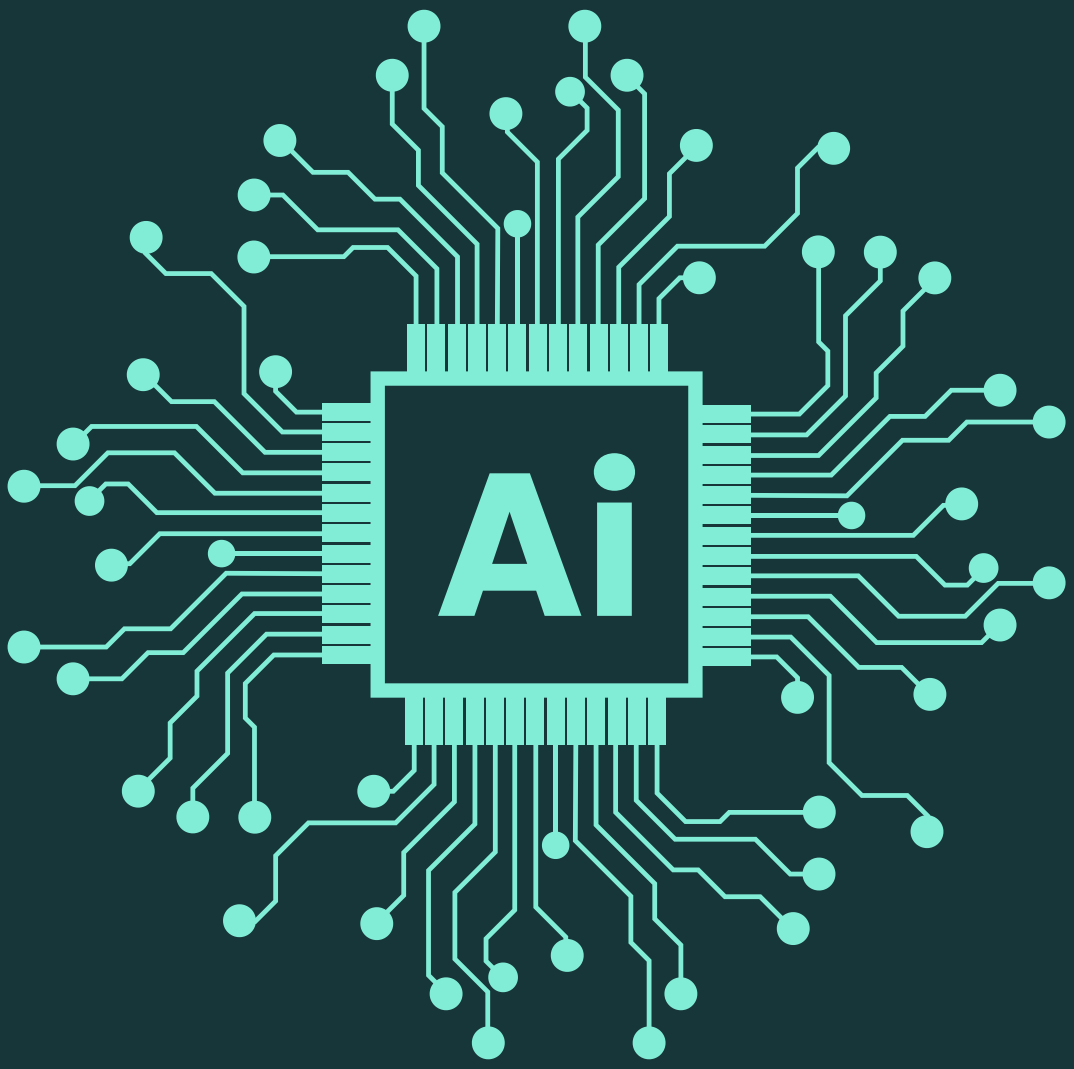
FAWWAZ ADIL

PYTHON FOR AI

INTRODUCTORY NOTES FOR BEGINNERS



DISCLAIMER:

These notes are based solely on the "AI For Everyone: Python for Beginners" course by Andrew Ng on Coursera (DeepLearning.AI).

- These notes are intended for educational and personal reference purposes only.
- The content covered follows the original course syllabus, focusing on Python for Artificial Intelligence.
- These are hand-written notes created by me to make learning easier and more intuitive.
- These notes are not officially affiliated with or endorsed by Andrew Ng, Coursera, or DeepLearning.AI.
- For hands-on implementation and exercises, visit: this link and enroll and practice for free.

Basics of AI Python

Date: 7TH JUNE '25

Coding

Python: used in:

- Self-Driving cars,
- Chatbots
- Smart Agriculture
- Etc

- JUPYTER NOTEBOOK:

  Run the program: SHIFT + ENTER

Data Types

Strings :-

i) Single quotation strings: "Hello, world!"

ii) Multi Line strings: print(""" Hello, world!
It's great to be here! """)

- Checking data type in Python:

  type function is used: type("Hello, world!")

  OUTPUT: str

```
type("""
    Hello, there
""")
```

OUT: str

Date:

Representing Numbers:

- INTEGERS: 9, 0, -1
- FLOATS: 2.99, 3.01

Python can be used as a calculator:

Print (100 + 45) [You can perform all operations like this]

OUT: 145

To the power of: **

so, print (3 ** 2)

would OUT: 9

- Use paranthesis to correct order of operation.

e.g: print ((75 - 32) * 5 / 9)

fstrings:

Basic syntax: print (f" text {calculation} more text if you want ")

e.g: print (f" Sum of 35 and 2 is {35+2} only")

Number of Decimal Places In Float Calculations:

Basic syntax: print (f" text {28/7:.No of Dec f} more text ")

e.g: print (f" Div of 28 and 7 is {28/7:.1f} only.")

OUT: Div of 28 and 7 is 4.0 only.

Date:

Variables ✓ Variables with fstrings ✓

Building LLM prompts with variables

Now that we have seen how to combine vars & fstring to customize & quickly update strings, lets explore how we can extend this pattern to create prompts that let us interact with AI models from inside a python program.

Code:

```
from helper_functions import print_llm_response
print_llm_response("What is the capital of France?")
```

OUT: The capital of France is Paris.

We can thus customize this code and use it for anything using fstrings & variables.

e.g:

```
age = 20
print_llm_response(f"What is the age of {age} in dog years?")
```

OUT: 5

Date:

Functions Some functions from helper_functions import *

1. type(17) 2. print_llm_response("Capital of France")

OUT: INT OUT: Paris.

3. print(len("Hello world!") 4. print(round(42.1))

OUT: 12 OUT: 42

Now, you could use variables with these functions. Ex!

For prompts in Python

1) response = get_llm_response(prompt)
2) print(response)

OR

2) print_llm_response("prompt")

It is better to use method 1. because this will allow us to modify the response if needed for e.g in RAG-Chain.

where prompt = Question

e.g: prompt = "What is the capital of France"

CURD
TL;DR

Module 2- Automating Tasks With Python

Date:.....

List: (Array)

e.g:- Syntax: ListName = [Data element 1, Data element 2, ...]

Code: friends_list = ["Tommy", "Isa", "Dan"]
0 1 2

Type(friends_list)

OUT: List

Code:
prompt = f""" Write poems for my friends {friends_list}. Each Poem should be inspired by the letters of their first names. """

print_llm_response(prompt)

OUT: 3 different poems for: Tommy, Isa, Dan.

Code:
friends_list.append("otto")
Print(friends_list)
OUT: ['Tommy', 'Isa', 'Dan', 'Otto']

Code:
friends_list.remove("Tommy")
Print(friends_list)
OUT: ['Isa', 'Dan', 'Otto']

List of tasks: Date:

```
# Python allows for multi-line lists
list_of_tasks = [
    "Compose email to boss explaining I will be late for work",
    "Write a birthday poem for Otto",
    "Write a 300-word review for The Lion King Movie"
]

print_llm_respose(list_of_tasks[0])
```

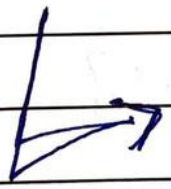
OUT: (writes an email)

and so on...

But this could be repetitive if number of elements were significantly large.

So for that we will use:

Loops :- FOR loop.



Repeating actions for multiple elements

FOR loops let you repeat a set of commands for each element in a list.

Code: A

```
for task in list_of_tasks:
<---- 4spaces ---->print_llm_response(task)
```

task: Name you define for the element at each iteration. (defined at runtime)

In plain English:

For each task in the list of tasks, print the llm response to ~~ea~~ the task.

Code A is equal to:

```
print_llm_response(list_of_tasks[0])
print_llm_response(list_of_tasks[1])
print_llm_response(list_of_tasks[2])
```

Adding LLM's response to list:

1) Create empty list:

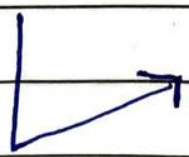
Code: promotional_descriptions = []

```
ice_cream_flavours = ["Vanila",
                      "Strawberry",
                      "Chocolate"]

for flavor in ice_cream_flavours:
    prompt = f"""Write
    icecream description of
    the following flavour:
    {flavor}
    """
    description = get_llm_response(prompt)
    #new var called description will hold response.
    # Append description in promotional_descriptions to create list.

    promotional_descriptions.append(description)
```

Lets now learn about dictionaries in Python.



WHY USE DICTIONARY?

Date:.....

- A long list with many data: items is fine, but a Problem arises in this case that you would not be able to remember what data was at what index number in the list.
- There is another way of organizing data in Python called a dictionary that makes it easier.

Dictionaries in Python

As a hard cover dictionary has words / concepts and definitions, a dictionary in Python has [keys] and [values]. "key: value" pairs.

Code:

Syntax:-
```
variable = {
        "key": "value",
        "key2": "value2",
    }
```

Example:
```
ice_cream_flavors = {
    "Mint-chocolate": "Refreshing brown",
    "Cookie": "Vanilla chunks"
}
```

functions of dicts

Date:.....

1) To Access keys only:

Code: print(ice_cream_flavors.keys())

OUT: dict_keys(['mint chocolate', 'Cookie'])

2) To access values only:

Code: print(ice_cream_flavors.values())

OUT: dict_values(['Refreshing brown', 'vanilla chunks'])

3) Accessing a specific value from key (given)

Code: cookie_desc = ice_cream_flavors["Cookie"]
print(cookie_desc)

OUT: Vanilla chunks :

4) Adding a new key-value pair

Code: ice_cream_flavors["RockyRoad"] = "Mixture"

5) Updation of existing key-value pair:

Code: ice_cream_flavors["Cookie"] = "Round & Crunch"

- In the same way, even an entire list can be added to dict:

Code: ice_cream_flavors["choco"] = ["Dark", "Light", "medium"]

Similarly, ["Dark", "Light", "Medium"] could be replaced with variable 'options' when options would be pre-defined as:

options = ["Dark, "Light", "medium"]

Recipe_Tracker/generator Project ✓

- Booleans are a data type that can only take two values, like the outcomes from a coin toss.

E.g: is_tommy_my_friend = True # tommy is indeed my good friend

AND operator: and

OR operator: or

Prioritizing Tasks through an LLM

```
task_list = [
    {
        "Desc": "Write email",
        "time" : 3
    },
    {
        "Desc": "Create outline",
        "time": 60
    }
]
```

```
task = task_list[0]
print(task)
OUT: {'desc': "Write email", 'time': 3}

task["time"] <= 5
OUT: True
# Now, making code to make a program to do a
# task that only takes 5 minutes or less
if task["time"] <= 5:
    task_to_do = task["description"]
    print_llm_response(task_to_do)
OUT: Does the task ✓
     OR
for task in task_list:
    if task["time_to_complete"] <= 5:
        task_to_do = task["desc"]
        print_llm_response(task_to_do)
    else:
        print(f"To complete later: {task['time']} time to complete")

OUT: Email
     To complete later: time: TO time to complete
✓ Any other task that takes less then or equal to 5.
```

Module 3: Working with your own data and documents

Date: 9th June 25

Using files in Python:

- Obsidian stores data in textfile / markdown files and you can use Python to read / extract data/text from these files and you can then use AI to process that extracted text.

Code: (usually what we do)

```
from helper_functions import get_llm_response
from IPython.display import display, Markdown
ingredients = [ List of ingredients ]
prompt = f"""
        Create recipe using {ingredients}.
"""
response = get_llm_response(prompt)
print(response).
```

OUT: Recipe ✓

It turns out that instead of typing all the data into the notebook like ingredients = [List of ingredients ..], you can load data from a text file or any other sort of file stored in your computer.

Date:

Lets start by loading an email stored in a text file.

Code:

```
f = open("email.txt", "r")   → in reading mode
email = f.read()
f.close()
#So by running this code, this just took all the
# text in the file email.txt, read it("r") and
# saved it in email variable.
print(email)
```

OUT: prints the whole email that was in the txt file ✓

Now, by using an LLM, we can process the text of the txt file.

Code:

```
prompt = f"""
        Extract bullet points & sender info from
        this email: {email}"""
print(get_llm_response(prompt))
bullet_points = get_llm_response(prompt)
```

OUT: bullet points (unformatted) ✓

Code: display(Markdown(bullet_points))

OUT: bullet points (properly formatted) ✓

- Directory = Folder
- Jupyter Notebooks are files with .ipynb extension
- **ipynb** means ipython Notebook.
- The folder in which Python looks for files is called the "current working directory."

Code:

```
from helper_functions import upload_txt_file,
                             list_files_in_directory,
                             print_llm_response
list_files_in_directory()
```

OUT:
```
Lesson2.ipynb
helper_functions.py
email.txt
recipe.txt
```

→ Basically all of the files that are present in the current working directory.

To upload a file

Code: `upload_txt_file()`

OUT: An [upload] button that allows you to upload files from your computer to the current working directory.

Date:

- Often times, it is required to process large amounts of data (text files or otherwise); we will learn how to do this using AI.
- We use Python & LLM to extract relevant information from these text files.
  For e.g, from a food critic journal entry, the useful info that we might want to extract out would be: The restaurant name, name of dish
- But before extracting, we might want to double check if this document is relevant.
  For eg if you have an article about the history of a city, that may not have current restaurant names that are worth trying to extract.
- Lets see how we can use Python to do this.

Date:

Code:

```
from helper_functions import * # get, print, list

f = open("capetown.txt", "r")
journal_cape_town = f.read()
f.close()
f = open("tokyo.txt", "r")
journal_tokyo = f.read()
f.close()
# If we output journal_cape_town & journal_tokyo,
# we will realize that the formats are different:
# journal_cape_town is in a paragraph format while
# journal_tokyo is in a bullet point format.
```

Now, before processing text files like these in order to extract restaurant names, and ~~dishes~~ specialities we'd like an LLM to check if doc is relevant:

Code:

```
prompt = f""" Respond with "Relevant" or "Not relevant":
        The journal describes restaurants & their specialities.
        Journal: {journal_tokyo} and Journal2: {journal_capetown}
"""

print_llm_response(prompt)
```

out: Relevant, Relevant.

Date:

In the following example, code we have a 5-file list and
we will be using a FOR loop to check relevancy
of each file.

Code:

```
files = ["cape_town.txt", "madrid.txt", "rio.txt", "sydney.txt", "a.txt"]

for file in files:
    f = open(file, "r")
    journal = f.read()
    f.close()

    prompt = f""" Respond with "Relevant" or "Not relevant":
        The journal describes restaurants and their
        specialties:
        journal: {journal}
    """
    print(f"{file} → {get_llm_response(prompt)}")
```

Date:

- Now we will move onto the process of how the LLM would extract particular bits of information from the file(s) and how we can save it to a new file for easy reading.

- For e.g, from a food critic journal entry, the useful information within the text might be:
  1) Restaurant name and 2) Name of dish.
  and we might want to save it like such:

Restaurant	Dish
Test Kitchen	Taco

Code:

```
from helper_functions import *
from IPython.display import display, HTML

journal_rio_de_janerio = read_journal("rio_de_janerio.txt")
# read_journal is a function that opens up this
# text file, it reads the text, and in this case,
# saves it to the variable: journal_rio_de_janerio.
```

3.43
3.2 - 3.54

Date:

- Similarly, you can write in the prompt that you want your response in an HTML format, and in order to apply that HTML format, you must use the HTML() function like so:

Code: display(HTML(html-response))
OUT: HTML-applied output. ✓

Tip: Change Prompt or Modify prompt to modify
HTML formatted output.

- Example of more prompts:
  1) CSV: (Tabular form) (Comma separated values)
  prompt = f""" Extract list of restaurants & their best dishes mentioned in the journal.
  Povide your answer in CSV format, ready to save..
  Exclude the ```csv declaration, dont add spaces after the comma, include column headers.
  Format:
  Restaurant, Dish
  Res_1, Dsh_1
  ...
  Journal entry: {journal_rio_de_janeiro}
  """

Date:----------

How to save html data that was extracted into a file:

Code:

```
f = open("highlighted_text.html", "w")
f.write(html_response)
f.close()
```

You can even download the file:

Code:

```
download_file()
# Opens up a bar like such:
```

File: [ ] [Download]

↳ Blinking cursor

Type in your filename in the bar and click Download

In this case, you will type: highlighted_text.html.

CSV Files

Date: 10 June 2025

- Text files can be formatted in very different ways therefore, we call text files "unstructured data."
- Structured data can be represented for example in a spreadsheet with rows and columns. And structured data like a spreadsheet is processed a bit differently in Python than unstructured data like text.
- You can often use Python code to process it directly, even without using an AI model.

For eg if you have a "vacation" spreadsheet with each column denoting a country, you could ~~filter~~ use Python to filter all of the stops made in a specific country.

Example:

Arrival	Departure	City	Country
July-01	July-08	NewYork	USA
July-09	July-16	Rio DeJaneiro	Brazil
⋮	⋮	⋮	⋮

- Each row corresponds to a destination w arrival/departure dates.

In a CSV file, ithis would look like:

Arrival, Departure, City, Country
July-01, July-08, NewYork, USA
July-09, July-16, Rio-, Brazil

Manipulating csv data with Python (much like SQL) Date:.....

Code:

```
from helper_functions import * #(getItinResp, display_table)
from IPython.display import Markdown
import csv

f = open("itinerary.csv", "r")
csv_reader = csv.DictReader(f)
itinerary = []
for row in csv_reader:
    print(row)
    itinerary.append(row) # And saving it in the list itinerary.
# This is loading itinerary.csv one row at a time.
```

OUT: {'Arrival': 'July-01', 'Departure': 'July-08', 'City': 'New York', 'County': 'USA'}
{'Arrival': 'July-09', 'Departure': 'July-16', 'City': 'Rio--', 'County': 'Brazil'}

Code: type(itinerary)
OUT: List

Code: print(itinerary[0])
OUT: {'Arrival': 'July-01',
'Departure': 'July-08',
'City': 'New York',
'County': 'USA'}

Code: print(itinerary[0]["County"])
OUT: USA.

Code

Date:

About csv.DictReader( ):

- A function used to read csv files and convert each row into a dictionary (key:value pair) where keys are the column headers and the values are the data from each row.
- Its parameters can be: 1) the file object,
  2) fieldnames: A list of field names to use as dict keys.
  3) restkey. 4) restval: A default value to use for any missing fields in a row, etc.
  5) dialect, **kwargs.

Continued from the previous code...

Code: display_table(itinerary)

out: prints all data in a suitable tabular format.

Filtering By Country (eg Japan)

Code: filtered_data = []

```
for trip_stop in itinerary:
    if trip_stop["Country"] == "Japan":
        filtered_data.append(trip_stop)
display_table(filtered_data)
```

out:

Arrival	Departure	City	Country
Aug-10	Aug-17	Tokyo	Japan

Code:

Date:

```
# Finally, using an LLM to plan your
# entire vacation:

city = trip_stop["City"]
country = trip_stop["country"]
arrival = trip_stop["Arrival"]
departure = trip_stop["Departure"]

prompt = f"""
    I will visit {city}, {country}, from
    {arrival} to {departure}. Please create a
    detailed daily itinerary"""

response = get_llm_response(prompt)
display(Markdown(response))
```

OUT: OUTPUTS THE DETAILED ITINERARY

✓

Date: 11 June 2025

Making Your Own Functions In Python

[Turning code blocks into reusable functions]

Example :

Code :

```
def print_journal(file):
    f = open(file, "r")
    journal = f.read()
    f.close
    Print(journal) or return journal
    journal_sydney = print_journal("sydney.txt")
print_journal("sydney.txt") or print(journal_sydney).
```

out : outputs the entire text of sydney.txt.

```
# def : Definition statement
# print_journal : function name
# file : Parameter.
# return : Returning.
```

- Functions let you avoid writing blocks of commands repeatedly. It also makes your program easier to read and understand.

Date:

A detailed itinerary for multiple cities

Code:

```
detailed_itinerary = [ ]

for trip_stop in itinerary:
    city = trip_stop["city"]
    country = trip_stop["country"]
    arrival = trip_stop["Arrival"]
    departure = trip_stop["Departure"]
    rest_dict = read_csv(f"{city}.csv")
```

```
def read_csv(file):
    f = open(file, "r")
    csv_reader = csv.DictReader(f)
    data = [ ]
    for row in csv_reader:
        data.append(row)
    f.close
    return data
itinerary = read_csv("itinerary.csv")
```

continued...

Date:

Code: detailed_itinerary = [ ]

```
for trip_stop in itinerary:
    city = trip_stop["City"]
    country = trip_stop["Country"]
    arrival = trip_stop["Arrival"]
    departure = trip_stop["Departure"]
    rest_dict = read_csv(f"{city}.csv")

    print(f"Creating detailed itinerary
          for {city}, {country}.")

    prompt = f"""I will visit {city}, {country} from
    {arrival} to {departure}. Create a ~~det~~ daily itinerary
    with detailed activities. Designate times for
    breakfast, lunch & dinner.
    I want to visit the restaurants listed in the
    restaurant dictionary; ~~dgga~~ without repeating any
    place. Make sure to mention the specialty
    that I should try at each of them.
    Restaurant dictionary: {rest_dict}
    """
```

Date:_____

Code: (continued)

detailed_itinerary[city] = get_llm_response(prompt)

OUT: - Creating detailed itinerary for {city}, country

" " " " " " "

Code: display(Markdown(detailed_itinerary["Tokyo"]))

OUT: Tokyo vacation [entire] gets printed!

Module 4

Date: 18 June 2025

In this final module we will learn how to extend Python with code that someone else has written and made available on the internet.

Python has tools available that let you download and install code written by many others, and this third party code is usually bundled together into something called a Package. So you will learn how to install Python packages which can immediately give you additional Power functions to call.
For e.g, if you want your code to download a webpage & extract the text so they can use AI to summarize it for you, you will be able to find a free package that lets your code download web pages and extract text.
We will see a lot of things that packages will be able to do for us later in this module.

Date:

- One important type of Package goes further and Provides application Programming interfaces, or APIs, that lets your code call on someone else's code thats running on someone else's computer on the internet.

For e.g, there are APIs that let your computer go online and ask some other computer, say run by a weather service, to return the current weather at this moment in time at your location.

There are also APIs that let your code go online to get the current news or get stock prices or even search the web.

In addition to using APIs to gather info, your code can ~~eve~~ also use APIs to even take action in the world, such as maybe have your code use an API to send a text or email message that will then land in someone else's phone, or even use an API to place an online shopping order that results in a physical product being shipped to someone's house.

Date:

There are two ways to get function into your code:

1. Use the def keyword and define your own function
2. Use the import command

You can in this way, choose to import specific tools (pieces of code). In this lesson, we will see how to import code that can help you with math and stats; and we will also introduce desirable randomness into your code. The concepts & code examples in this lesson will create a foundation for the next one to come and which will then look at 3rd party packages that you can download and install from the internet.

Example math:

Code:

```
from math import cos, sin, pi
print(pi)
type(pi)
```

OUT: 3.141592653589793
float

Code:

```
values = [0, pi/2, pi, 3/2 * pi, 2*pi]
for value in values:
    print(f"The cosine of {value:.2f} is {cos(value):.2f}")
```

OUT: The cos

Date:

OUT: The cosine of 0.00 is 1.00
The cosine of 1.57 is 0.00
" " " 3.14 is -1
.. and so on.

Function for rounding-down numbers:
You can use the floor function.
Code: floor(5.7) # floor function is part of math library.
OUT: 5 # only when from math import floor is there.

Stats:
Code: from statistics import mean, stdev
my_friends_heights = [160, 172]
mean(my_friends_heights)
OUT: 166
Code: stdev(my_friends_heights)
OUT: 12

Randomness in Python:

Code:

```
from random import sample
mylist = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sample_size = 3
random_samples = sample(mylist, sample_size)
print(random_samples)
```

OUT: Three random numbers from the list mylist.

Randomness in a recipe with the LLM

```
from random import sample
spices = [ list of spices ]
vegetables = [ list of vegetables ]
proteins = [ list of proteins ]

random_spices = sample (spices, 2)
random_vegetables = sample(vegetable, 2)
random_proteins = sample(proteins, 1)
prompt = f""" suggest a recipe with the
        following ingredients:
        Spices : { random_spices }
        Vegetables : { random_vegetables }
        Proteins : { random_proteins } """
```

Continued...

Date:

```
from helper_functions import get_llm_response

recipe = get_llm_response(prompt)
print(recipe)
```

OUT: Prints relevant recipe! ✓

Using Third-Party Packages

Amongst the hundreds of thousands of third-party packages available, the packages that we will see are called pandas and matplotlib.

Pandas: Used for processing structured data like kind of stored in spreadsheets and CSV files.

Code: `import pandas as pd`

Meaning: This means that from now on, any function from the Pandas library that you use, you can use the syntax pd.function() instead of pandas.function()

In this lesson, we will use a data set on used car sales prices.

Code:

```
data = pd.read_csv("car_data.csv")
print(data)
```

OUT: outputs a properly formatted csv file with car details: model, price, year, kilometer.

Pandas also lets you filter information, for e.g., you can get all the costs for which the selling "prices" is above $10,000.

1) To do that you write this code:

Code: `print(data[data["Price"] >= 10000])`

OUT: Prints the substantive cost of the Price over $10,000.

2)

Code: `print(data[data["Year"] == 2015])`

OUT: Prints all cars with modelyear of 2015.

The use of filtering was done & learned through the AI chatbot as it was mentioned that there may be a lot of functions to remember, so it is best to just use AI chatbot.

Date:

Another thing that you could do:

```
filtered_data = data[data["Year"] == 2015]
#filtered_data now stores all cars of 2015.
```

You don't need to remember any specific line of code because if ever you want to filter the data or compute the mean or compute the median, it is encouraged to ask the AI Chatbot for help.

Code: `print(filtered_data["Price"].median())`

out: 5475.0

Pandas, by the way, is a very powerful tool that Professional Data Scientists use day-to-day in their work.

Beyond, manipulating data, there is another very popular package for plotting data called matplotlib.

matplotlib

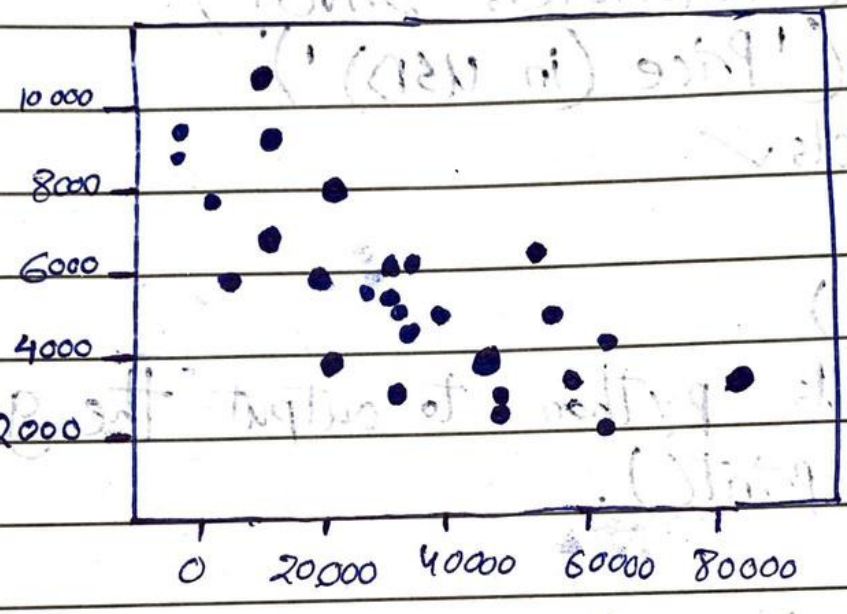
To import, use:

Code: `import matplotlib.pyplot as plt`

To plot a graph, use:

Code: `plt.scatter(data["Kilometer"], data["Price"])`

OUT:



where the x-axis is the mileage in kms, and the y-axis is the price in US dollars.

From the previous graph, if you want to label the axes, you can add a few extra commands like this:

Code:

```
plt.title('Car Price vs. Kilometers Driven')
plt.xlabel('Kilometers Driven')
plt.ylabel('Price (in USD)')
```

OUT: Adds labels ✓

Code: `plt.show()`

Explicitly tells python to output the graph. just like print().

To add a grid, just use:

```
plt.grid(True)
```

Date:

Installing Packages

Some packages are pre-installed, while you might have to install some of the packages on your own computer.

In this lesson, you will see what the process of downloading and installing packages is like. You go through the installation process for a package for interpreting HTML web pages.

If your code downloads a web page off the internet, you see that the raw web page might have a lot of HTML tags that indicate where paragraphs start and end, what font size to use, and so on. You learn how to download and install a package called bs4 for extracting the text you want from this mass of HTML. And in addition, you also learn how to install a package called 'ai-setup', which is provided by deeplearning.ai and contains all the key helper functions that you have used throughout this course. So if you want, you can install ai-setup and then run functions yourself that we have used, in a different Jupyter notebook

Date:

There are many ways to install Python packages.
Here, we would learn how to install directly in your Jupyter Notebook:

bs4: for interpreting web pages.

Code/Command: !pip install bs4
#! is known as bang
#syntax: !pip install package_name
OUT: Successfully installed bs4-0.0.2
#version 0.0.2.
#bs4: beautiful soup version 4.

After installing the package, you can import it or any function it contains:

Code: from bs4 import BeautifulSoup
import requests
from helper_functions import *
from IPython.display import HTML, display

Date:

To scrape data from the web, first you need to get a web page's contents:

Code: url = "https://www.deeplearning.ai/the-batch/the-world-needs-more-intelligence"

Now, to fetch this url, we will use requests:

Code: response = requests.get(url)
print(response)
OUT: <Response [200]>

To display the webpage,
Code: HTML(f"iframe src={url} width='60%'
height='400'></iframe>")
OUT: THE ENTIRE WEB PAGE GETS PRINTED INSIDE THE JUPYTER NOTEBOOK

Now that we have an entire webpage to us, with all of its tabs, links, pictures, we might want to extract just the main text of this with our large language model. And for that, we will use BeautifulSoup.

#Using BeautifulSoup to extract the text:

Code:

```
Line1  soup = BeautifulSoup(response.text, "html.parser")
Line2  all_text = soup.find_all("p")

Line3  combined_text = ""
Line4  for text in all_text:
Line5      combined_text = combined_text + "\n" + text.get_text()
Line6  print(combined_text)
```

OUT: outputs all paragraph <p> text of the HTML file ✓

Extracted text, 'combined_text' is just a string that you can then insert into prompt for a large language model:

Code:

```
prompt = f""" Extract key bullet points from the text: {combined_text}"""
print_llm_response(prompt)
```

OUT: outputs key bullet points ✓

Installing aisetup Package

Code/command: !pip install aisetup

OUT: installed successfully.

aisetup Package includes all the helper functions we've seen so far. (works on website ✓)

Code: from aisetup import get_llm_response

To actually use AI setup on your own computer has one more step that we will talk about ahead. since it turns out that get_llm_response is using something called an API (Application Programming Interface to go on the internet to use one of the large language model provider's website (OpenAI in this case), to get a response APIs are a very powerful tool to use someoneelse's computer to get answers. (with permission, of course!)

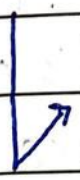
Date:

APIs to get data from the web

In order to access data such as the current weather, or stock prices, or news, or calling a web server to search for relevant web pages, You need to use an API. It gives a way for your computer to talk to another computer & return relevant data to your computer.
There are lots of APIs on the internet.
Some are free, while many require a fee.
But in contrast to downloading a package to run on your computer, APIs let you get someone else's computer, with permission, to do work, to do something for you. Lets see how APIs work:

Example: Getting real-time weather data using an API to plan an outfit for the day.

Code:
```
import os
import requests
from aisetup import print_llm_response
from dotenv import load_dotenv
```



Date:

The API we will use is provided by: openweathermap.org

- most APIs will ask for a key.

Q) What is a key (API key)?
A) A password that is unique to you.

Q) Why is this needed
A) This way, the computer that is delivering on your request, knows who the request is coming from.

Code:
```
# Get the weather API key from the .env file
load_dotenv('.env', override=True)
api_key = os.getenv('WEATHER_API_KEY')

lat = 37.14    # Input city lat and lon coordinates.
lon = -122.1430
url = f"https://api.openweathermap.____")
response = requests.get(url)
data = response.json()
print(data)
```

Date:----------

Code: (continued)

```
temperature = data['list'][0]['main']['temp']
desc = data['list'][0]['weather'][0]['description']
wind_speed = data['list'][0]['wind']['speed']

print(f"Temperature: {temperature}")
print(f"Weather Description: {desc}")
print(f"Wind Speed: {wind_speed}")
```

OUT: Temperature: 24.02
Weather Description: clear sky
Wind Speed: 4.29

```
# Now, asking AI chatbot for clothes.
prompt = f""" Based on the following weather, suggest an
        appropriate outdoor outfit:
        Forecast: Temperature: {temperature}
        Description: {desc} and Wind Speed: {wind_speed}"""
print_llm_response(prompt)
```

OUT: Wear a lightweight shirt and shorts. Dont forget sunglasses.

Date:----------

To actually do this on your own computer, outside of the current Jupyter Notebook environment, you would need to register an account yourself at openweathermap.org and fetch a unique, secret API key.

One way to do that would be to have a line of code that says API key equals to whatever string of characters and numbers.

api_key = "123myAPI" # or whatever.

But, even though this works, most programmers do not do this because if your code ever gets leaked, then others will have access to your secret API key. So what most programmers will do is use the .env packages,

"from dotenv import load_dotenv"

and use the load_dotenv function which goes through a couple of extra steps to securely load and use the API key.

Date:

In this last lesson we will be looking at how to use APIs to use AI models.

To use OpenAI's API, first we run:

```
!pip install openai
```

Then we write the following code:

Code:

```
import os
from dotenv import load_dotenv
from openai import OpenAI
# So the function, get_llm_response() is stored in
# this library.
```

get_llm_response

How this function was created.

Code:

```
def get_llm_response(prompt):
    completion = client.chat.completions.create(
    model= "gpt-4o-mini",
    messages = [
                {
                "role" : "system",
                "content": "Youare an AI assistent-",
                }
                {"role": "user", "content": prompt},
    ],
    temperature = 0.0
    )
```

Date:.....

continued code...

```
    response = completion.choices[0].message.content
    return response
```

EXPLANATION OF CODE:

Code: `completion = client.chat.completions.create(...)`

↓

This creates the request, sends it, and handles the response from the OpenAI API.

client.chat.completions.create() is a function provided by OpenAI.

Code: `model = "gpt-4o-mini"`

This line selects the large language model that we want to use, and when we use an LLM, one of the things we often do is tell the LLM how to respond, that is called the system message and we tell the LLM that we want it to act like an AI assistant:

Code:

```
messages = [
    {
        "role": "system",
        "content": "You are an AI assistant.",
    }
```

Date:

And then we also specify the prompt, which can be a question like, "what is the capital of France?"

Code:
```
{ "role": "user", "content": prompt},
] # The prompt to pass to the LLM.
```

- LLMs have a parameter called the temperature parameter, which controls how random the response is. and we set it to 0.0 if we don't want it to be too random.

Code: `temperature = 0.0 #lowest possible temperature`

You donot need to worry so much abaut what every line of this code does. In fact, you can find such snippets of codes in OpenAI's API documentation, or Anthropic's Cloud documentation, or Google Gemini's documentation, or whatever you are using, and then get it to work on yar own code.

Note: AI language models have learned by reading text off the internet, and so they will be better at understanding better-known APIs / older APIs that have been around on the internet for a while.

Date:

They may not be as good at understanding less popular APIs or APIs that have been created and released by someone else on the internet.

Code:

```
import os
from dotenv import load_dotenv
from openai import OpenAI

# Get the OpenAI API key from the .env file
load_dotenv('.env', override=True)
openai_api_key = os.getenv('OPENAI_API_KEY')
client = OpenAI(api_key=openai_api_key)

def get_llm_response(prompt):
    ---  # Done before
    ---

return response.

prompt = "What is the capital of France?"
response = get_llm_response(prompt)
print(response)
```

OUT: Paris.

Date:

Note:

- Vary temperature b/w 0-2 to achieve randomness. 0.0 will keep giving you the same answer. The average temperature value used in the industry is 0.7.

- Modify the value of the key "content" to do fun things with the chatbot.

How to get API key into the code:

1) !pip install aisetup

2) from aisetup import authenticate, print_llm_response, get_llm_response.

authenticate("Your API key here")

print_llm_response("what is the capital of France?")

Date:----------

However, having your API key in the code is not safe so a better way to do it would be: from dotenv import load_dotenv

Code: import os

```
load_dotenv('.env', override=True)
openai_api_key = os.getenv('OPENAI_API_KEY')
authenticate(openai_api_key)

print_llm_response("Whats the capital of France?")
```

OUT: Paris