A Collection of 25 Awesome Python Scripts (mini projects)

In this lesson, I've compiled a collection of 25 Python programmes. I've included links to learn more about each script, such as `packages installation` and `how to execute script?`.

Follow me on twitter - @harendraverma2

Follow me on medium.com - @harendraverma21

Buy Me A Coffee

1. Convert JSON to CSV

```
import json
if __name__ == '__main__':
    try:
        with open('input.json', 'r') as f:
            data = json.loads(f.read())
        output = ','.join([*data[0]])
        for obj in data:
            output += f'\n{obj["Name"]},{obj["age"]},{obj["birthyear"]}'

        with open('output.csv', 'w') as f:
            f.write(output)
    except Exception as ex:
        print(f'Error: {str(ex)}')
```

2. Password Generator

```
import random
import string
total = string.ascii_letters + string.digits + string.punctuation
length = 16
password = "".join(random.sample(total, length))
print(password)
```

3. String search from multiple files

```
import os
text = input("input text : ")
path = input("path : ")
# os.chdir(path)
def getfiles(path):
    f = 0
    os.chdir(path)
    files = os.listdir()
    # print(files)
    for file_name in files:
        abs_path = os.path.abspath(file_name)
        if os.path.isdir(abs_path):
            getfiles(abs_path)
        if os.path.isfile(abs_path):
            f = open(file_name, "r")
            if text in f.read():
                f = 1
                print(text + " found in ")
                final_path = os.path.abspath(file_name)
                print(final_path)
                return True
    if f == 1:
        print(text + " not found! ")
        return False

getfiles(path)
```

4. Fetch all links from a given webpage

```python
import requests as rq
from bs4 import BeautifulSoup

url = input("Enter Link: ")
if ("https" or "http") in url:
    data = rq.get(url)
else:
    data = rq.get("https://" + url)
soup = BeautifulSoup(data.text, "html.parser")
links = []
for link in soup.find_all("a"):
    links.append(link.get("href"))

# Writing the output to a file (myLinks.txt) instead of to stdout
# You can change 'a' to 'w' to overwrite the file each time
with open("myLinks.txt", 'a') as saved:
    print(links[:10], file=saved)
```

5. Image Watermarking

```python
import os
from PIL import Image

def
watermark_photo(input_image_path,watermark_image_path,output_image_path):
    base_image = Image.open(input_image_path)
    watermark = Image.open(watermark_image_path).convert("RGBA")
    # add watermark to your image
    position = base_image.size
    newsize = (int(position[0]*8/100),int(position[0]*8/100))
    # print(position)
    watermark = watermark.resize(newsize)
    # print(newsize)
    # return watermark

    new_position = position[0]-newsize[0]-20,position[1]-newsize[1]-20
    # create a new transparent image
    transparent = Image.new(mode='RGBA',size=position,color=(0,0,0,0))
    # paste the original image
    transparent.paste(base_image,(0,0))
    # paste the watermark image
    transparent.paste(watermark,new_position,watermark)
    image_mode = base_image.mode
    print(image_mode)
    if image_mode == 'RGB':
        transparent = transparent.convert(image_mode)
    else:
        transparent = transparent.convert('P')
    transparent.save(output_image_path,optimize=True,quality=100)
    print("Saving"+output_image_path+"...")

folder = input("Enter Folder Path:")
watermark = input("Enter Watermark Path:")
os.chdir(folder)
files = os.listdir(os.getcwd())
print(files)

if not os.path.isdir("output"):
    os.mkdir("output")

c = 1
for f in files:
    if os.path.isfile(os.path.abspath(f)):
        if f.endswith(".png") or f.endswith(".jpg"):
            watermark_photo(f,watermark,"output/"+f)
```

---

6. Scrap and Download all images from the WEB Page

```
from selenium import webdriver
import requests as rq
import os
from bs4 import BeautifulSoup
import time

# path= E:\web scraping\chromedriver_win32\chromedriver.exe
path = input("Enter Path : ")

url = input("Enter URL : ")

output = "output"


def get_url(path, url):
    driver = webdriver.Chrome(executable_path=r"{}".format(path))
    driver.get(url)
    print("loading.....")
    res = driver.execute_script("return
document.documentElement.outerHTML")

    return res


def get_img_links(res):
    soup = BeautifulSoup(res, "lxml")
    imglinks = soup.find_all("img", src=True)
    return imglinks


def download_img(img_link, index):
    try:
        extensions = [".jpeg", ".jpg", ".png", ".gif"]
        extension = ".jpg"
        for exe in extensions:
            if img_link.find(exe) > 0:
                extension = exe
                break

        img_data = rq.get(img_link).content
        with open(output + "\\" + str(index + 1) + extension, "wb+") as f:
            f.write(img_data)

        f.close()
    except Exception:
        pass


result = get_url(path, url)
time.sleep(60)
img_links = get_img_links(result)
if not os.path.isdir(output):
    os.mkdir(output)

for index, img_link in enumerate(img_links):
    img_link = img_link["src"]
    print("Downloading...")
    if img_link:
        download_img(img_link, index)
print("Download Complete!!")
```

---

7. Low Battery Notification

```
# pip install psutil
import psutil

battery = psutil.sensors_battery()
plugged = battery.power_plugged
percent = battery.percent

if percent <= 30 and plugged!=True:

    # pip install py-notifier
    # pip install win10toast
    from pynotifier import Notification

    Notification(
        title="Battery Low",
        description=str(percent) + "% Battery remain!!",
        duration=5,  # Duration in seconds

    ).send()
```

---

8. Calculate Your Age

```
import time
from calendar import isleap

# judge the leap year
def judge_leap_year(year):
    if isleap(year):
        return True
    else:
        return False


# returns the number of days in each month
def month_days(month, leap_year):
    if month in [1, 3, 5, 7, 8, 10, 12]:
        return 31
    elif month in [4, 6, 9, 11]:
        return 30
    elif month == 2 and leap_year:
        return 29
    elif month == 2 and (not leap_year):
        return 28


name = input("input your name: ")
age = input("input your age: ")
localtime = time.localtime(time.time())

year = int(age)
month = year * 12 + localtime.tm_mon
day = 0

begin_year = int(localtime.tm_year) - year
end_year = begin_year + year

# calculate the days
for y in range(begin_year, end_year):
    if (judge_leap_year(y)):
        day = day + 366
    else:
        day = day + 365

leap_year = judge_leap_year(localtime.tm_year)
```

```
for m in range(1, localtime.tm_mon):
    day = day + month_days(m, leap_year)

day = day + localtime.tm_mday
print("%s's age is %d years or " % (name, year), end="")
print("%d months or %d days" % (month, day))
```

---

9. Organized download folder with different categories

```
import os
import shutil
os.chdir("E:\downloads")
#print(os.getcwd())

#check number of files in  directory
files = os.listdir()

#list of extension (You can add more if you want)
extentions = {
    "images": [".jpg", ".png", ".jpeg", ".gif"],
    "videos": [".mp4", ".mkv"],
    "musics": [".mp3", ".wav"],
    "zip": [".zip", ".tgz", ".rar", ".tar"],
    "documents": [".pdf", ".docx", ".csv", ".xlsx", ".pptx", ".doc",
".ppt", ".xls"],
    "setup": [".msi", ".exe"],
    "programs": [".py", ".c", ".cpp", ".php", ".C", ".CPP"],
    "design": [".xd", ".psd"]


}


#sort to specific folder depend on extenstions
def sorting(file):
    keys = list(extentions.keys())
    for key in keys:
        for ext in extentions[key]:
            # print(ext)
            if file.endswith(ext):
                return key


#iterat through each file
for file in files:
    dist = sorting(file)
    if dist:
        try:
            shutil.move(file, "../download-sorting/" + dist)
        except:
```

```python
                print(file + " is already exist")
        else:
            try:
                shutil.move(file, "../download-sorting/others")
            except:
                print(file + " is already exist")
```

---

10. Send Emails in Bulk From CSV File

```python
import csv
from email.message import EmailMessage
import smtplib


def get_credentials(filepath):
    with open("credentials.txt", "r") as f:
        email_address = f.readline()
        email_pass = f.readline()
    return (email_address, email_pass)


def login(email_address, email_pass, s):
    s.ehlo()
    # start TLS for security
    s.starttls()
    s.ehlo()
    # Authentication
    s.login(email_address, email_pass)
    print("login")


def send_mail():
    s = smtplib.SMTP("smtp.gmail.com", 587)
    email_address, email_pass = get_credentials("./credentials.txt")
    login(email_address, email_pass, s)

    # message to be sent
    subject = "Welcome to Python"
    body = """Python is an interpreted, high-level,
    general-purpose programming language.\n
    Created by Guido van Rossum and first released in 1991,
    Python's design philosophy emphasizes code readability\n
    with its notable use of significant whitespace"""

    message = EmailMessage()
    message.set_content(body)
    message['Subject'] = subject

    with open("emails.csv", newline="") as csvfile:
        spamreader = csv.reader(csvfile, delimiter=" ", quotechar="|")
        for email in spamreader:
            s.send_message(email_address, email[0], message)
            print("Send To " + email[0])

    # terminating the session
    s.quit()
    print("sent")


if __name__ == "__main__":
    send_mail()
```

11. Get the IP Address and Hostname of A Website

```
# Get Ipaddress and Hostname of Website
# importing socket library
import socket

def get_hostname_IP():
    hostname = input("Please enter website address(URL):")
    try:
        print (f'Hostname: {hostname}')
        print (f'IP: {socket.gethostbyname(hostname)}')
    except socket.gaierror as error:
        print (f'Invalid Hostname, error raised is {error}')

get_hostname_IP()
```

12. Terminal Progress Bar

```
from tqdm import tqdm
from PIL import Image
import os
from time import sleep


def Resize_image(size, image):
    if os.path.isfile(image):
        try:
            im = Image.open(image)
            im.thumbnail(size, Image.ANTIALIAS)
            im.save("resize/" + str(image) + ".jpg")
        except Exception as ex:
            print(f"Error: {str(ex)} to {image}")


path = input("Enter Path to images : ")
```

```
size = input("Size Height , Width : ")
size = tuple(map(int, size.split(",")))

os.chdir(path)

list_images = os.listdir(path)
if "resize" not in list_images:
    os.mkdir("resize")

for image in tqdm(list_images, desc="Resizing Images"):
    Resize_image(size, image)
    sleep(0.1)
print("Resizing Completed!")
```

---

13. Wifi Password Ejector

```
import subprocess

data = (
    subprocess.check_output(["netsh", "wlan", "show", "profiles"])
    .decode("utf-8")
    .split("\n")
)
profiles = [i.split(":")[1][1:-1] for i in data if "All User Profile" in i]
for i in profiles:
    results = (
        subprocess
        .check_output(["netsh", "wlan", "show", "profile", i, "key=clear"])
        .decode("utf-8")
        .split("\n")
    )
    results = [b.split(":")[1][1:-1] for b in results if "Key Content" in
b]
    try:
        print("{:<30}|  {:<}".format(i, results[0]))
    except IndexError:
        print("{:<30}|  {:<}".format(i, ""))
```

---

14. Snapshot of The Given Website

```
import sys
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
import chromedriver_binary


script_name = sys.argv[0]
```

```
options = Options()
options.add_argument('--headless')
driver = webdriver.Chrome(options=options)

try:
    url = sys.argv[1]

    driver.get(url)
    page_width = driver.execute_script('return document.body.scrollWidth')
    page_height = driver.execute_script('return 
document.body.scrollHeight')
    driver.set_window_size(page_width, page_height)
    driver.save_screenshot('screenshot.png')
    driver.quit()
    print("SUCCESS")

except IndexError:
    print('Usage: %s URL' % script_name)
```

---

15. Split Files Into Chunks

```
import sys
import os
import shutil
import pandas as pd

class Split_Files:
    '''
        Class file for split file program
    '''
    def __init__(self, filename, split_number):
        '''
            Getting the file name and the split index
            Initializing the output directory, if present then truncate it.
            Getting the file extension
        '''
        self.file_name = filename
        self.directory = "file_split"
        self.split = int(split_number)
        if os.path.exists(self.directory):
            shutil.rmtree(self.directory)
        os.mkdir(self.directory)
        if self.file_name.endswith('.txt'):
            self.file_extension = '.txt'
        else:
            self.file_extension = '.csv'
        self.file_number = 1

    def split_data(self):
        '''
            spliting the input csv/txt file according to the index provided
        '''
        data = pd.read_csv(self.file_name, header=None)
        data.index += 1

        split_frame = pd.DataFrame()
        output_file = f"{self.directory}/split_file{self.file_number}
{self.file_extension}"

        for i in range(1, len(data)+1):
            split_frame = split_frame.append(data.iloc[i-1])
            if i % self.split == 0:
                output_file = f"
{self.directory}/split_file{self.file_number}{self.file_extension}"
                if self.file_extension == '.txt':
                    split_frame.to_csv(output_file, header=False,
index=False, sep=' ')
                else:
                    split_frame.to_csv(output_file, header=False,
index=False)
                split_frame.drop(split_frame.index, inplace=True)
                self.file_number += 1
        if not split_frame.empty:
            output_file = f"{self.directory}/split_file{self.file_number}
{self.file_extension}"
            split_frame.to_csv(output_file, header=False, index=False)

if __name__ == '__main__':
    file, split_number = sys.argv[1], sys.argv[2]
    sp = Split_Files(file, split_number)
    sp.split_data()
```

16. Encrypt and Decrypt Texts

```
from Crypto.Cipher import AES
from Crypto import Random
from binascii import b2a_hex
import sys

# get the plaintext
plain_text = sys.argv[1]

# The key length must be 16 (AES-128), 24 (AES-192), or 32 (AES-256) Bytes.
key = b'this is a 16 key'

# Generate a non-repeatable key vector with a length
# equal to the size of the AES block
iv = Random.new().read(AES.block_size)
```

```
# Use key and iv to initialize AES object, use MODE_CFB mode
mycipher = AES.new(key, AES.MODE_CFB, iv)

# Add iv (key vector) to the beginning of the encrypted ciphertext
# and transmit it together
ciphertext = iv + mycipher.encrypt(plain_text.encode())


# To decrypt, use key and iv to generate a new AES object
mydecrypt = AES.new(key, AES.MODE_CFB, ciphertext[:16])

# Use the newly generated AES object to decrypt the encrypted ciphertext
decrypttext = mydecrypt.decrypt(ciphertext[16:])

# output
file_out = open("encrypted.bin", "wb")
file_out.write(ciphertext[16:])
file_out.close()

print("The key k is: ", key)
print("iv is: ", b2a_hex(ciphertext)[:16])
print("The encrypted data is: ", b2a_hex(ciphertext)[16:])
print("The decrypted data is: ", decrypttext.decode())
```

17. Capture Screenshots At Regular Intervals of Time

```
import os
import argparse
import pyautogui
import time

parser = argparse.ArgumentParser()

parser.add_argument("-p", "--path", help="absolute path to store
screenshot.", default=r"./images")
parser.add_argument("-t", "--type", help="h (in hour) or m (in minutes) or
s (in seconds)", default='h')
parser.add_argument("-f", "--frequency", help="frequency for taking
screenshot per h/m/s.", default=1, type=int)

args = parser.parse_args()


sec = 0.

if args.type == 'h':
    sec = 60 * 60 / args.frequency
elif args.type == 'm':
    sec = 60 / args.frequency
```

```
if sec < 1.:
    sec = 1.


if os.path.isdir(args.path) != True:
    os.mkdir(args.path)


try:
    while True:
        t = time.localtime()
        current_time = time.strftime("%H_%M_%S", t)
        file = current_time + ".jpg"
        image = pyautogui.screenshot(os.path.join(args.path,file))
        print(f"{file} saved successfully.\n")
        time.sleep(sec)

except KeyboardInterrupt:
    print("End of script by user interrupt")
```

---

18. Decimal to Binary converter

```
try:
    menu = int(input("Choose an option: \n 1. Decimal to binary \n 2.
Binary to decimal\n Option: "))
    if menu < 1 or menu > 2:
        raise ValueError
    if menu == 1:
        dec = int(input("Input your decimal number:\nDecimal: "))
        print("Binary: {}".format(bin(dec)[2:]))
    elif menu == 2:
        binary = input("Input your binary number:\n Binary: ")
        print("Decimal: {}".format(int(binary, 2)))
except ValueError:
    print ("please choose a valid option")
```

---

19. CLI Todo App

```
import click

@click.group()
@click.pass_context
def todo(ctx):
    '''Simple CLI Todo App'''
    ctx.ensure_object(dict)
    #Open todo.txt – first line contains latest ID, rest contain tasks and
IDs
```

```
    with open('./todo.txt') as f:
        content = f.readlines()
    #Transfer data from todo.txt to the context
    ctx.obj['LATEST'] = int(content[:1][0])
    ctx.obj['TASKS'] = {en.split('```')[0]:en.split('```')[1][:-1] for en
in content[1:]}

@todo.command()
@click.pass_context
def tasks(ctx):
    '''Display tasks'''
    if ctx.obj['TASKS']:
        click.echo('YOUR TASKS\n**********')
        #Iterate through all the tasks stored in the context
        for i, task in ctx.obj['TASKS'].items():
            click.echo('• ' + task + ' (ID: ' + i + ')')
        click.echo('')
    else:
        click.echo('No tasks yet! Use ADD to add one.\n')

@todo.command()
@click.pass_context
@click.option('-add', '--add_task', prompt='Enter task to add')
def add(ctx, add_task):
    '''Add a task'''
    if add_task:
        #Add task to list in context
        ctx.obj['TASKS'][ctx.obj['LATEST']] = add_task
        click.echo('Added task "' + add_task + '" with ID ' +
str(ctx.obj['LATEST']))
        #Open todo.txt and write current index and tasks with IDs
(separated by " ``` ")
        curr_ind = [str(ctx.obj['LATEST'] + 1)]
        tasks = [str(i) + '```' + t for (i, t) in ctx.obj['TASKS'].items()]
        with open('./todo.txt', 'w') as f:
            f.writelines(['%s\n' % en for en in curr_ind + tasks])

@todo.command()
@click.pass_context
@click.option('-fin', '--fin_taskid', prompt='Enter ID of task to finish',
type=int)
def done(ctx, fin_taskid):
    '''Delete a task by ID'''
    #Find task with associated ID
    if str(fin_taskid) in ctx.obj['TASKS'].keys():
        task = ctx.obj['TASKS'][str(fin_taskid)]
        #Delete task from task list in context
        del ctx.obj['TASKS'][str(fin_taskid)]
        click.echo('Finished and removed task "' + task + '" with id ' +
str(fin_taskid))
        #Open todo.txt and write current index and tasks with IDs
(separated by " ``` ")
        if ctx.obj['TASKS']:
            curr_ind = [str(ctx.obj['LATEST'] + 1)]
```

```
            tasks = [str(i) + '```' + t for (i, t) in
ctx.obj['TASKS'].items()]
            with open('./todo.txt', 'w') as f:
                f.writelines(['%s\n' % en for en in curr_ind + tasks])
        else:
            #Resets ID tracker to 0 if list is empty
            with open('./todo.txt', 'w') as f:
                f.writelines([str(0) + '\n'])
    else:
        click.echo('Error: no task with id ' + str(fin_taskid))

if __name__ == '__main__':
    todo()
```

---

20. Currency Converter

```
import requests
import json
import sys
from pprint import pprint

# The below 4 lines bring out the value of currency from the api at
fixer.io.  I had to register there, the key is unique to me.
url = "http://data.fixer.io/api/latest?
access_key=33ec7c73f8a4eb6b9b5b5f95118b2275"
data = requests.get(url).text
data2 = json.loads(data) #brings whether request was successful,timestamp
etc
fx = data2["rates"]

currencies = [
    "AED : Emirati Dirham,United Arab Emirates Dirham",
    "AFN : Afghan Afghani,Afghanistan Afghani",
    "ALL : Albanian Lek,Albania Lek",
    "AMD : Armenian Dram,Armenia Dram",
    "ANG : Dutch Guilder,Netherlands Antilles
Guilder,Bonaire,Cura&#231;ao,Saba,Sint Eustatius,Sint Maarten",
    "AOA : Angolan Kwanza,Angola Kwanza",
    "ARS : Argentine Peso,Argentina Peso,Islas Malvinas",
    "AUD : Australian Dollar,Australia Dollar,Christmas Island,Cocos
(Keeling) Islands,Norfolk Island,Ashmore and Cartier Islands,Australian
Antarctic Territory,Coral Sea Islands,Heard Island,McDonald
Islands,Kiribati,Nauru",
    "AWG : Aruban or Dutch Guilder,Aruba Guilder",
    "AZN : Azerbaijan Manat,Azerbaijan Manat",
    "BAM : Bosnian Convertible Mark,Bosnia and Herzegovina Convertible
Mark",
    "BBD : Barbadian or Bajan Dollar,Barbados Dollar",
    "BDT : Bangladeshi Taka,Bangladesh Taka",
    "BGN : Bulgarian Lev,Bulgaria Lev",
```

```
    "BHD : Bahraini Dinar,Bahrain Dinar",
    "BIF : Burundian Franc,Burundi Franc",
    "BMD : Bermudian Dollar,Bermuda Dollar",
    "BND : Bruneian Dollar,Brunei Darussalam Dollar",
    "BOB : Bolivian Bol&#237;viano,Bolivia Bol&#237;viano",
    "BRL : Brazilian Real,Brazil Real",
    "BSD : Bahamian Dollar,Bahamas Dollar",
    "BTC : Bitcoin,BTC, XBT",
    "BTN : Bhutanese Ngultrum,Bhutan Ngultrum",
    "BWP : Botswana Pula,Botswana Pula",
    "BYN : Belarusian Ruble,Belarus Ruble",
    "BYR : Belarusian Ruble,Belarus Ruble",
    "BZD : Belizean Dollar,Belize Dollar",
    "CAD : Canadian Dollar,Canada Dollar",
    "CDF : Congolese Franc,Congo/Kinshasa Franc",
    "CHF : Swiss Franc,Switzerland Franc,Liechtenstein,Campione
d&#039;Italia,B&#252;singen am Hochrhein",
    "CLF : Chilean Unit of Account",
    "CLP : Chilean Peso,Chile Peso",
    "CNY : Chinese Yuan Renminbi,China Yuan Renminbi",
    "COP : Colombian Peso,Colombia Peso",
    "CRC : Costa Rican Colon,Costa Rica Colon",
    "CUC : Cuban Convertible Peso,Cuba Convertible Peso",
    "CUP : Cuban Peso,Cuba Peso",
    "CVE : Cape Verdean Escudo,Cape Verde Escudo",
    "CZK : Czech Koruna,Czech Republic Koruna",
    "DJF : Djiboutian Franc,Djibouti Franc",
    "DKK : Danish Krone,Denmark Krone,Faroe Islands,Greenland",
    "DOP : Dominican Peso,Dominican Republic Peso",
    "DZD : Algerian Dinar,Algeria Dinar",
    "EGP : Egyptian Pound,Egypt Pound,Gaza Strip",
    "ERN : Eritrean Nakfa,Eritrea Nakfa",
    "ETB : Ethiopian Birr,Ethiopia Birr,Eritrea",
    "EUR : Euro,Euro Member Countries,Andorra,Austria,Azores,Baleares
(Balearic Islands),Belgium,Canary Islands,Cyprus,Finland,France,French
Guiana,French Southern Territories,Germany,Greece,Guadeloupe,Holland
(Netherlands),Holy See (Vatican City),Ireland
(Eire),Italy,Luxembourg,Madeira
Islands,Malta,Monaco,Montenegro,Netherlands",
    "FJD : Fijian Dollar,Fiji Dollar",
    "FKP : Falkland Island Pound,Falkland Islands (Malvinas) Pound",
    "GBP : British Pound,United Kingdom Pound,United Kingdom
(UK),England,Northern Ireland,Scotland,Wales,Falkland
Islands,Gibraltar,Guernsey,Isle of Man,Jersey,Saint Helena and
Ascension,South Georgia and the South Sandwich Islands,Tristan da Cunha",
    "GEL : Georgian Lari,Georgia Lari",
    "GGP : Guernsey Pound,Guernsey Pound",
    "GHS : Ghanaian Cedi,Ghana Cedi",
    "GIP : Gibraltar Pound,Gibraltar Pound",
    "GMD : Gambian Dalasi,Gambia Dalasi",
    "GNF : Guinean Franc,Guinea Franc",
    "GTQ : Guatemalan Quetzal,Guatemala Quetzal",
    "GYD : Guyanese Dollar,Guyana Dollar",
    "HKD : Hong Kong Dollar,Hong Kong Dollar",
    "HNL : Honduran Lempira,Honduras Lempira",
    "HRK : Croatian Kuna,Croatia Kuna",
    "HTG : Haitian Gourde,Haiti Gourde",
    "HUF : Hungarian Forint,Hungary Forint",
    "IDR : Indonesian Rupiah,Indonesia Rupiah,East Timor",
    "ILS : Israeli Shekel,Israel Shekel,Palestinian Territories",
    "IMP : Isle of Man Pound,Isle of Man Pound",
    "INR : Indian Rupee,India Rupee,Bhutan,Nepal",
    "IQD : Iraqi Dinar,Iraq Dinar",
    "IRR : Iranian Rial,Iran Rial",
    "ISK : Icelandic Krona,Iceland Krona",
    "JEP : Jersey Pound,Jersey Pound",
    "JMD : Jamaican Dollar,Jamaica Dollar",
    "JOD : Jordanian Dinar,Jordan Dinar",
    "JPY : Japanese Yen,Japan Yen",
    "KES : Kenyan Shilling,Kenya Shilling",
    "KGS : Kyrgyzstani Som,Kyrgyzstan Som",
    "KHR : Cambodian Riel,Cambodia Riel",
    "KMF : Comorian Franc,Comorian Franc",
    "KPW : North Korean Won,Korea (North) Won",
    "KRW : South Korean Won,Korea (South) Won",
    "KWD : Kuwaiti Dinar,Kuwait Dinar",
    "KYD : Caymanian Dollar,Cayman Islands Dollar",
    "KZT : Kazakhstani Tenge,Kazakhstan Tenge",
    "LAK : Lao Kip,Laos Kip",
    "LBP : Lebanese Pound,Lebanon Pound",
    "LKR : Sri Lankan Rupee,Sri Lanka Rupee",
    "LRD : Liberian Dollar,Liberia Dollar",
    "LSL : Basotho Loti,Lesotho Loti",
    "LTL : Lithuanian litas",
    "LVL : Latvia Lats",
    "LYD : Libyan Dinar,Libya Dinar",
    "MAD : Moroccan Dirham,Morocco Dirham,Western Sahara",
    "MDL : Moldovan Leu,Moldova Leu",
    "MGA : Malagasy Ariary,Madagascar Ariary",
    "MKD : Macedonian Denar,Macedonia Denar",
    "MMK : Burmese Kyat,Myanmar (Burma) Kyat",
    "MNT : Mongolian Tughrik,Mongolia Tughrik",
    "MOP : Macau Pataca,Macau Pataca",
    "MRU : Mauritanian Ouguiya,Mauritania Ouguiya",
    "MUR : Mauritian Rupee,Mauritius Rupee",
    "MVR : Maldivian Rufiyaa,Maldives (Maldive Islands) Rufiyaa",
    "MWK : Malawian Kwacha,Malawi Kwacha",
    "MXN : Mexican Peso,Mexico Peso",
    "MYR : Malaysian Ringgit,Malaysia Ringgit",
    "MZN : Mozambican Metical,Mozambique Metical",
    "NAD : Namibian Dollar,Namibia Dollar",
    "NGN : Nigerian Naira,Nigeria Naira",
    "NIO : Nicaraguan Cordoba,Nicaragua Cordoba",
    "NOK : Norwegian Krone,Norway Krone,Bouvet Island,Svalbard,Jan
Mayen,Queen Maud Land,Peter I Island",
    "NPR : Nepalese Rupee,Nepal Rupee,India (unofficially near India-Nepal
border)",
    "NZD : New Zealand Dollar,New Zealand Dollar,Cook Islands,Niue,Pitcairn
```

```
Islands,Tokelau",
    "OMR : Omani Rial,Oman Rial",
    "PAB : Panamanian Balboa,Panama Balboa",
    "PEN : Peruvian Sol,Peru Sol",
    "PGK : Papua New Guinean Kina,Papua New Guinea Kina",
    "PHP : Philippine Peso,Philippines Peso",
    "PKR : Pakistani Rupee,Pakistan Rupee",
    "PLN : Polish Zloty,Poland Zloty",
    "PYG : Paraguayan Guarani,Paraguay Guarani",
    "QAR : Qatari Riyal,Qatar Riyal",
    "RON : Romanian Leu,Romania Leu",
    "RSD : Serbian Dinar,Serbia Dinar",
    "RUB : Russian Ruble,Russia Ruble,Tajikistan,Abkhazia,South Ossetia",
    "RWF : Rwandan Franc,Rwanda Franc",
    "SAR : Saudi Arabian Riyal,Saudi Arabia Riyal",
    "SBD : Solomon Islander Dollar,Solomon Islands Dollar",
    "SCR : Seychellois Rupee,Seychelles Rupee",
    "SDG : Sudanese Pound,Sudan Pound",
    "SEK : Swedish Krona,Sweden Krona",
    "SGD : Singapore Dollar,Singapore Dollar",
    "SHP : Saint Helenian Pound,Saint Helena Pound",
    "SLL : Sierra Leonean Leone,Sierra Leone Leone",
    "SOS : Somali Shilling,Somalia Shilling",
    "SRD : Surinamese Dollar,Suriname Dollar",
    "STN : Sao Tomean Dobra,S&#227;o Tom&#233; and Pr&#237;ncipe Dobra",
    "SVC : Salvadoran Colon,El Salvador Colon",
    "SYP : Syrian Pound,Syria Pound",
    "SZL : Swazi Lilangeni,eSwatini Lilangeni",
    "THB : Thai Baht,Thailand Baht",
    "TJS : Tajikistani Somoni,Tajikistan Somoni",
    "TMT : Turkmenistani Manat,Turkmenistan Manat",
    "TND : Tunisian Dinar,Tunisia Dinar",
    "TOP : Tongan Pa&#039;anga,Tonga Pa&#039;anga",
    "TRY : Turkish Lira,Turkey Lira,North Cyprus",
    "TTD : Trinidadian Dollar,Trinidad and Tobago Dollar,Trinidad,Tobago",
    "TWD : Taiwan New Dollar,Taiwan New Dollar",
    "TZS : Tanzanian Shilling,Tanzania Shilling",
    "UAH : Ukrainian Hryvnia,Ukraine Hryvnia",
    "UGX : Ugandan Shilling,Uganda Shilling",
    "USD : US Dollar,United States Dollar,America,American Samoa,American
Virgin Islands,British Indian Ocean Territory,British Virgin
Islands,Ecuador,El Salvador,Guam,Haiti,Micronesia,Northern Mariana
Islands,Palau,Panama,Puerto Rico,Turks and Caicos Islands,United States
Minor Outlying Islands,Wake Island,East Timor",
    "UYU : Uruguayan Peso,Uruguay Peso",
    "UZS : Uzbekistani Som,Uzbekistan Som",
    "VEF : Venezuelan Bol&#237;var,Venezuela Bol&#237;var",
    "VND : Vietnamese Dong,Viet Nam Dong",
    "VUV : Ni-Vanuatu Vatu,Vanuatu Vatu",
    "WST : Samoan Tala,Samoa Tala",
    "XAF : Central African CFA Franc BEAC,Communaut&#233; Financi&#232;re
Africaine (BEAC) CFA Franc BEAC,Cameroon,Central African
Republic,Chad,Congo/Brazzaville,Equatorial Guinea,Gabon",
    "XAG : Silver Ounce,Silver",
    "XAU : Gold Ounce,Gold",
    "XCD : East Caribbean Dollar,East Caribbean Dollar,Anguilla,Antigua and
Barbuda,Dominica,Grenada,The Grenadines and Saint Vincent,Montserrat",
    "XDR : IMF Special Drawing Rights,International Monetary Fund (IMF)
Special Drawing Rights",
    "XOF : CFA Franc,Communaut&#233; Financi&#232;re Africaine (BCEAO)
Franc,Benin,Burkina Faso,Ivory Coast,Guinea-
Bissau,Mali,Niger,Senegal,Togo",
    "XPF : CFP Franc,Comptoirs Fran&#231;ais du Pacifique (CFP)
Franc,French Polynesia,New Caledonia,Wallis and Futuna Islands",
    "YER : Yemeni Rial,Yemen Rial",
    "ZAR : South African Rand,South Africa Rand,Lesotho,Namibia",
    "ZMK : Zambian Kwacha,Zambia Kwacha",
    "ZMW : Zambian Kwacha,Zambia Kwacha",
    "ZWL : Zimbabwean Dollar,Zimbabwe Dollar",
]


# The below function calculates the actual conversion
def function1():
    query = input(
        "Please specify the amount of currency to convert, from currency,
to currency (with space in between).\nPress SHOW to see list of currencies
available. \nPress Q to quit. \n"
    )
    if query == "Q":
        sys.exit()
    elif query == "SHOW":
        pprint(currencies)
        function1()
    else:
        qty, fromC, toC = query.split(" ")
        fromC = fromC.upper()
        toC = toC.upper()
        qty = float(round(int(qty), 2))
        amount = round(qty * fx[toC] / fx[fromC], 2)
        print(f"{qty} of currency {fromC} amounts to {amount} of currency
{toC} today")


try:
    function1()
except KeyError:
    print("You seem to have inputted wrongly, retry!")
    function1()
```

---

21. Create a simple stopwatch

```
import tkinter as Tkinter
from datetime import datetime
```

```
counter = 0
running = False


def counter_label(label):
    def count():
        if running:
            global counter
            # To manage the intial delay.
            if counter == 0:
                display = 'Ready!'
            else:
                tt = datetime.utcfromtimestamp(counter)
                string = tt.strftime('%H:%M:%S')
                display = string

            label['text'] = display

            # label.after(arg1, arg2) delays by
            # first argument given in milliseconds
            # and then calls the function given as second argument.
            # Generally like here we need to call the
            # function in which it is present repeatedly.
            # Delays by 1000ms=1 seconds and call count again.
            label.after(1000, count)
            counter += 1

    # Triggering the start of the counter.
    count()


# start function of the stopwatch
def Start(label):
    global running
    running = True
    counter_label(label)
    start['state'] = 'disabled'
    stop['state'] = 'normal'
    reset['state'] = 'normal'


# Stop function of the stopwatch
def Stop():
    global running
    start['state'] = 'normal'
    stop['state'] = 'disabled'
    reset['state'] = 'normal'
    running = False


# Reset function of the stopwatch
def Reset(label):
    global counter
    counter = 0
    # If reset is pressed after pressing stop.
    if not running:
        reset['state'] = 'disabled'
        label['text'] = '00:00:00'
    # If reset is pressed while the stopwatch is running.
    else:
        label['text'] = '00:00:00'


root = Tkinter.Tk()
root.title("Stopwatch")

# Fixing the window size.
root.minsize(width=250, height=70)
label = Tkinter.Label(root, text='Ready!', fg='black', font='Verdana 30
bold')
label.pack()
f = Tkinter.Frame(root)
start = Tkinter.Button(f, text='Start', width=6, command=lambda:
Start(label))
stop = Tkinter.Button(f, text='Stop', width=6, state='disabled',
command=Stop)
reset = Tkinter.Button(f, text='Reset', width=6, state='disabled',
command=lambda: Reset(label))
f.pack(anchor='center', pady=5)
start.pack(side='left')
stop.pack(side='left')
reset.pack(side='left')
root.mainloop()
```

---

22. Python script to compress folders and files

```
import zipfile
import sys
import os


# compress file function
def zip_file(file_path):
    compress_file = zipfile.ZipFile(file_path + '.zip', 'w')
    compress_file.write(path, compress_type=zipfile.ZIP_DEFLATED)
    compress_file.close()


# Declare the function to return all file paths of the particular directory
def retrieve_file_paths(dir_name):
    # setup file paths variable
    file_paths = []

    # Read all directory, subdirectories and file lists
    for root, directories, files in os.walk(dir_name):
        for filename in files:
            # Create the full file path by using os module.
            file_path = os.path.join(root, filename)
            file_paths.append(file_path)

    # return all paths
    return file_paths


def zip_dir(dir_path, file_paths):
    # write files and folders to a zipfile
    compress_dir = zipfile.ZipFile(dir_path + '.zip', 'w')
    with compress_dir:
        # write each file separately
        for file in file_paths:
            compress_dir.write(file)


if __name__ == "__main__":
    path = sys.argv[1]

    if os.path.isdir(path):
        files_path = retrieve_file_paths(path)
        # print the list of files to be zipped
        print('The following list of files will be zipped:')
        for file_name in files_path:
            print(file_name)
        zip_dir(path, files_path)
    elif os.path.isfile(path):
        print('The %s will be zipped:' % path)
        zip_file(path)
    else:
        print('a special file(socket,FIFO,device file), please input file
or dir')
```

23. Find IMDB Ratings

```
from bs4 import BeautifulSoup
import requests
import pandas as pd
import os

# Setting up session
s = requests.session()

# List contaiting all the films for which data has to be scraped from IMDB
films = []

# Lists contaiting web scraped data
```

```
names = []
ratings = []
genres = []

# Define path where your films are present
# For eg: "/Users/utkarsh/Desktop/films"
path = input("Enter the path where your films are: ")

# Films with extensions
filmswe = os.listdir(path)

for film in filmswe:
    # Append into my films list (without extensions)
    films.append(os.path.splitext(film)[0])
    # print(os.path.splitext(film)[0])

for line in films:
    # x = line.split(", ")
    title = line.lower()
    # release = x[1]
    query = "+".join(title.split())
    URL = "https://www.imdb.com/search/title/?title=" + query
    print(URL)
    # print(release)
    try:
        response = s.get(URL)

        #getting contect from IMDB Website
        content = response.content

        # print(response.status_code)

        soup = BeautifulSoup(response.content, features="html.parser")
        #searching all films containers found
        containers = soup.find_all("div", class_="lister-item-content")
        for result in containers:
            name1 = result.h3.a.text
            name = result.h3.a.text.lower()

            # Uncomment below lines if you want year specific as well,
define year variable before this
            # year = result.h3.find(
            # "span", class_="lister-item-year text-muted unbold"
            # ).text.lower()

            #if film found (searching using name)
            if title in name:
                #scraping rating
                rating = result.find("div",class_="inline-block ratings-
imdb-rating")["data-value"]
                #scraping genre
                genre = result.p.find("span", class_="genre")
                genre = genre.contents[0]
                #appending name, rating and genre to individual lists
                names.append(name1)
                ratings.append(rating)
                genres.append(genre)



    except Exception:
        print("Try again with valid combination of tile and release year")

#storing in pandas dataframe
df = pd.DataFrame({'Film Name':names,'Rating':ratings,'Genre':genres})

#making csv using pandas
df.to_csv('film_ratings.csv', index=False, encoding='utf-8')
```

---

24. Web Scrapping Youtube Comment

```
from selenium import webdriver
import csv
import time

items=[]
driver=webdriver.Chrome(r"C:/Users/hp/Anaconda3/chromedriver.exe")

driver.get('https://www.youtube.com/watch?v=iFPMz36std4')

driver.execute_script('window.scrollTo(1, 500);')

#now wait let load the comments
time.sleep(5)

driver.execute_script('window.scrollTo(1, 3000);')


username_elems = driver.find_elements_by_xpath('//*[@id="author-text"]')
comment_elems = driver.find_elements_by_xpath('//*[@id="content-text"]')
for username, comment in zip(username_elems, comment_elems):
    item = {}
    item['Author'] = username.text
    item['Comment'] = comment.text
    items.append(item)
filename = 'C:/Users/hp/Desktop/commentlist.csv'
with open(filename, 'w', newline='', encoding='utf-8') as f:
    w = csv.DictWriter(f,['Author','Comment'])
    w.writeheader()
    for item in items:
        w.writerow(item)
```

---

25. Text To Speech

```python
from gtts import gTTS
import os
file = open("abc.txt", "r").read()

speech = gTTS(text=file, lang='en', slow=False)
speech.save("voice.mp3")
os.system("voice.mp3")
```

Thank you for reading!

Buy Me A Coffee