحات انتخابی تقریباً نسبت به یک‌دیگر بی‌ارتباط هستند؛ موضوعاتی نظیر ورزش، طنز، سیاست، موسیقی و خوانندگی، متفرقه، فیلم و سریال، غذا و خوراکی، و هنر.

حال به چند نمونه از نتایج نگاه می‌کنیم. شکل‌های ۵.۳ و ۶.۳ و ۷.۳ به ترتیب کلمات پرتکرار سه صفحه‌ی ورزشی، فیلم و سینمایی، و سیاسی را بدون حذف کلمات خنثی نشان می‌دهند.

```
('و', 483)
('به', 328)
('از', 271)
('که', 253)
('این', 229)
('تو', 224)
('رو', 188)
('هم', 166)
('با', 163)
('گل', 145)
('رزومه', 133)
('تیم', 129)
('به', 108)
('نا', 107)
('بود', 98)
('دایی', 90)
('فقط', 87)
('در', 86)
('های', 78)
('زده', 74)
('علی', 72)
('مربی', 69)
('داره', 68)
('همه', 66)
('ایران', 65)
('اون', 65)
('رونالدو', 65)
('بازی', 64)
('ی', 60)
('چه', 58)
(' ', 57)
('ب', 56)
('ای', 56)
('دیگه', 54)
('وزیر', 52)
('چرا', 52)
('ها', 51)
('الان', 50)
('خودت', 49)
('ولی', 49)
('بی', 47)
```

شکل ۵.۳: لیست کلمات پرتکرار صفحه‌ی ورزشی بدون حذف کلمات خنثی

```
( 'و', 825 )
( 'از', 403 )
( 'که', 374 )
( 'فیلم', 361 )
( 'رو', 329 )
( 'بود', 301 )
( 'به', 290 )
( 'این', 290 )
( 'من', 248 )
( 'هم', 222 )
( 'خیلی', 220 )
( 'با', 190 )
( 'تو', 185 )
( 'در', 132 )
( 'یه', 128 )
( 'عالی', 127 )
( 'فصل', 120 )
( 'فقط', 116 )
( 'سریال', 110 )
( 'ولی', 110 )
( 'های', 104 )
( 'واقعا', 100 )
( 'ها', 95 )
( 'دیدم', 92 )
( 'اون', 82 )
( 'خوب', 74 )
( 'تا', 73 )
( ' ', 69 )
( 'بازی', 65 )
( 'دیگه', 63 )
( 'همه', 63 )
( 'برای', 62 )
( 'بهترین', 61 )
( 'می', 60 )
( 'دو', 60 )
( 'بی', 58 )
```

شکل ۶.۳: لیست کلمات پرتکرار صفحه‌ی مربوط به فیلم و سریال بدون حذف کلمات خنثی

از طرف دیگر، شکل‌های ۷.۳ و ۸.۳ و ۹.۳ کلمات پرتکرار را پس از آن که کلمات خنثی حذف شده‌اند نشان می‌دهد. حذف این کلمات به کمک تابع `ignore_list_creator` صورت گرفته است.

چیزی که از این شش تصویر به وضوح مشاهده می‌شود، آن است که اگر کلمات خنثی حذف نشوند، نتایج اصلاً قابل اعتنا نیستند و کلمات پرتکرار بیشتر شامل افعال و حروف ربط پرتکرار بوده و بسیاری از آن‌ها اصلاً نمی‌توانند به تشخیص موضوع صفحه‌ی مربوطه کمک کنند. از طرف دیگر، مشاهده می‌شود که پس از حذف کلمات مشترک بین این صفحات (همان کلمات صرف‌نظرشونده)، کلمات پرتکرار هر سه صفحه تا حدّ بسیار خوبی به موضوع آن صفحه مربوط هستند و می‌توان از روی آن‌ها، در مورد موضوع صفحه قضاوت کرد.

البته آنچه تا به این جا انجام شد، صرفاً یک گام بسیار کوچک و ابتدایی در راستای تحلیل داده و رسیدن به هدف اصلی این پروژه بود، اما فرصت محدود دوره‌ی کارآموزی امکان آن را نداد تا در این دوره، بتوانیم این پروژه را فراتر ببریم.

## ۶.۳ جمع‌بندی

در این فصل، به بررسی پروژه‌ی استخراج و تحلیل داده از شبکه‌ی اجتماعی اینستاگرام پرداختیم. ابتدا ابزار قدرتمندی به نام کتابخانه‌ی `selenium` را معرفی نمودیم؛ سپس به کمک آن ابزارهایی ساختیم که می‌توانستند از یک صفحه‌ی اینستاگرام، اطلاعات مورد نیاز، یعنی همه‌ی کامنت‌های تعداد دلخواهی از پست‌های آن را استخراج کنند. در کنار این

( 'و' , 1233 )  
 ( 'که' , 909 )  
 ( 'به' , 798 )  
 ( 'از' , 762 )  
 ( 'این' , 741 )  
 ( 'تو' , 451 )  
 ( 'رو' , 409 )  
 ( 'با' , 351 )  
 ( 'در' , 320 )  
 ( 'هم' , 308 )  
 ( 'ما' , 251 )  
 ( 'را' , 229 )  
 ( 'همه' , 228 )  
 ( 'مسیح' , 209 )  
 ( 'دختر' , 204 )  
 ( 'من' , 188 )  
 ( 'برای' , 187 )  
 ( 'ایران' , 185 )  
 ( 'بی' , 181 )  
 ( 'به' , 175 )  
 ( 'بر' , 174 )  
 ( 'چه' , 169 )  
 ( 'ها' , 161 )  
 ( 'چرا' , 154 )  
 ( 'بدر' , 152 )  
 ( 'مردم' , 150 )  
 ( 'تا' , 142 )  
 ( 'سحر' , 142 )  
 ( 'دیگه' , 132 )  
 ( 'بود' , 129 )  
 ( 'فقط' , 128 )  
 ( 'با بد' , 127 )  
 ( 'ای' , 124 )  
 ( 'شما' , 123 )  
 ( 'اون' , 121 )  
 ( 'استاد یوم' , 116 )  
 ( 'داره' , 113 )  
 ( 'خیلی' , 111 )

شکل ۷.۳: لیست کلمات پرتکرار صفحه‌ی سیاسی بدون حذف کلمات خنثی

ابزارها، توابع مقدماتی دیگری نیز در راستای تحلیل داده‌های استخراج شده نوشتیم.

البته مجدداً ذکر می‌شود که مسیر تحلیل داده‌های حاصل از این شبکه‌ی اجتماعی، مسیری طولانی و جذاب است که در پایان این دوره‌ی کارآموزی، ما تنها در ابتدای آن هستیم و امکان پیاده‌سازی الگوریتم‌های هوشمند برای پردازش زبان و تشخیص چگونگی رفتار مخاطبین صفحات مجازی برای تکمیل و به‌ثمررساندن این پروژه در آینده وجود خواهد داشت.

در این گزارش سعی شده توضیحات به مقدار کافی و به صورت گویا بیان شوند، با این حال در صورت وجود هر گونه کاستی، می‌توانید از طریق آدرس ایمیل [afsharrad.a@gmail.com](mailto:afsharrad.a@gmail.com) این موارد را جویا شوید.

( 'رزومه' , 133 )  
 ( 'تیم' , 129 )  
 ( 'دایی' , 90 )  
 ( 'علی' , 72 )  
 ( 'مری' , 69 )  
 ( 'رونالدو' , 65 )  
 ( 'وزیر' , 52 )  
 ( 'بازیکن' , 46 )  
 ( 'کیسه' , 46 )  
 ( 'استقلال' , 46 )  
 ( 'فرهاد' , 44 )  
 ( 'جام' , 37 )  
 ( 'رنال' , 37 )  
 ( 'بارسا' , 34 )  
 ( 'باشگاه' , 32 )  
 ( 'مس' , 32 )  
 ( 'سرمربی' , 32 )  
 ( 'پرسپولیس' , 30 )  
 ( 'لیگ' , 30 )  
 ( 'تیمای' , 27 )  
 ( 'مجیدی' , 25 )  
 ( 'برزیل' , 24 )  
 ( 'لباس' , 24 )  
 ( 'هائری' , 23 )  
 ( 'مالدیو' , 22 )  
 ( 'زد' , 21 )  
 ( 'آرسنال' , 21 )  
 ( 'انگلیس' , 21 )  
 ( 'نزده' , 21 )  
 ( 'بارسلونا' , 18 )  
 ( 'ج' , 18 )  
 ( 'پرسپولیس' , 18 )  
 ( 'جایزه' , 18 )  
 ( 'اینترا' , 18 )  
 ( 'فرنایز' , 18 )  
 ( 'خب' , 18 )  
 ( 'ایتالیا' , 18 )  
 ( 'اروپا' , 17 )  
 ( 'حکومتی' , 17 )

شکل ۸.۳: لیست کلمات پرتکرار صفحه‌ی ورزشی پس از حذف کلمات خنثی

( 'سریال' , 110 )  
 ( 'جانی' , 54 )  
 ( 'the' , 46 )  
 ( 'لالند' , 38 )  
 ( 'موزیکال' , 35 )  
 ( 'تام' , 34 )  
 ( 'دب' , 33 )  
 ( 'بتمن' , 33 )  
 ( 'رایان' , 33 )  
 ( 'The' , 31 )  
 ( 'نقش' , 31 )  
 ( 'هری' , 28 )  
 ( 'بیل' , 27 )  
 ( 'فیلم' , 26 )  
 ( 'فیلمای' , 26 )  
 ( 'نبود' , 24 )  
 ( 'بندیکت' , 24 )  
 ( 'پاتر' , 23 )  
 ( 'of' , 23 )  
 ( 'سمت' , 23 )  
 ( 'بینویان' , 23 )  
 ( 'نظرم' , 22 )  
 ( 'برد' , 22 )  
 ( 'موسیقی' , 21 )  
 ( 'کارایلانکا' , 21 )  
 ( 'عاشقا نه' , 20 )  
 ( 'بازیگر' , 19 )  
 ( 'هاردی' , 19 )  
 ( 'اسکار' , 18 )  
 ( 'خوبه' , 17 )  
 ( 'تايتانیک' , 17 )  
 ( 'چی' , 17 )  
 ( 'کلا' , 17 )  
 ( 'نمیدونم' , 17 )  
 ( 'نولان' , 17 )

شکل ۹.۳: لیست کلمات پرتکرار صفحه‌ی مربوط به فیلم و سریال پس از حذف کلمات خنثی

( '209 , مسیح )  
 ( '152 , پدر )  
 ( '116 , استا د یوم )  
 ( '76 , زبان )  
 ( '64 , دروغ )  
 ( '64 , مرگ )  
 ( '62 , خاک )  
 ( '53 , ملت )  
 ( '51 , مراسم )  
 ( '50 , آزادی )  
 ( '48 , حکومت )  
 ( '46 , خودشون )  
 ( '42 , ♥ )  
 ( '42 , لعنت )  
 ( '41 , حجاب )  
 ( '41 , کاری )  
 ( '40 , داریوش )  
 ( '38 , غیرت )  
 ( '36 , نظام )  
 ( '36 , برن )  
 ( '34 , خون )  
 ( '34 , حقوق )  
 ( '33 , بگه )  
 ( '32 , اسلامی )  
 ( '32 , خودشو )  
 ( '32 , دخترت )  
 ( '31 , جمهوری )  
 ( '31 , ! )  
 ( '31 , با نوان )  
 ( '30 , دخترش )  
 ( '30 , علینژاد )

شکل ۱۰.۳: لیست کلمات پرتکرار صفحه‌ی سیاسی پس از حذف کلمات خنثی



## فصل ۴

# جمع‌بندی، نتیجه‌گیری، و پیشنهادها

## ۱.۴ جمع‌بندی و نتیجه‌گیری

گذراندن دوره‌ی کارآموزی تابستانی در شرکت مهندسی تحول‌آفرین برنا (متاب) تحت سرپرستی جناب آقای محمدمهدی کیانی و با همکاری آقای احسان شریفیان، ثمرات متعددی برای نگارنده‌ی این گزارش در بر داشته است. عناوین زیر، بخشی از مواردی است که نگارنده در طول این دوره آموخته و در آن‌ها صاحب تجربه شده است:

- زبان برنامه‌نویسی پایتون به صورت عمومی
  - کتابخانه‌های requests، bs4، و selenium از زبان پایتون جهت استخراج اطلاعات از صفحات وب
  - دیتابیس‌های خانواده‌ی SQL، و به طور خاص دیتابیس SQLite
  - فهم زبان برنامه‌نویسی HTML و استخراج اطلاعات از کدهای آن
- در کنار این موارد، انجام دو پروژه‌ی مبتنی بر استخراج اطلاعات از صفحات وب (Web Scraping) موجب شد تا کارآموز در این زمینه به مهارت قابل قبولی دست یابد.
- در پروژه‌ی اول، دیتابیس‌های کتب مذهبی با استفاده از اطلاعات آنلاین تهیه شدند، که به عنوان یکی از محصولات رایگان شرکت متاب مورد استفاده قرار خواهند گرفت.
- در پروژه‌ی دوم، یک کار گسترده و ارزشمند آغاز شد، اگر چه انجام آن فراتر یک دوره‌ی کوتاه کارآموزی است، اما ابزارهای مقدماتی آن در این دوره طراحی شدند. به طور کلی، این پروژه با هدف تحلیل داده‌های شبکه‌ی اجتماعی اینستاگرام است، تا در آینده به محصولی تبدیل شود که افراد بتوانند به کمک آن، صفحات مجازی مورد نیاز خود برای اموری نظیر تبلیغات را شناسایی کنند.

در این دوره‌ی کارآموزی که حدوداً ۲۴۰ ساعت به طول انجامید، می‌توان توزیع زمان را به صورت تقریبی به شکل زیر در نظر گرفت:

- یادگیری زبان پایتون به صورت عمومی: ۵۰ ساعت
- طراحی دیتابیس‌های کتب مذهبی: ۸۰ ساعت
- آشنایی با کتاب‌خانه‌ی selenium و انجام پروژه‌های تمرینی: ۳۵ ساعت
- پروژه‌ی استخراج و تحلیل داده‌های شبکه‌ی اجتماعی اینستاگرام: ۷۵ ساعت

در کل از نظر نگارنده، این دوره هم برای کارآموز و هم برای شرکت مفید بود، به این معنی که کارآموز توانست موارد ارزشمندی را بیاموزد و در آن‌ها کسب تجربه کند؛ و نهایتاً نتایجی را به شرکت تحویل داد که مطابق نظر آن بود و توانست بخشی از پروژه‌های جاری آن را به انجام رساند.

## ۲۰۴ پیشنهادها

یکی از مواردی که به ذهن می‌رسد، آن است که با توجه به نظامی که دانشجویان در آن تحصیل می‌کنند، به نظر می‌رسد که دانشجو تقریباً در هر زمینه‌ای که مشغول به کارآموزی شود، با توجه به عدم تجربه‌ی کاری پیشین، لازم است زمان قابل توجهی را صرف یادگیری مقدمات آن کار کند؛ به گونه‌ای که در فرصت محدود کارآموزی شاید امکان آن میسر نشود که کارآموز بتواند حجم قابل توجهی از کار را انجام دهد. نمونه‌ی آن در مورد نگارنده، پروژه‌ی دوم در همین دوره‌ی کارآموزی است که امکان آن بود که بسیار بیشتر روی آن کار شود، اما به دلیل کمبود زمان (و با توجه به این که حدود یک‌سوم از دوره‌ی کارآموزی صرف یادگیری ابزارها شده بود) ادامه دادن آن مقدور نبود. البته قطعاً بخش زیادی از ارزش دوره‌ی کارآموزی به همین یادگیری است، اما به نظر می‌رسد ناآمادگی دانشجویان برای ورود به محیط کار، باعث می‌شود در بسیاری از موارد نتیجه‌ی دوره‌ی کارآموزی نیز از نظر عملی چندان قابل اعتنا نباشد.

همچنین مورد دیگری که به نظر نگارنده می‌رسد، آن است که اگر دانشکده اقدام به فراهم آوردن بستر مناسب‌تری برای ارتباط دانشجویان با شرکت‌ها به منظور انتخاب محل کارآموزی کند، امید آن خواهد رفت که دوره‌های کارآموزی مفیدتری برای دانشجویان رقم بخورد. به عنوان مثال، به عنوان یک دانشجوی مهندسی برق، بسیاری از مواردی که به عنوان محل کارآموزی مورد بررسی و پیشنهاد قرار می‌گرفت، هیچ ارتباطی به رشته و علاقه‌ی نگارنده نداشت، و شاید در دسترس نبودن محیطی منطبق با رشته و گرایش و علاقه‌ی نگارنده، وی را بر آن داشت تا دوره‌ی کارآموزی خود را به انجام کاری بپردازد که چندان ارتباطی به رشته و گرایشش ندارد.

امید است که در آینده، تمهیداتی جهت افزایش بازدهی و ثمردهی دوره‌های کارآموزی اندیشیده شود.