## What is Django?

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

#### Django helps you write software that is:

* Complete
* Versatile
* Secure
* Scalable
* Maintainable
* Portable

**Where did it come from?**

Django was initially developed between 2003 and 2005 by a web team who were responsible for creating and maintaining newspaper websites. After creating a number of sites, the team began to factor out and reuse lots of common code and design patterns. This common code evolved into a generic web development framework which was open-sourced as the Django project in July 2005

[**How popular is Django?**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#how_popular_is_django)

There isn't any readily-available and definitive measurement of popularity of server-side frameworks (although you can estimate popularity using mechanisms like counting the number of GitHub projects and Stack Over flow questions for each platform). A better question is whether Django is "popular enough" to avoid the problems of unpopular platforms. Is it continuing to evolve? Can you get help if you need it? Is there an opportunity for you to get paid work if you learn Django?

## SO AGAINWhat is the Django Framework?

Django is an open-source framework for backend web applications based on Python one of the top web development languages. Its main goals are simplicity, flexibility, reliability, and scalability.

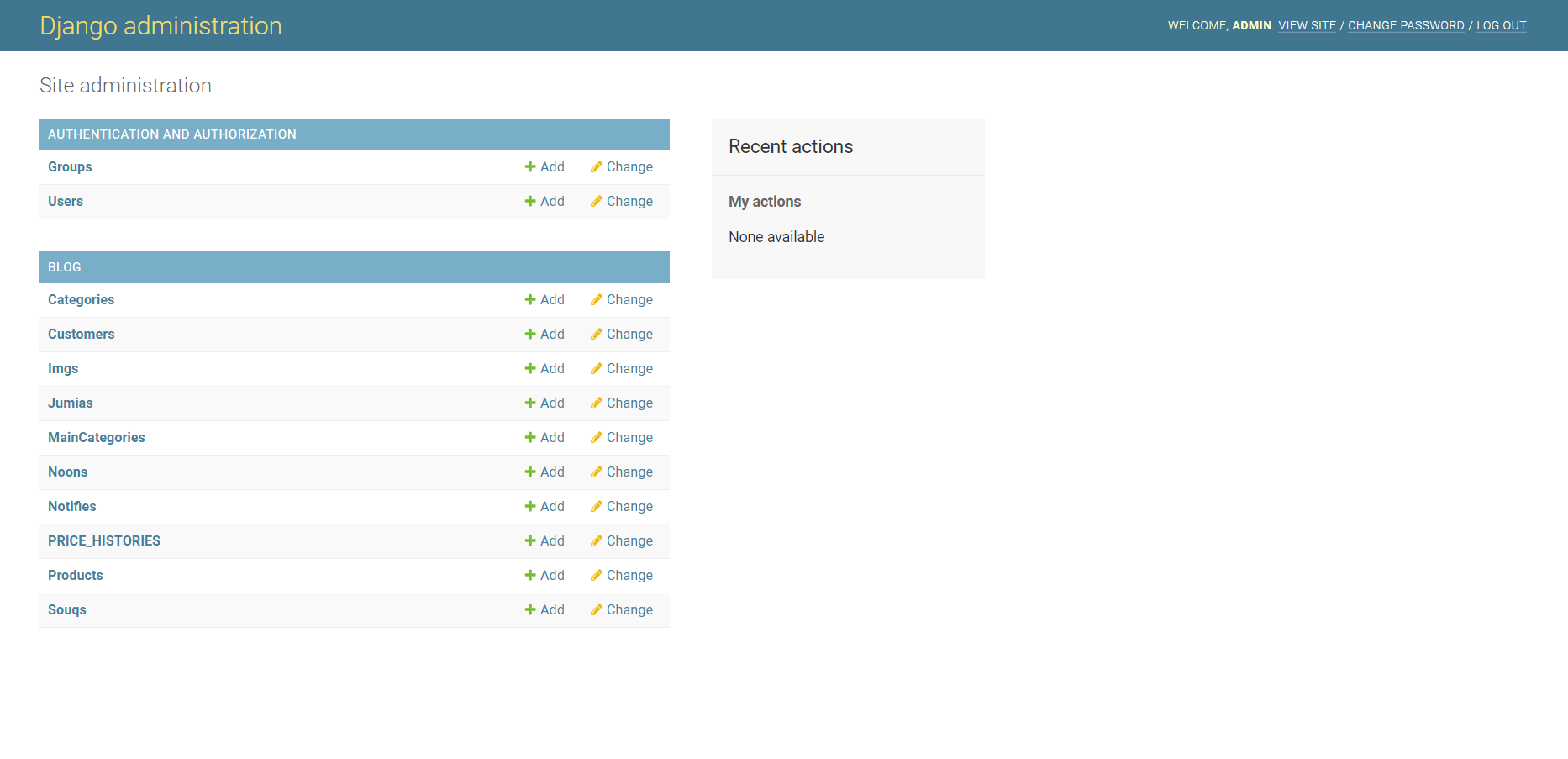
Django has its own naming system for all functions and components e.g., HTTP responses are called “views”. It also has an admin panel, which is deemed easier to work with than in Lavarel or Yii and other technical features, including:

* Simple syntax
* Its own web server
* MVC (Model-View-Controller) core architecture
* Batteries included (comes with all the essentials needed to solve solving common cases)
* An ORM (Object Relational Mapper)
* HTTP libraries
* Middleware support
* A Python unit test framework

## Why do you need a framework?

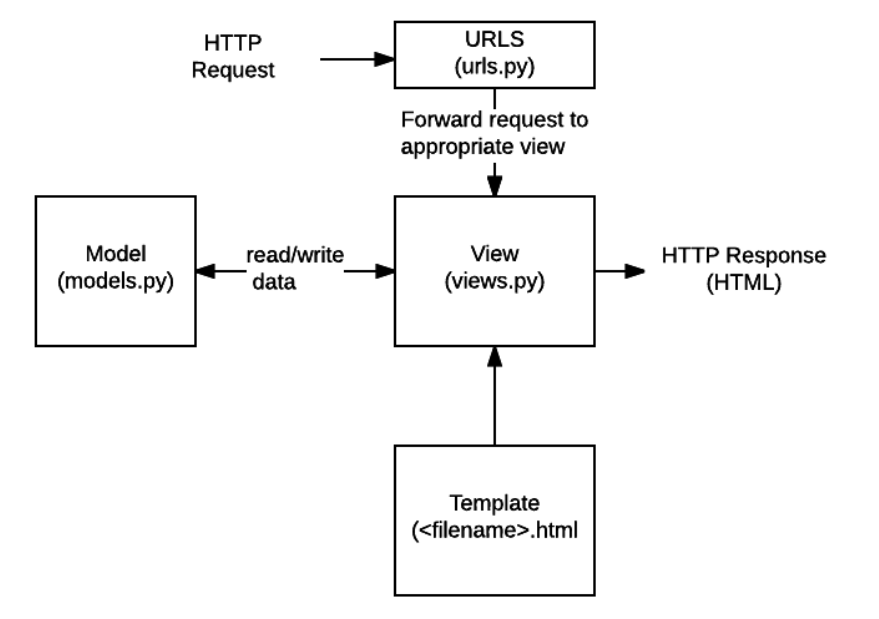
To understand what Django is actually for, we need to take a closer look at the servers. The first thing is that the server needs to know that you want it to serve you a web page.

Imagine a mailbox (port) which is monitored for incoming letters (requests). This is done by a web server. The web server reads the letter and then sends a response with a webpage. But when you want to send something, you need to have some content. And Django is something that helps you create the content.



## [What does Django code look like?](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#what_does_django_code_look_like)

In a traditional data-driven website, a web application waits for HTTP requests from the web browser or other client. When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.



## 

* **URLs**: While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
* **View**: A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via models, and delegate the formatting of the response to templates.
* **Models**: Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
* **Templates**: A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A view can dynamically create an HTML page using an HTML template, populating it with data from a model. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

## What happens when someone requests a website from your server?

When a request comes to a web server, it's passed to Django which tries to figure out what is actually requested. It takes a web page address first and tries to figure out what to do. This part is done by Django's **url resolver** (note that a website address is called a URL – Uniform Resource Locator – so the name *url resolver* makes sense). It is not very smart – it takes a list of patterns and tries to match the URL. Django checks patterns from top to bottom and if something is matched, then Django passes the request to the associated function which is called *view*.

In the *view* function all the interesting things are done we can look at a database to look for some information.

So instead of diving too much into details, we will start creating something with Django and we will learn all the important parts along the way

[**What else can you do?**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#what_else_can_you_do)

* **Forms**: HTML Forms are used to collect user data for processing on the server. Django simplifies form creation, validation, and processing.
* **User authentication and permissions**: Django includes a robust user authentication and permission system that has been built with security in mind.
* **Caching**: Creating content dynamically is much more computationally intensive (and slow) than serving static content. Django provides flexible caching so that you can store all or part of a rendered page so that it doesn't get re-rendered except when necessary.
* **Administration site:** The Django administration site is included by default when you create an app using the basic skeleton. It makes it trivially easy to provide an admin page for site administrators to create, edit, and view any data models in your site.
* **Serialising data:** Django makes it easy to serialise and serve your data as XML or JSON. This can be useful when creating a web service or when creating a website in which the client-side code handles all the rendering of data.

Leading market research firm, Gartner, suggests that [40% of all business initiatives lose value](https://www.data.com/export/sites/data/common/assets/pdf/DS_Gartner.pdf) because of incorrectly linked, or messy data. [Data deduplication](https://dataladder.com/data-deduplication-software/) and record linkage are two sides of the same coin. While the applications of both vary widely, the underlying techniques used to identify matching records for both [data cleansing](https://dataladder.com/data-cleansing-software/)/deduplication and record linkage are the same.

Whether you want to identify duplicates before migrating to a new CRM, or want to build a [Single Customer View](https://econsultancy.com/what-is-the-single-customer-view-and-why-do-you-need-it/) under an enterprise-wide digital transformation initiative, you will have to perform ‘data matching’: the ability to identify all records that point to the same entity within and across data sources. Easier said than done though.

In this blog, we will take an in-depth look at fuzzy matching, the go-to approach for data deduplication and record linkage. We will cover:

* Data Matching: Deterministic and Probabilistic Matching
* What is Fuzzy Matching?
* How to Minimize False Positives and Negatives
* Why Do Businesses Need Fuzzy Matching?
* Fuzzy Matching Techniques
* Example of Real-World Fuzzy Matching Scenario

**Data Matching: Deterministic and Probabilistic Matching**

We know we need to match records to identify duplicates and link records for [entity resolution](https://dataladder.com/entity-resolution-software/). But how exactly do we go about identifying matching records? What properties should we focus on?

Let’s start with ‘unique identifiers’. These are properties in the records you want to match that are unlikely to change over time, Customer Name for instance. You can assign weights to each property to improve your matching process. Think about it; if you are migrating customer data from one system to another and need to check for duplicates pre- and post-migration, you could, for instance, choose Name as the one unique identifier and phone number as the second. Now it’s just a matter of running a search for matching Customer IDs and phone numbers and you have all potential matches identified. That method is known as ‘deterministic data matching’.

Although effective in theory, the method is rarely used because of its inflexibility: The approach assumes that all entries are free of mistakes and standardized across systems – which is almost never the case in real-world linkage scenarios. In our previous example, if some phone numbers have country code in the ‘+1’ format and the rest start with ‘001’, the matching would go awry. That’s just one instance; there could potentially be dozens of different ways data could be entered. The names might be misspelled, acronyms used, middle name included, etc. In one system, a customer’s name could be ‘William Warner’ while another might have ‘Williaam Warner’ — it’s obvious that there’s been a small typo and both are in fact the same customer — but the method only allows discrete outcomes, that is, all or nothing.

How do you go about determining a match when so many variations exist?

By performing probabilistic data matching, that’s how. More commonly known as fuzzy matching’, this approach permits the user to account for variations like spelling errors, nicknames, punctuation differences, and many more by combining a variety of algorithms.

## What is fuzzy matching?

Rather than flagging records as a ‘match’ or ‘non-match’, fuzzy matching identifies the likelihood that two records are a true match based on whether they agree or disagree on the various identifiers.

The identifiers or parameters you choose here and the weight you assign forms the basis of fuzzy matching. If the parameters are too broad, you will find more matches, true, but you will also invariably increase the chances of ‘false positives’. These are pairs that are identified by your algorithm or [fuzzy matching software](https://dataladder.com/fuzzy-matching-software/) of choice as a match, but upon manual review, you will find that your approach identified a false positive.

Consider the strings “Kent” and “10th”. While there is clearly no match here, popular fuzzy matching algorithms still rate these two strings nearly 50% similar, based as character count and phonetic match. [Check for yourself](https://asecuritysite.com/forensics/simstring).

False positives are one of the biggest issues with fuzzy matching. The more efficient the system you’re using, the fewer the false positives. An efficient system will identify:

* Acronyms
* name reversal
* name variations
* phonetic spellings
* deliberate misspellings
* inadvertent misspellings
* abbreviations e.g. ‘Ltd’ instead of ‘Limited’
* insertion/removal of punctuation, spaces, special characters
* different spelling of names e.g. ‘Elisabeth’ or ‘Elizabeth’, ‘Jon’ instead of ‘John’
* shortened names e.g. ‘Elizabeth’ matches with ‘Betty’, ‘Beth’, ‘Elisa’, ‘Elsa’, ‘Beth’ etc.And many other variations.

**How to Minimize False Positives and Negatives**

We have discussed false positives in the previous section briefly. While they make matching more difficult by adding manual review time to the process, they’re not a genuine risk to the business because the system will flag false positives based on the overall match score. Let’s take a look at ‘false negatives’ now. This refers to matches that are missed altogether by the system: not just a low match score, but an absence of match score. This leads to a serious risk for the business as false negatives are never reviewed because no one knows they exist. Factors that commonly lead to false negatives include:

* Lack of relevant data
* Significant errors in data entry
* System limitations
* Match criterion is too narrow
* Inappropriate level of fuzzy matching

The most effective method to minimize both false positives and negatives is to profile and clean the data sources separately before you conduct matching. Leading [data matching solution](https://dataladder.com/data-matching-software/) providers typically bundle a data profiler that quickly provides enough metadata to construct a cogent profile analysis of data quality, as in missing values, lack of standardization, any other discrepancies in your data. By [profiling your data](https://dataladder.com/data-profiling/), you can quickly quantify the scope and depth of the primary project, whether it’s Master Data Management,  matching, cleansing, deduplication, or standardization.

Once you’ve profiled your data, you will know exactly which business rules to apply to clean and standardize your data most efficiently. You will also be able to quickly recognize and fill missing values, perhaps by purchasing 3rd party data.

Cleaner, more complete data reduces false positives and negatives significantly by increasing match accuracy because your data is now standardized. The fuzzy matching algorithms you use, the matching criteria you define, the weight you assign to different parameters, the way you combine different algorithms and assign priority – these are all important factors in minimizing false positives and negatives too. But none of these are going to help much if you haven’t profiled and cleaned your data first. See how DataMatch Enterprise has helped 4,000+ customers in over 40 countries clean, deduplicate, and link their data efficiently.

**Why Do Businesses Need Fuzzy Matching**

Research reveals that 94% of businesses admit to having duplicate data, and the majority of these duplicates are non-exact matches and therefore usually remain undetected. Fuzzy matching software helps you make those connections automatically using sophisticated proprietary matching logic, regardless of spelling errors, unstandardized data, or incomplete information.

But it’s not just about deduplication. From a strategic perspective, fuzzy matching comes into play when you’re conducting record linkage or entity resolution. We touched upon this briefly in the previous section too; the fuzzy matching approach is invaluable when creating a Single Source of Truth for business analytics or building a foundation for Master Data Management (MDM), helping organizations integrate data from dozens of different sources across the enterprise while ensuring accuracy and minimizing manual review. See how [a major healthcare provider](https://dataladder.com/resources/case-studies/st-john-associates/) was able to save hundreds of man-hours annually.

Here are some ways that fuzzy matching is used to improve the bottom-line:

* Realize a Single Customer View
* Work with Clean Data You Can Trust
* Prepare Data for Business Intelligence
* Enhance the Accuracy of Your Data for Operational Efficiency
* Enrich Data for Deeper Insights
* Ensure Better Compliance
* Refine Customer Segmentation
* Improve Fraud Prevention

## Fuzzy Matching Techniques

Now you know what fuzzy matching is and the many different ways you can use it to grow your business. Question is, how do you about implementing fuzzy matching processes in your organization?

Here’s a list of the various fuzzy matching techniques that are in use today:

* Levenshtein Distance (or Edit Distance)
* Damerau-Levenshtein Distance
* Jaro-Winkler Distance
* Keyboard Distance
* Kullback-Leibler Distance
* Jaccard Index
* Metaphone 3
* Name Variant
* Syllable Alignment
* Acronym

**Example of a Real-World Fuzzy Matching Scenario**

The following example shows how record linkage techniques can be used to detect fraud, waste or abuse of federal government programs. Here, two databases were merged to get information not previously available from a single database.

A database consisting of records on 40,000 airplane pilots licensed by the U.S. Federal Aviation Administration (FAA) and residing in Northern California was matched to a database consisting of individuals receiving disability payments from the Social Security Administration. Forty pilots whose records turned up on both databases were arrested.

A prosecutor in the U.S. Attorney’s Office in Fresno, California stated, according to an AP report:

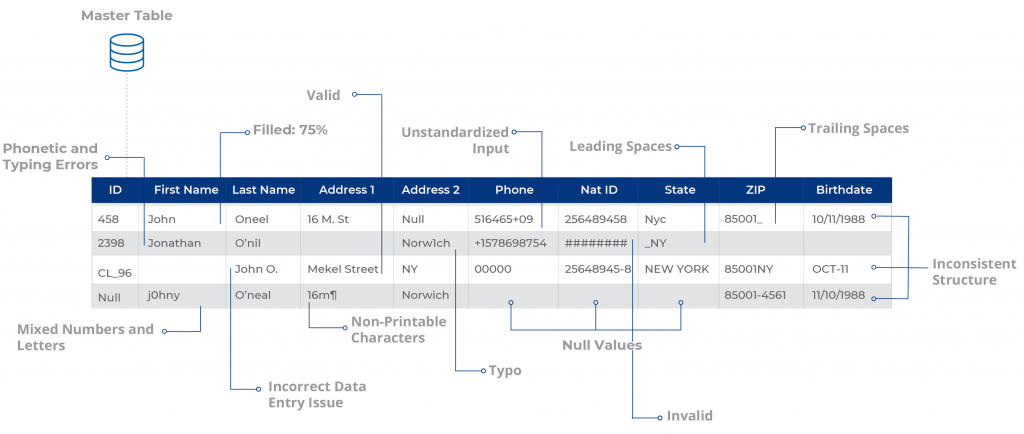
“There was probably criminal wrongdoing.” The pilots were either lying to the FAA or wrongfully receiving benefits. The pilots claimed to be medically fit to fly airplanes. However, they may have been flying with debilitating illnesses that should have kept them grounded, ranging from schizophrenia and bipolar disorder to drug and alcohol addiction and heart conditions.”

At least twelve of these individuals “had commercial or airline transport licenses,” the report stated. The FAA revoked 14 pilots’ licenses. The other pilots were found to be lying about having illnesses in order to collect Social Security payments.

The quality of the linkage of the files was highly dependent on the quality of the names and addresses of the licensed pilots within both of the files being linked. The detection of the fraud was also dependent on the completeness and accuracy of the information in a particular Social Security Administration database.

## Fuzzy Matching Made Easy, Fast, and Laser-Focused on Driving Business Value

Traditionally, fuzzy matching has been considered a complex, arcane art, where project costs are typically in the hundreds of thousands of dollars, taking months, if not years, to deliver tangible ROI, and even then, security, scalability, and accuracy concerns remain. That is no longer the case with modern data quality software. Based on decades of research and 4,000+ deployments across more than 40 countries, [DataMatch Enterprise](https://dataladder.com/products/datamatch-enterprise/" \t "_blank) is a highly visual data cleansing application specifically designed to resolve data quality issues. The platform leverages multiple proprietary and standard algorithms to identify phonetic, fuzzy, miskeyed, abbreviated, and domain-specific variations.



Build scalable configurations for deduplication & [record linkage](https://dataladder.com/record-linkage-software/), suppression, enhancement, extraction, and [standardization](https://dataladder.com/data-standardization-software/) of business and customer data and create a Single Source of Truth to maximize the impact of your data across the enterprise.

How to Use It Correctly

As we have just defined **Fuzzy matching** allows you to identify non-exact matches of your target item. It is the foundation stone of many search engine frameworks and one of the main reasons why you can get relevant search results even if you have a typo in your query or a different verbal tense.

As you might expect, there are many algorithms that can be used for fuzzy searching on text, but virtually all search engine frameworks (including bleve) use primarily the Levenshtein Distance for fuzzy string matching:

#### Levenshtein Distance

Also known as **Edit Distance**, it is the number of transformations (deletions, insertions, or substitutions) required to transform a source string into the target one. For example, if the target term is “book” and the source is “back”, you will need to change the first “o” to “a” and the second “o” to “c”, which will give us a Levenshtein Distance of 2.Edit Distance is very easy to implement, and it is a popular challenge during code interviews (You can find Levenshtein implementations in JavaScript, Kotlin, Java, and many others here).

Additionally, some frameworks also support the Damerau-Levenshtein distance:

#### Damerau-Levenshtein distance

It is an extension to Levenshtein Distance, allowing one extra operation: Transposition of two adjacent characters:

**Ex**: TSAR to STAR

**Damerau-Levenshtein distance** = 1 (Switching S and T positions cost only one operation)

**Levenshtein distance** = 2 (Replace S by T and T by S)

**Fuzzy matching and relevance**

Fuzzy matching has one big side effect; it messes up with relevance. Although Damerau-Levenshtein is an algorithm that considers most of the common user’s misspellings, it also can include a significant number of false positives, especially when we are using a language with an average of just 5 letters per word, such as English. That is why most of the search engine frameworks prefer to stick with Levenshtein distance.

What Is Web Scraping?

**Web scraping** is the process of gathering information from the Internet. Even copy-pasting the lyrics of your favorite song is a form of web scraping! However, the words “web scraping” usually refer to a process that involves automation. Some websites don’t like it when automatic scrapers gather their data, while others don’t mind.

If you’re scraping a page respectfully for educational purposes, then you’re unlikely to have any problems. Still, it’s a good idea to do some research on your own and make sure that you’re not [violating](https://benbernardblog.com/web-scraping-and-crawling-are-perfectly-legal-right/) any Terms of Service before you start a large-scale project. To learn more about the legal aspects of web scraping, check out [Legal Perspectives on Scraping Data From The Modern Web](https://www.lawinsociety.org/legal-perspectives-on-scraping-data-from-the-modern-web).

**Why Scrape the Web?**

Say you’re a surfer (both online and in real life) and you’re looking for employment. However, you’re not looking for just any job. With a surfer’s mindset, you’re waiting for the perfect opportunity to roll your way!

There’s a job site that you like that offers exactly the kinds of jobs you’re looking for. Unfortunately, a new position only pops up once in a blue moon. You think about checking up on it every day, but that doesn’t sound like the most fun and productive way to spend your time.

Thankfully, the world offers other ways to apply that surfer’s mindset! Instead of looking at the job site every day, you can use Python to help automate the repetitive parts of your job search. **Automated web scraping** can be a solution to speed up the data collection process. You write your code once and it will get the information you want many times and from many pages.

In contrast, when you try to get the information you want manually, you might spend a lot of time clicking, scrolling, and searching. This is especially true if you need large amounts of data from websites that are regularly updated with new content. Manual web scraping can take a lot of time and repetition.

There’s so much information on the Web, and new information is constantly added. Something among all that data is likely of interest to you, and much of it is just out there for the taking. Whether you’re actually on the job hunt, gathering data to support your grassroots organization, or are finally looking to get all the lyrics from your favorite artist downloaded to your computer, automated web scraping can help you accomplish your goals.

**Challenges of Web Scraping**

The Web has grown organically out of many sources. It combines a ton of different technologies, styles, and personalities, and it continues to grow to this day. In other words, the Web is kind of a hot mess! This can lead to a few challenges you’ll see when you try web scraping.One challenge is **variety**. Every website is different. While you’ll encounter general structures that tend to repeat themselves, each website is unique and will need its own personal treatment if you want to extract the information that’s relevant to you. Another challenge is **durability**. Websites constantly change. Say you’ve built a shiny new web scraper that automatically cherry-picks precisely what you want from your resource of interest. The first time you run your script, it works flawlessly. But when you run the same script only a short while later, you run into a discouraging and lengthy stack of [tracebacks](https://realpython.com/python-traceback/)!This is a realistic scenario, as many websites are in active development. Once the site’s structure has changed, your scraper might not be able to navigate the sitemap correctly or find the relevant information. The good news is that many changes to websites are small and incremental, so you’ll likely be able to update your scraper with only minimal adjustments.However, keep in mind that because the internet is dynamic, the scrapers you’ll build will probably require constant maintenance. You can set up [continuous integration](https://realpython.com/python-continuous-integration/) to run scraping tests periodically to ensure that your main script doesn’t break without your knowledge.

**APIs: An Alternative to Web Scraping**

Some website providers offer **Application Programming Interfaces (APIs)** that allow you to access their data in a predefined manner. With APIs, you can avoid parsing HTML and instead access the data directly using formats like [JSON](https://realpython.com/courses/working-json-data-python/) and XML. HTML is primarily a way to visually present content to users.When you use an [API](https://realpython.com/python-api/), the process is generally more stable than gathering the data through web scraping. That’s because APIs are made to be consumed by programs, rather than by human eyes. If the design of a website changes, then it doesn’t mean that the structure of the API has changed.to APIs just as they do to websites. Additionally, it’s much harder to inspect the structure of an API by yourself if the provided documentation is lacking in quality.The approach and tools you need to gather information using APIs are outside the scope of this tutorial. To learn more about it, check out [API Integration in Python](https://realpython.com/api-integration-in-python/).

**Requests - Overview**

Requests is a HTTP library that provides easy functionality to deal with http request/response in your web application. The library is developed in python.

***Features of Requests***

**Request:** The python requests library has easy to use methods available to handle Http request. Passing of parameters and handling the request type like GET, POST, PUT, DELETE, etc. is very easy.

**Response:** You can get the response in the format you need and the supported ones are text format, binary response, json response, and raw response.

**Headers:** The library allows you to read, update or send new headers as per your requirements.

**Timeouts:** Timeouts can be easily added to the URL you are requesting using python requests library. It so happens that you are using a third-party URL and waiting for a response.It is always a good practice to give a timeout on the URL as we might want the URL to respond within that timeout with a response or an error that is coming because of timeout. Not doing so can cause either to wait on that request indefinitely.

**Error handling:** The requests module gives support for error handling and some of which are Connection Error, Timeout errors, TooManyRedirects, Response.raise\_for\_status errors, etc.

**Cookies:** The library allows you to read, write and update for the requested URL.

**Sessions:** To maintain the data, you require between requests you need sessions. So, if the same host is called again and again you can re-use the TCP connection which in turn will improve the performance.

**SSL certificates:**SSL certificate is a security feature that comes with secure urls. When you use Requests, it also verifies SSL certificates for the https URL given. SSL Verification is enabled by default in the requests library and will throw an error if the certificate is not present.

**Authentication:**HTTP authentication is on the server-side asking for some authentication information like username, password when the client requests a URL. This is an additional security for the request and the response being exchanged between the client and the server.

***Advantages of using Python Requests Library***

Following are the advantages of using Python Requests Library −

* Easy to use and fetch the data from the URL given.
* Requests library can be used to scrape the data from the website.
* Using requests, you can get, post, delete, update the data for the URL given.
* The handling of cookies and session is very easy.
* The security is also taken care of the help of authentication module support.

***Requests - Environment Setup***

we will work on the installation of Requests. To start working with the Requests module, we need to install Python first.

* Install Python
* Install Requests

***Installing Python***

Go to the Python official site: <https://www.python.org/downloads/> as shown below and click on the latest version available for Windows, Linux/Unix, and Mac OS. Download Python as per your 64 or 32 bit OS available with you. Once you have downloaded, click on the .exe file and follow the steps to install python on your system.

The python package manager, i.e., pip will also get installed by default with the above installation. To make it work globally on your system, directly add the location of python to the PATH variable. The same is shown at the start of the installation to remember to check the checkbox which says ADD to PATH. In case you forget to check it, please follow the below-given steps to add to PATH.

To add to PATH follow the steps−Right-click on your Computer icon and click on properties > Advanced System Settings.

* Click on Environment Variables as shown above. It will display the screen as shown below
* Select Path and click on Edit button, add the location path of your python at the end. Now, let us check the python version.

***Checking the python version***

E:\prequests>python --version

Python 3.7.3

***Requests - How Http Requests Work?***

Python’s Requests is a HTTP library that will help us exchange data between the client and the server. Consider you have a UI with a form, wherein you need to enter the user details, so once you enter it, you have to submit the data which is nothing but a Http POST or PUT request from the client to server to save the data.

When you want the data, you need to fetch it from the server, which is again a Http GET request. The exchange of data between the client when it requests the data and the server responding with the required data, this relationship between the client and the server is very important.

The request is made to the URL given and it could be a secure or non-secure URL.

The request to the URL can be done using GET, POST, PUT, DELETE. The most commonly used is the GET method, mainly used when you want to fetch data from the server.

Using the request library, the URL is called as follows using a string dictionary.

Wherein the data to the URL is sent as a dictionary of strings. If you want to pass id=9 and username=Delphine, you can do as follows−

payload = {'id': '9', 'username': 'Delphine'}

The requests library is called as follows−

res = requests.get('[https://www.noon.com/egypt-en/'](https://jsonplaceholder.typicode.com/users), params=payload')

*Using POST*

res = requests.post(‘https://www.jumia.com.eg/', data = {'id':'9', 'username':'Delphine'})

*Using PUT*

res = requests.put(‘https://egypt.souq.com/eg-ar/’, data = {'id':'9', 'username':'Delphine'})

The response from the Http request can be in text encoded form, binary encoded, json format or raw response. The details of the request and response are explained in detail in the next chapters.

***Requests - Working with Requests***

* Making HTTP Requests.
* Passing Parameters to HTTP Requests.

***Making HTTP Requests***

To make a Http request, we need to first import the request module as shown below−

import requests

Let us now see, how to make a call to URL using the requests module.

Example

url = "https://www.noon.com/\_svc/catalog/api/search"

response = requests.request("POST", url, headers=headers, json=payload).json()

w=response.get('hits')

print(response.status\_code)

The url− https://www.noon.com/\_svc/catalog/api/search is called using requests.get() method. The response object of the URL is stored in the(w) variable. When we print the variable, it gives the 200 response code, which means that we have got the response successfully.

*Output*

E:\prequests>python makeRequest.py

<Response [200]>

Handling Response for HTTP Requests

In this part, we will get into more details of the response received from the requests module. We will discuss the following details−

* Getting Response
* JSON Response
* RAW Response
* Binary Response

Getting Response

We will make a request to the URL using request.get() method.

  payload={"brand":[],"category":l,"filterKey":[],"f":{},"sort":{"by":"popularity","dir":"desc"},"limit":50,"page":page\_number}

  response = requests.request("POST", url, headers=headers, json=payload)

  if(response.status\_code==429):

    print('error 429')

    time.sleep(5)

    w=Datapayload()

  else:

    response=response.json()

    w=response.get('hits')

JSON Response

You can also get the response for the Http request in json format by using response.json()

import requests

import time

import psycopg2

def des(sku,offer\_code):

    url1 = f"https://www.noon.com/\_svc/catalog/api/u/{sku}/p?o={offer\_code}"

    response1 = requests.request("GET", url1, headers=headers1, data=payload1)

    if(response1.status\_code==429):

      print('error 429')

      time.sleep(20)

      a=des(sku,offer\_code)

    else:

      response1=response1.json()

      a=response1['product']['feature\_bullets']

    return(a)

Requests - HTTP Requests Headers

**Custom Headers**

You can also send headers to the URL being called as shown below.

**Example**

import requests

import time

import psycopg2

payload1={}

headers1 = {

  'authority': 'www.noon.com',

  'pragma': 'no-cache',

  'cache-control': 'no-cache, max-age=0, must-revalidate, no-store',

  'x-locale': 'en-eg',

  'x-content': 'mobile',

  'x-mp': 'noon',

  'user-agent': 'Mozilla/5.0 (iPhone; CPU iPhone OS 13\_2\_3 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.0.3 Mobile/15E148 Safari/604.1',

  'x-cms': 'v2',

  'accept': 'application/json, text/plain, \*/\*',

  'x-platform': 'web',

  'sec-fetch-site': 'same-origin',

  'sec-fetch-mode': 'cors',

  'sec-fetch-dest': 'empty',

  'referer': 'https://www.noon.com/egypt-en/iphone-12-pro-max-with-facetime-256gb-graphite-5g-middle-east-version/N41044056A/p?o=e0211e466e4a3c6a',

  'accept-language': 'en-US,en;q=0.9',

}

def des(sku,offer\_code):

    url1 = f"https://www.noon.com/\_svc/catalog/api/u/{sku}/p?o={offer\_code}"

    response1 = requests.request("GET", url1, headers=headers1, data=payload1)

    if(response1.status\_code==429):

      print('error 429')

      time.sleep(20)

      a=des(sku,offer\_code)

    else:

      response1=response1.json()

      a=response1['product']['feature\_bullets']

    return(a)

The headers passed has to be string, bytestring, or Unicode format. The behavior of the request will not change as per the custom headers passed.

Response Headers

The response headers look like below when you check the URL in the browser developer tool, network tab−To get the details of the headers from the requests module use.

*Example*

lists=[['electronics-and-mobiles/mobiles-and-accessories'],['electronics-and-mobiles/computers-and-accessories'],['electronics-and-mobiles/video-games-10181'],['electronics-and-mobiles/television-and-video'],['electronics-and-mobiles/camera-and-photo-16165'],['electronics-and-mobiles/portable-audio-and-video'],['electronics-and-mobiles/wearable-technology'],['electronics-and-mobiles/home-audio'],['electronics-and-mobiles/accessories-and-supplies']]

for l in lists:

  l=l

  page\_number=1

  a=True

  while a :

    url = "https://www.noon.com/\_svc/catalog/api/search"

    payload="{\"brand\":[\"apple\"],\"category\":[\"electronics-and-mobiles/mobiles-and-accessories/mobiles-20905\"],\"filterKey\":[],\"f\":{},\"sort\":{\"by\":\"popularity\",\"dir\":\"desc\"},\"limit\":50,\"page\":2}"

    headers = {

    'authority': 'www.noon.com',

    'pragma': 'no-cache',

    'cache-control': 'no-cache, max-age=0, must-revalidate, no-store',

    'x-locale': 'en-eg',

    'x-content': 'mobile',

    'x-mp': 'noon',

    'x-platform': 'web',

    'x-cms': 'v2',

    'content-type': 'application/json',

    'accept': 'application/json, text/plain, \*/\*',

    'user-agent': 'Mozilla/5.0 (iPhone; CPU iPhone OS 13\_2\_3 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.0.3 Mobile/15E148 Safari/604.1',

    'origin': 'https://www.noon.com',

    'sec-fetch-site': 'same-origin',

    'sec-fetch-mode': 'cors',

    'sec-fetch-dest': 'empty',

    'referer': 'https://www.noon.com/egypt-en/electronics-and-mobiles/mobiles-and-accessories/mobiles-20905/apple',

    'accept-language': 'en-US,en;q=0.9,ar;q=0.8'}

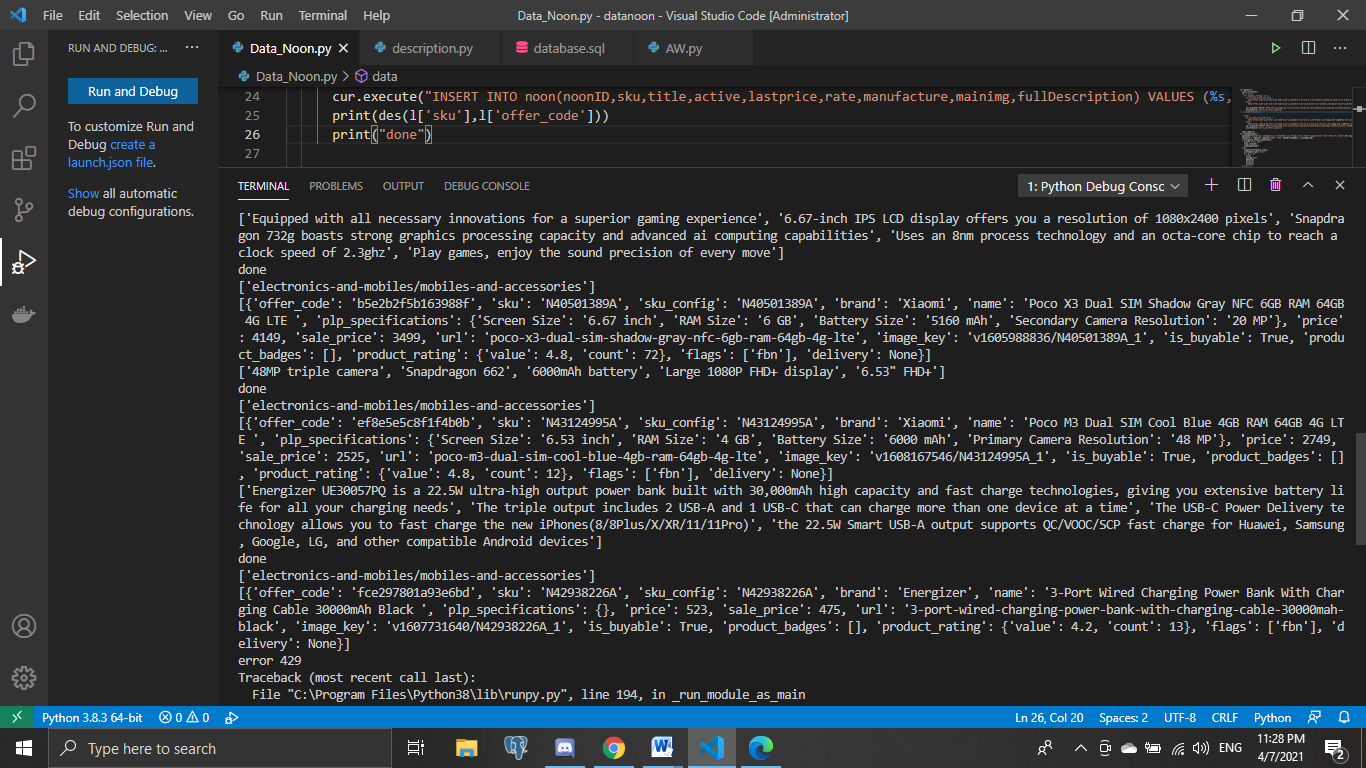
    Datapayload()

    print(page\_number)

    page\_number+=1

Requests - Handling GET Requests

This chapter will concentrate more on the GET requests, which is the most common and used very often. The working of GET in the requests module is very easy. Here is a simple example about working with the URL using the GET method.



Oop

Introduction to Object-oriented programming

In order to begin understanding the intuition behind this programming technique, let’s take a look at an initial example. Imagine that you have to describe a car to someone who’s never seen one before, how would you do it?

You might want to start saying that it’s a wheeled motor vehicle used for transportation. Also, you might say that there are several brands of car-maker companies, which make different types of cars that fulfill various needs. In an even more basic level you might say that it has four tires, that they carry up to five people in most cases and that they are mainly used to transport people, not goods.

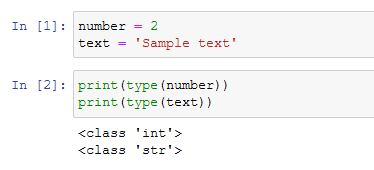
When explaining to a computer what kind of object this is, it’s a good idea to approach it in a similar way. Computers have no innate idea of what a car is, why were they created or who uses them. And if you want your computer to correctly understand the object, a car in this case, you have to clearly explain which are its attributes.

To make it easier for computers to understand these new concepts, Python uses a programming pattern called **object-oriented programming**, which models concepts using classes and objects. This is a flexible, powerful paradigm where classes represent and define concepts, while **objects are instances of** **classes**. In the car example, we can create a class called car that defines its characteristics to the computer. We would be able to create thousands of instances of that class car, which are the individual objects of that class.

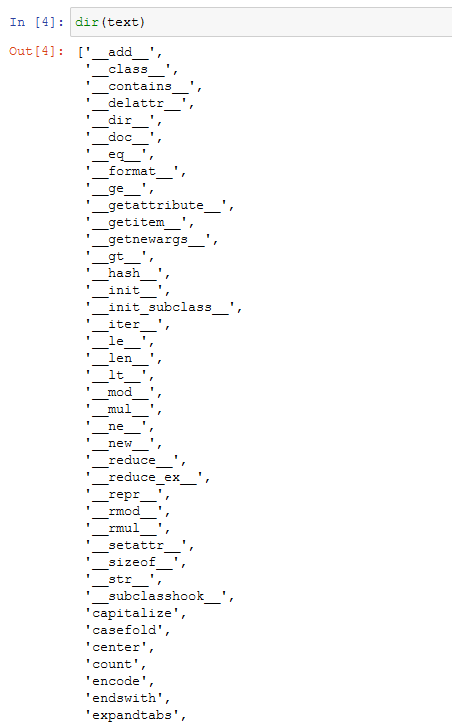
The idea of object-oriented programming might sound abstract and complex, but if you’ve programmed any software you might have already used objects without even realizing it. **Almost everything in Python is an object**, all of the numbers, strings, lists, and dictionaries are included in this type of element. The core concept of object-oriented programming comes down to **attributes**and **methods**associated with a type:

* **Attributes**are the characteristics associated to a type.
* **Methods**are the functions associated to a type.

In the car example, color and brand are two attributes associated with every instance, or car, created with the program. On the other hand, methods would be actions performed with or by the car, such as driving. A more computer-oriented example would be a file in a directory, as every file has a name, a size and a date of when it was created.



As you can see in the image above, when we use the **type**function as we just did here, Python tells us which class the value or variable belongs to. And since **integers**and **strings**are classes, they have a bunch of attributes and methods associated with them. You can access attributes and methods of a class with the **dir** function in Python, as shown below:

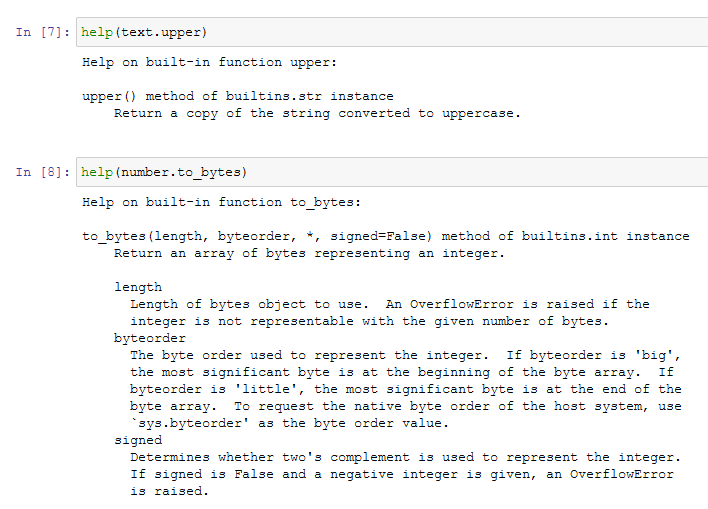


different **instance**of the string class. This means that they all have the same methods, although the content in the string is different.

## Why are there a bunch of methods that begin and end with double underscores?

These are called **special methods**and theyaren’t usually called by those weird names. Instead, they’re called by some of the internal Python functions. for example, the **\_\_len\_\_** method is called by the **len**function.

If you want to know what a specific method does, you should use the **help** function:

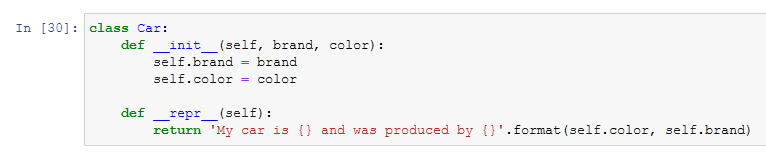


When we use the **help**function on any variable or value, we’re accessing all the documentation for the corresponding class. In this case, we’re looking at the documentation for the **str**and the **int** class.

Although Python comes with a lot of predefined classes for us, the power of **object-oriented programming**comes when we define our own classes with their own attributes and methods.

Defining a New Class

As mentioned earlier, the point of object oriented programming is to help define a real-world concept in a way that the computer understands. Let’s get hands on to build a new class with the car example:



Let’s clarify specific elements of the code:

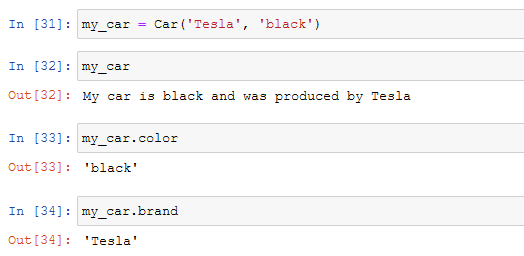
* In Python, we use the **class**reserved keyword to tell the computer that we’re starting a new class. We follow this with the name of the class and a colon. Python style guidelines recommend that class names should start with a **capital letter**. In my case, the class is called Car.
* After the line with the class definition comes the body of the class, which is indented to the right, following the pattern of [loops or functions](https://towardsdatascience.com/understand-loops-in-python-with-one-article-bace2ddba789?source=your_stories_page-------------------------------------).
* We’ll get to the special methods **init**and **repr** in the fourth section of this article.

**How might we expand our definition of the Car class?**It would probably have the same attributes that represent the information we want to associate with a car like brand and color.

Now, let’s proceed with the creation of an instance of the class Car, and assign it to a variable called “my\_car”. To create a new instance of any class, we call the name of the class as if it were a function:

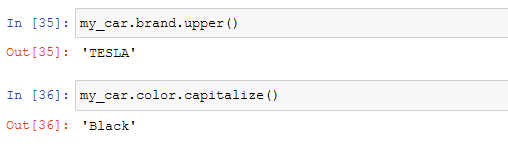


As you can see, I’m passing as an argument the brand and the color as I’ve configured my class to require both items in the creation of a new object of the class. As a result, we can call the attributes of the created instance and receive the value previously assigned:

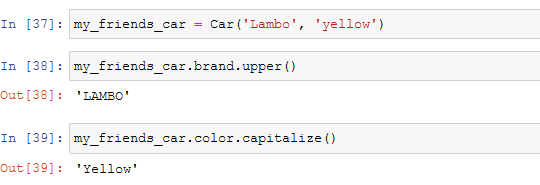


The syntax used to access the attributes is called **dot notation**because of the dot used in the expression. Dot notation lets you access any of the abilities that the object might have, such as brand or color.

The attributes and methods of some objects can be other objects and can have attributes and methods of their own. For example, we could use the upper or the capitalize methods to modify both string attributes of my instance:



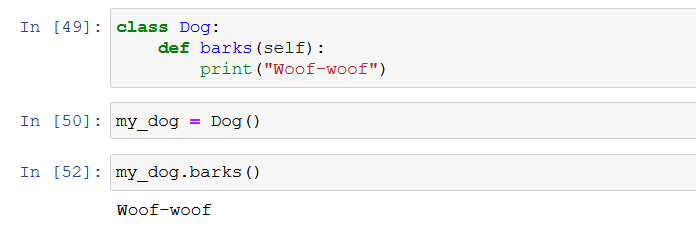
So far we’ve created one instance of the Car class and set its attributes. Now, we could create a new instance of the the class with different attributes:



Instance Methods

Methods are essentially called to make objects do stuff. For example, **upper** and **capitalize**for strings. The key to learn the intuition of methods in O.O.P. is to understand that **methods are functions that operate on the attributes of a specific instance of a class**. When we call the **lower**method on the string, we’re making the contents of that specific string lowercase.

Let’s take a closer look by creating a new class called **Dog**and defining our own methods. First, we need to define the class and create an instance of it like we’ve done before with the Car class. While my dog might be great, it can’t perform any actions as long as I don’t define methods for them. Take a look at the example:



As shown in the image, we must start defining a method with the **def**keyword just like we would for a function, and indent to the right the body of the method, also as we would for a function.

The function is receiving a parameter called “self”. This parameter represents the instance that the method is being executed on.

Even though my dog barks, it will always do it in the same way. We can change how it will bark with a simple modification in the code, in order to gain flexibility in the attributes and methods that we configure to our classes:



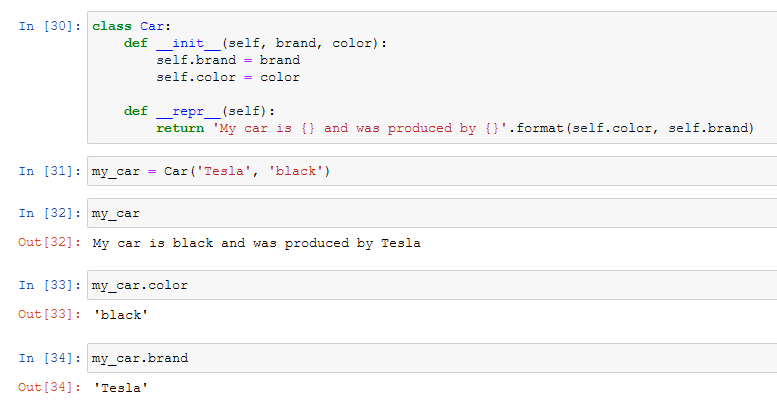
Defining Constructors and Other Special Methods

Both classes created up to this paragraph contain default values as attributes and methods. This is not an ideal scenario as it creates redundant code for each attribute, and more importantly, **as it makes it really easy to forget to set an important value**.

So, when writing code it's a good idea to set attributes and methods that will vary with instances upon creating the class, in order to make sure that each instance contains the same important attributes. To do this, **we need to use a special method called Constructor**.

*The constructor of the class is the method that's called when you call the name of the class. It's always named****init****.*

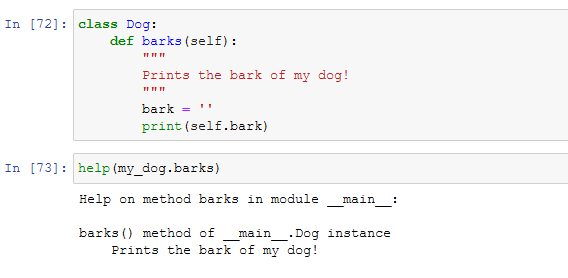
It contains the keyword self, which refers to the instance being created, and the arguments that will be passed as attributes by the programmer once the instance is created, like we do in the example below:



The second method of the class is the **repr**method, which tells Python to print a predetermined statement every time an instance of the class is called, as in the image above.

**Want to know which is a specific method’s function?** Refer to the **help** function introduced previously. As built-in classes include a guide to help users understand the intuition behind each method or attribute, we could also do this on our own classes, methods, and functions. We can do that by adding a **Docstring**.

*A Docstring is a brief text that explains what something does.*



Once you include documentation to your classes and objects, you’ll get much more information about the methods created that will facilitate re-usability of the code and help other users to understand it. Remember that the docstring always has to be indented at the same level of the block it’s documenting.

Code Reuse

Another important aspect of object-oriented programming is **Inheritance**. Just like people have parents, grandparents, and so on, objects have an ancestry. The principle of inheritance lets a programmer build relationships between concepts and group them together. In particular, this allows us to reduce code duplication by generalizing our code.

For example, how can we define a representation of “**other means of transport”** apart from the car that we already created, or other pets apart from my dog. This grouping characteristic allows us to create other classes that share some of the attributes of the existing classes, but not all of them, in order to add other similarinstances without rewriting existing code.

In Python, we use parentheses in the class declaration to show an inheritance relationship. In the transportation example above, I’ve used Python’s syntax to tell the computer that both the car and the train inherit from the MyTransports class. Because of this, they automatically have the same constructor, which sets the color and brand attributes.

With the inheritance technique, we can use the transportation class to store information that applies to all kinds of transports available, and keep car or train specific attributes in their own classes. You can think of the MyTransports class as the parent class, and the Car and Train classes as siblings.

An example that is closer to the tasks you would perform in an IT department would be a system that handles the employees at your company. You may have a class called employee, which could have the attributes for things like full name of the person, the username used in company systems, the groups the employee belongs to, and so on. The system could also have a manager class which also is an employee, but additional information associated with it, like the employees that report to a specific manager.