Robotic Process Automation (RPA)

Gilles Pilon

2024-08-03

Improving life one Python script at a time

Gilles Pilon

Abstract

Robotic Process Automation (RPA) uses software robots to handle repetitive computer tasks. These include things like moving data between applications, filling out forms, sending emails, handling a long list of repetitive actions within a program such as Systems, Applications, and Products (SAP) and Power BI (Microsoft 2024a), and extracting data from online applications. RPA interacts with different computer programs just like a human would, but much faster and more accurately. RPA frees up employees to focus on more complex work and improve accuracy, efficiency, and productivity. This article presents several approaches and examples of RPA.

Contents

1 Introduction

RPA provides several benefits:

- Automation of repetitive tasks to save time and effort
- Customization of specific needs
- Efficiency of a single code script to perform complex workflows
- Increase accuracy by eliminating human error by consistent, repeatable execution of tasks

2 Python libraries, scripts, and shell scripts for RPA

These are a few of many possibilities for implementing RPA.

- PyAutoGUI
- shell script
- Scrapy + Beautiful Soup + Requests + urllib
- Openpyxl + pandas
- Python Tesseract + OpenCV

2.1 PyAutoGUI

"PyAutoGUI lets your Python scripts control the mouse and keyboard, and interact with dialogues, to automate interactions with other applications that a user would perform manually. The API is designed to be simple. PyAutoGUI works on Windows, macOS, and Linux, and runs on Python 2 and 3." (Sweigart 2023)

PyAutoGUI has many benefits:

- Eliminate human errors
- Increase productivity
- Increase accuracy
- Reduce costs
- Scalable to complex workflows

PyAutoGUI has many capabilities:

- Keyboard control
- Mouse control
- Display message boxes (alert, confirm, prompt, password, ...
- Screen shot functions

PyAutoGUI has a few limitations:

- Dependency onf screen resolution
- Limited to visible elements
- Complexity in dynamic graphical user interfaces (GUIs)

2.2 shell script

A shell script is a plain text file of commands that is executed on a computer terminal, also known as a shell. Linux (BaSH gnu.org 2024, zsh Stephenson 2024), Mac (zsh Stephenson 2024), and Windows PowerShell (Microsoft 2024b) can easily execute shell scripts. Windows PowerShell offers similar functionality and has its own syntax for the commands.

2.3 Scrapy + Beautiful Soup + Requests + urllib

"Scrapy is a fast high-level web crawling and web scraping framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing." (scrapy.org 2024) Beautiful Soup (Richardson 2024) parses HTML and XML documents to create a tree-like representation of the document structure and extract data. "Requests is a simple, yet elegant, HTTP library. Requests allows you to send HTTP/1.1 requests extremely easily." (pypi.org 2023) "urllib is a package that collects several modules for working with URLs." (pypi.org 2024)

2.4 openpyxl + pandas

"openpyxl is a Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltm files." (Gazoni and Clark 2024) "pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language." (pandas.pydata.org 2024)

2.5 Python Tesseract + OpenCV

"Tesseract is an optical character recognition engine (libtesseract) and a command line program (tesseract)." (Tesseract OCR 2024) "Python Tesseract is a wrapper for Tesseract. It is also useful as a stand-alone invocation script to Tesseract image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, and others. Additionally, if used as a script, Python Tesseract will print the recognized text instead of writing it to a file." (Lee 2024) "OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source library that includes several hundreds of computer vision algorithms." (opencv.org 2024)

3 Examples

3.1 Automate sending an email (PyAutoGUI)

There are times when one needs to send a recurring email to a group of people and include multiple attachments. Preparing and sending an email takes time, ensuring you have an up-

to-date list of recipients, and enclosing the attachments. This can easily be done with PyAutoGUI (Sweigart 2023).

```
#! /usr/bin/env python3
'''
Send an Outlook email, multiple recipients, multiple attachments, body text.
Execute:
- revise message, emails, attachments, directory as required
- in a terminal: python -m email_demo.py
- add to Windows Task Scheduler or Linux cron for timed execution
'''

import sys


def excepthook(*args, **kwargs):
    """
    Diagnose missing packages.
    This is used because the main script may be executed by Task Scheduler
    and prevents an error from closing a terminal.
    """
    print(*args, kwargs)
    from traceback import print_tb
    print_tb(args[2])
    input('Press <Enter> to exit.')


sys.excepthook = excepthook


from pathlib import Path
import time

import pyautogui as pyg
import dawgdad as dd


def main():
    message = [
        'This is the first line of text with a line break.\n\n'
        'This is the second line, first sentence. '
        'This is the second sentence.\n\n'
        '<user>'
    ]
    emails = ['gillespilon13@gmail.com']
    attachments = ['presentation.pptx', 'data.csv']
    directory = '~/<user>/documents/'
    header_title = 'email demonstration'
    output_url = 'email_demo.html'
    header_id = 'email_demo'
    original_stdout = dd.html_begin(
        output_url=output_url,
        header_title=header_title,
        header_id=header_id
    )
    dd.script_summary(
        script_path=Path(__file__),
        action='started at'
    )
    start_time = time.perf_counter()
    path_files = Path(Path.home().resolve(), directory)
    # create an email window in Outlook
    pyg.hotkey('win')
    time.sleep(1)
    pyg.write('out')
    time.sleep(1)
    pyg.hotkey('enter')
    time.sleep(5)
```

```
    pyg.hotkey('ctrl', 'n')
    time.sleep(1)
    for email in emails:
        print(email)
        print('tab')
    for email in emails:
        pyg.write(email)
        pyg.press('tab')
    pyg.press('tab')
    pyg.write('Test of PyAutoGUI attachment')
    pyg.hotkey('enter')
    pyg.hotkey('enter')
    time.sleep(0.5)
    for item in message:
        time.sleep(0.5)
        pyg.write(item)
    time.sleep(0.5)
    for attachment in attachments:
        print('TEST START')
        print(path_files)
        print(attachment)
        path = Path(path_files, attachment)
        print(path)
        print('TEST END')
        print()
        pyg.keyDown('alt')
        pyg.press('6')
        pyg.keyUp('alt')
        time.sleep(0.5)
        pyg.keyDown('alt')
        pyg.press('n')
        pyg.keyUp('alt')
        time.sleep(0.5)
        pyg.write(str(path))
        pyg.keyDown('alt')
        pyg.press('s')
        pyg.keyUp('alt')
    pyg.keyDown('alt')
    pyg.press('s')
    pyg.keyUp('alt')
    print('emails sent to:')
    print()
    dd.print_list_by_item(list=emails)
    print()
    stop_time = time.perf_counter()
    dd.script_summary(
        script_path=Path(__file__),
        action='finished at'
    )
    dd.report_summary(
        start_time=start_time,
        stop_time=stop_time
    )
    dd.html_end(
        original_stdout=original_stdout,
        output_url=output_url
    )


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

3.2 Automate updating repositories (shell script)

A workgroup often requires access to one or more code repositories, even if they are users and not developers of code. There are several powerful code repositories [GitHub (GitHub, Inc. 2024), GitLab (GitLab Inc. 2024), Heptapod FOSS (Heptapod FOSS 2024)] to store and access code. This can easily be done with a shell script. The following script works with BaSH (gnu.org 2024) and zsh (Stephenson 2024).

```
#! /bin/sh
# update clones of GitHub repositories
# in a terminal: sh update_repositories.sh
echo
# change path to home directory
cd ~/
# update dawgdad
echo 'update dawgdad repository'
echo
pip install --upgrade dawgdad
# update packages
echo
echo 'update required Python packages'
echo
pip install --user -U pandoc pyautogui pyarrow tabulate
# git pull gitlab_code
echo
echo 'git pull on GitLab code repository'
echo
cd Documents/gitlab_code
git pull
# git pull gitlab_job_aids
echo
echo 'git pull on GitLab job aids repository'
echo
cd ../gitlab_job_aids
git pull
# convert mkd -> html
cd fol
./update_html.sh
```

3.3 Automate SAP report (PyAutoGUI)

An SAP user will be granted access to various reports and portions of the SAP warehouse. If a user needs data extracted from a report, and is not trained in how to create one, it often takes a long time to get a report written and made available in the user's profile. An alternative is to mimic the mouse and keyboard movements of manually access an existing report to extract and compile the data required. The following Python script logs into SAP, navigates to a specific report, extracts the data for multiple prior months, and saves the data in csv files in a particular directory for further processing. This script is easily scheduled with Windows Task Scheduler.

```
#! /usr/bin/env python3
'''
Execute SAP spend report for a number of prior calendar months.
In a terminal: python -m sap_spend_report.py
'''

import sys


def excepthook(*args, **kwargs):
    """
    Diagnose missing packages.
    This is used because the main script may be executed by Task Scheduler
```

```
    and prevents an error from closing a terminal.
    """
    print(*args, kwargs)
    from traceback import print_tb
    print_tb(args[2])
    input('Press <Enter> to exit.')


sys.excepthook = excepthook


from datetime import datetime
from pathlib import Path
import subprocess
import time

from dateutil.relativedelta import relativedelta
import pyautogui as pyg
import dawgdad as dd


def main():
    # define parameters
    start_time_0 = time.perf_counter()
    sap_process = r"C:\Program Files (x86)\SAP\FrontEnd\SapGui\saplogon.exe"
    directory_files = 'supplier_risk/supplier_risk_data_input'
    directory_gitlab_code = 'Documents/gitlab_code'
    output_url = 'sap_spend_report.html'
    path_home = Path.home().resolve()
    header_title = 'SAP spend report'
    header_id = 'sap-spend-report'
    number_months_prior_start = 1
    number_months_prior_end = 1
    original_stdout = dd.html_begin(
        output_url=output_url,
        header_title=header_title,
        header_id=header_id
    )
    dd.script_summary(
        script_path=Path(__file__),
        action='started at'
    )
    stop_time_0 = time.perf_counter()
    elapsed_time = stop_time_0 - start_time_0
    print(
        'Set-up time:',
        round(elapsed_time, 3),
        's'
    )
    print()
    # create list of dates
    dates = [
            f"{(datetime.today() + relativedelta(months=-x)):%Y%m}"
        for x in range(number_months_prior_start, number_months_prior_end + 1)
    ]
    # use PyAutoGUI to create csv file from
    # SAP > BWP > Global Vendor Detail Spend Report_New
    for date in dates:
        path_file_name = Path(date + '_vendor_data.csv')
        path_output_file = Path(
            path_home, directory_gitlab_code, directory_files, path_file_name
        )
        print('Started:', path_output_file)
        # quit SAP and Excel
        dd.quit_sap_excel()
        start_time_1 = time.perf_counter()
        # open SAP > BWP report
        subprocess.Popen(sap_process)
```

```
time.sleep(20)
pyg.write('09')
pyg.keyDown('enter')
time.sleep(20)
pyg.hotkey('ctrl', 'f')
time.sleep(5)
pyg.write('global vendor detail spend report_new')
pyg.keyDown('enter')
time.sleep(5)
pyg.hotkey('f2')
# fill fields in report
# this is critical to allow SAP time
time.sleep(90)
for x in range(0, 9):
    pyg.hotkey('tab')
pyg.write(date)
for x in range(0, 2):
    pyg.hotkey('tab')
pyg.write(date)
pyg.keyDown('enter')
# wait for Excel workbook to open
# this is critical to allow SAP time
time.sleep(90)
# select and copy data
pyg.hotkey('win', 'up')
time.sleep(2)
pyg.hotkey('ctrl', 'a')
time.sleep(2)
pyg.keyDown('alt')
pyg.press('h')
pyg.press('c')
pyg.press('c')
pyg.keyUp('alt')
time.sleep(2)
# paste data in blank workbook
pyg.keyDown('alt')
pyg.press('f')
time.sleep(1)
pyg.press('n')
time.sleep(1)
pyg.press('l')
pyg.keyUp('alt')
time.sleep(1)
pyg.press('left')
pyg.keyDown('alt')
pyg.press('h')
pyg.press('v')
pyg.press('v')
pyg.keyUp('alt')
time.sleep(1)
pyg.press('up')
time.sleep(1)
# delete row above labels
pyg.keyDown('alt')
pyg.press('h')
pyg.press('d')
pyg.press('r')
pyg.keyUp('alt')
time.sleep(1)
# save as csv file
pyg.keyDown('alt')
pyg.press('f')
time.sleep(1)
pyg.press('a')
time.sleep(1)
pyg.press('c')
time.sleep(1)
pyg.press('1')
```

```
        time.sleep(1)
        pyg.keyUp('alt')
        pyg.write(str(path_output_file))
        time.sleep(1)
        pyg.hotkey('tab')
        time.sleep(1)
        pyg.press('down')
        time.sleep(1)
        pyg.press('w')
        time.sleep(1)
        pyg.press('up')
        time.sleep(1)
        pyg.press('up')
        time.sleep(1)
        pyg.press('up')
        time.sleep(1)
        pyg.keyDown('enter')
        time.sleep(1)
        pyg.keyDown('alt')
        pyg.press('s')
        time.sleep(1)
        pyg.press('y')
        time.sleep(1)
        pyg.keyUp('alt')
        print('Created:', path_output_file)
        print(f'Size    : {path_output_file.stat().st_size/1024:.0f} KiB')
        stop_time_1 = time.perf_counter()
        elapsed_time = stop_time_1 - start_time_1
        print(
            'Execution time:',
            round(elapsed_time, 3),
            's'
        )
        print()
    # quit SAP and Excel
    dd.quit_sap_excel()
    stop_time_2 = time.perf_counter()
    dd.script_summary(
        script_path=Path(__file__),
        action='finished at'
    )
    dd.report_summary(
        start_time=start_time_0,
        stop_time=stop_time_2
    )
    dd.html_end(
        original_stdout=original_stdout,
        output_url=output_url
    )


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

3.4 Automate data extraction from Power BI (PyAutoGUI)

A Power BI (Microsoft 2024a) user does not normally have access to the data behind the reports. If a user needs data extracted from a report, it often takes a long time to get a report written and made available in the Power BI report. An alternative is to mimic the mouse and keyboard movements of manually access an existing report to extract and compile the data required. The following Python script creates a csv file from a Power BI dashboard. This script is easily scheduled with Windows Task Scheduler.

```
#! /usr/bin/env python3
'''
```

```
Create a csv file from a SharePoint dashboard.
'''

import sys


def excepthook(*args, **kwargs):
    """
    Diagnose missing packages.
    This is used because the main script may be executed by Task Scheduler
    and prevents an error from closing a terminal.
    """
    print(*args, kwargs)
    from traceback import print_tb
    print_tb(args[2])
    input('Press <Enter> to exit.')


sys.excepthook = excepthook


from typing import List
from pathlib import Path
import webbrowser
import time

import pyautogui as pyg
import dawgdad as dd
import pandas as pd


def main():
    # define parameters
    reorder_columns = [
        'Vendor', 'Vendor_Name', 'Notifications', 'Material_Group_Number',
        'Months'
    ]
    dashboard_url = (
        'https://app.powerbi.com/groups/me/reports/'
        '8daf409d-c0a7-40a4-b979-4cc28bbf363b/ReportSection?'
        'ctid=101ce67d-13f2-447a-bb65-0989b89dfdb4'
    )
    rename_columns_initially = {
        'Supplier': 'Vendor',
        'Sum of Number of notifications': 'Notifications'
    }
    path_final_file = Path('qn_notifications.csv')
    output_url = 'report.html'
    header_title = 'report'
    path_files = Path('../../Downloads')
    header_id = 'report'
    rename_columns_finally = {
        'VN': 'Vendor',
        'VT': 'Vendor_Name'
    }
    material_groups = [
        '101 - Plastic Resins',
        '102 - Metal Raw Materials',
        '103 - Die Castings',
        '201 - Non-Metallic Parts',
        '202 - Metal & Mech Parts',
        '203 - Wire & Cable',
        '204 - Conn Assy&Subcon Mfg',
        # '205 - Resale Tools & Press'
        '206 - Electronic Comp',
        '207 - Electronic Assy',
        '208 - Competitor Products',
        '209 - PCB, FPC and FFC',
```

```
    '301 - Packaging'
]
extensions = ['.xlsx', '.XLSX']
months = [12, 6]
start_time = time.perf_counter()
material_group_number = [int(x[:3]) for x in material_groups]
number_files_created = len(months) * len(material_groups)
combinations = [
    (mg, month) for mg in material_group_number for month in months
]
original_stdout = dd.html_begin(
    output_url=output_url,
    header_title=header_title,
    header_id=header_id
)
dd.script_summary(
    script_path=Path(__file__),
    action='started at'
)
# open Dashboard
webbrowser.open(url=dashboard_url)
time.sleep(30)
# create individual files
for mg in material_groups:
    material_group(material_group=mg)
    for month in months:
        change_date(month=month)
        export_file()
        time.sleep(2)
    deselect_material_group()
# create list of paths, sorted newest to oldest modification time
files_in_downloads = sorted(
    dd.list_files(
        directory=path_files,
        patterns=extensions
    ),
    key=dd.get_mtime,
    reverse=False
)[-number_files_created:]
print_file_size(files=files_in_downloads)
# create final dataframe
dfs = pd.DataFrame()
for file, (x, y) in zip(files_in_downloads, combinations):
    # create individual DataFrames
    df = dd.read_file(
        file_name=file,
        skiprows=[0, 1]
    ).rename(columns=rename_columns_initially)
    df['VN'] = df['Vendor'].str[:6]
    df['VT'] = df['Vendor'].str[9:]
    df['Material_Group_Number'] = x
    df['Months'] = y
    df = df.drop(columns=['Vendor'])
    df = df.rename(columns=rename_columns_finally)
    df = df[reorder_columns]
    dfs = dfs.append(df)
# remove rows where Vendor contains '-'
dfs = dfs.loc[~dfs['Vendor'].str.contains(pat='-', na=False)]
# create final file
dd.save_file(
    df=dfs,
    file_name=Path(path_final_file)
)
# delete individual files
for file in files_in_downloads:
    Path(file).unlink(missing_ok=True)
stop_time = time.perf_counter()
dd.script_summary(
```

```
        script_path=Path(__file__),
        action='finished at'
    )
    dd.report_summary(
        start_time=start_time,
        stop_time=stop_time
    )
    dd.html_end(
        original_stdout=original_stdout,
        output_url=output_url
    )


def change_date(
    *,
    month: int
) -> None:
    '''
    Change date and format.
    '''
    # change date
    pyg.moveTo(460, 335)
    time.sleep(2)
    pyg.doubleClick()
    time.sleep(2)
    pyg.write(str(month))
    time.sleep(2)
    # change to calendar months
    pyg.hotkey('tab')
    pyg.press('down')
    for x in range(1, 9):
        pyg.press('up')
        time.sleep(0.2)
    for x in range(1, 6):
        pyg.press('down')
        time.sleep(0.2)
    pyg.keyDown('enter')
    time.sleep(2)


def material_group(
    *,
    material_group: str
) -> None:
    '''
    Select the material group.
    '''
    pyg.moveTo(420, 755)
    time.sleep(2)
    pyg.click()
    pyg.moveTo(420, 510)
    time.sleep(2)
    pyg.click()
    pyg.write(material_group)
    time.sleep(2)
    pyg.moveTo(420, 535)
    time.sleep(2)
    pyg.click()
    time.sleep(2)


def deselect_material_group() -> None:
    '''
    Select all material groups available.
    '''
    pyg.moveTo(420, 755)
    time.sleep(2)
    pyg.click()
```

```
    pyg.moveTo(420, 510)
    time.sleep(2)
    pyg.click()
    time.sleep(2)
    pyg.hotkey('ctrl', 'a', 'backspace')
    time.sleep(2)
    pyg.moveTo(420, 540)
    time.sleep(2)
    pyg.click()
    time.sleep(2)
    pyg.click()
    time.sleep(2)
    pyg.moveTo(250, 755)
    pyg.click()
    time.sleep(2)


def export_file() -> None:
    '''
    Export the data.
    '''
    # export file
    pyg.moveTo(1150, 645)
    time.sleep(2)
    pyg.click()
    pyg.moveTo(1150, 610)
    time.sleep(2)
    pyg.click()
    pyg.moveTo(1230, 725)
    time.sleep(2)
    pyg.click()
    pyg.moveTo(1200, 915)
    time.sleep(2)
    pyg.click()
    pyg.moveTo(1840, 250)
    time.sleep(2)
    pyg.click()
    # close downloads bar
    pyg.moveTo(1890, 1035)
    time.sleep(2)
    pyg.click()
    time.sleep(2)


def print_file_size(
    *,
    files: List[str]
) -> None:
    '''
    Print the file size for all individual files created.
    '''
    for file in files:
        print('Created:', file)
        print(f'Size    : {file.stat().st_size/1024:.0f} KiB')
        print()


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

3.5 Automate data extraction from Wikipedia table (pandas)

There are many libraries to perform web scraping. The best one is Scrapy (scrapy.org 2024). However, for many web scraping tasks one might only need the pandas library (pandas.pydata.org

2024) which has several methods (pd.read_html, df.to_html, pd.read_xml, df.to_xml).

```
#! /usr/bin/env python3
"""
Get country codes, names, domains from Wikipedia table and create csv file.
Execute in a terminal: python -m country_codes_names_domains.py
"""

import sys


def excepthook(*args, **kwargs):
    """
    Diagnose missing packages.
    This is used because the main script may be executed by Task Scheduler
    and prevents an error from closing a terminal.
    """
    print(*args, kwargs)
    from traceback import print_tb

    print_tb(args[2])
    input("Press <Enter> to exit.")


sys.excepthook = excepthook


from pathlib import Path
from os import chdir
import time

import dawgdad as dd
import pandas as pd


def main():
    # required for cron
    chdir(Path(__file__).parent.resolve())
    # define parameters
    start_time = time.perf_counter()
    wiki_url = "https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2"
    output_url = "country_codes_names_domains.html"
    header_title = "Country codes, names, domains"
    column_labels = ["Code", "Country", "Domain"]
    header_id = "country-codes-names-domains"
    path_file_out = Path("country_codes.csv")
    drop_columns = ["Year", "Notes"]
    original_stdout = dd.html_begin(
        output_url=output_url, header_title=header_title, header_id=header_id
    )
    dd.script_summary(script_path=Path(__file__), action="started at")
    # create DataFrame from Wikipedia table
    data = (
        pd.DataFrame(
            data=pd.read_html(
                io=wiki_url,
                attrs={"class": "wikitable sortable"},
                keep_default_na=False,
            )[0]
        )
        .drop(labels=drop_columns, axis="columns")
        .set_axis(labels=column_labels, axis="columns")
    )
    # save DataFrame as csv file
    dd.save_file(df=data, file_name=path_file_out)
    print(data)
    print()
    stop_time = time.perf_counter()
```

```
    dd.script_summary(script_path=Path(__file__), action="finished at")
    dd.report_summary(
        start_time=start_time,
        stop_time=stop_time,
        save_file_names=path_file_out.resolve(),
    )
    dd.html_end(original_stdout=original_stdout, output_url=output_url)


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

4 dawgdad

dawgdad (Pilon 2024) is a Python package for statistical, graphical, and predictive analysis. Instructions on how to use dawgdad are provided in relevant sections of this document. To use dawgdad, install a Python distribution appropriate to your operating system. Then install dawgdad following the instructions at https://github.com/gillespilon/dawgdad. dawgdad is compatible with the latest Python distribution from Python.org. Instructions on how to use dawgdad are provided in relevant sections of this document. Full documentation is available at https://dawgdad.readthedocs.io/en/latest/.

5 Abbreviations

GUI graphical user interface 4

RPA Robotic Process Automation 1, 3

SAP Systems, Applications, and Products 1, 7

6 Glossary

B

Beautiful Soup It parses HTML and XML documents to create a tree-like representation of the document structure and extract data. (Richardson 2024) 4

G

GitHub It is a cloud-based platform where you can store, share, and work together with others to write code. 7

GitLab It is a cloud-based platform where you can store, share, and work together with others to write code. 7, 19

graphical user interface It is a visual interface that allows users to interact with a computer or other electronic device using icons, buttons, and menus. GUIs are interactive and replace text-based commands with user-friendly actions. The goal of a GUI is to make decision points easy for users to find, understand, and use.

H

Heptapod FOSS It is a web-based platform designed to bring Mercurial support to GitLab. 7

O

openpyxl It is a Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltm files. (Gazoni and Clark 2024) 4

P

pandas It is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. (pandas.pydata.org 2024) 4, 14

Power BI It is a collection of software services, apps, and connectors designed to transform data into clear and actionable insights. It falls under the umbrella of Business Intelligence (BI) tools, which help businesses analyze data to make informed decisions. (Microsoft 2024a) 1, 10

PyAutoGUI It lets your Python scripts control the mouse and keyboard to automate interactions with other applications. The API is designed to be simple. It works on Windows, macOS, and Linux, and runs on Python 2 and 3." (Sweigart 2023) 3–5

Python Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. (Python Software Foundation 2024) 3, 4, 17, 19

Python script A Python script is a plain text file of Python commands that is executed on a compute terminal, also known as a shell. A Python script is typically saved in a plain text file with a .py extension. To run a Python script, you need a Python interpreter installed on your computer. The interpreter reads the code in the script line by line, translates it into machine code the computer understands, and executes the instructions. 3, 7, 10, 19

Python Tesseract Python Tesseract is a wrapper for Tesseract. It is also useful as a standalone invocation script to tesseract, as it can read all image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, and others. Additionally, if used as a script, Python Tesseract will print the recognized text instead of writing it to a file. (Lee 2024) 4

R

Requests It is a simple, yet elegant, HTTP library. Requests allows you to send HTTP/1.1 requests extremely easily. (pypi.org 2023) 4

Robotic Process Automation It uses software robots to handle repetitive computer tasks. The benefits include reduced execution time, customization for specific needs, perform complex workflows, and consistent repeatability to eliminate human errors.

S

Scrapy Scrapy is a fast high-level web crawling and web scraping framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing. (scrapy.org 2024 4, 14

shell script A shell script is a plain text file of commands that is executed on a compute terminal, also known as a shell. Linux (gnu.org 2024, Stephenson 2024), Mac (Stephenson 2024), and Windows PowerShell (Microsoft 2024b) can easily execute shell scripts. Windows PowerShell offers similar functionality and has its own syntax for the commands. 4, 7

software robot It is a software program that automates repetitive, rule-based tasks on a computer. 1

Systems, Applications, and Products It is a suite of software solutions developed by the German company SAP SE. SAP software is primarily focused on Enterprise Resource Planning (ERP), which means it helps businesses manage and integrate various core functions.

T

Tesseract It is an optical character recognition engine (libtesseract) and a command line program (tesseract). (Tesseract OCR 2024) 4, 19

U

urllib It is a package that collects several modules for working with URLs. (pypi.org 2024) 4

W

web scraping It is the practice of gathering data through any means other than a program interacting with an API (or, obviously, through a human using a web browser). This is most commonly accomplished by writing an automated program that queries a web server, requests data (usually in the form of HTML and other files that compose web pages), and then parses that data to extract needed information. (Mitchell 2024) 4, 14, 20

Windows PowerShell It is a powerful task automation and configuration management framework designed specifically for the Windows operating system. It combines the features of a command-line shell with a scripting language, offering a versatile environment for system administrators and power users. (Microsoft 2024b) 4

Windows Task Scheduler is a built-in utility that allows you to automate the execution of programs, scripts, or various tasks at specific times or intervals on a Windows computer. (Microsoft 2024c) 7, 10

7 References

Gazoni, Eric and Charlie Clark (2024). *openpyxl*. Version 3.1.2. URL: https://foss.heptapod.net/openpyxl/openpyxl.

GitHub, Inc. (2024). *GitHub*. URL: https://github.com/.

GitLab Inc. (2024). *GitLab*. URL: https://about.gitlab.com/.

gnu.org (2024). *bash*. Version 5.2. URL: https://git.savannah.gnu.org/cgit/bash.git.

Heptapod FOSS (2024). *Heptapod FOSS*. URL: https://foss.heptapod.net/.

Lee, Matthias A. (Apr. 5, 2024). *Python Tesseract*. Version 0.3.13-2. URL: https://github.com/madmaze/pytesseract.

Microsoft (2024a). *Power BI*. Version 2.128.751.0. URL: https://learn.microsoft.com/en-us/training/powerplatform/power-bi.

– (2024b). *PowerShell*. Version 5.1. URL: https://learn.microsoft.com/en-us/powershell/.

– (2024c). *Windows Task Scheduler*. URL: https://learn.microsoft.com/en-us/windows/win32/taskschd/about-the-task-scheduler.

Mitchell, Ryan (2024). *Web Scraping with Python. Data Extraction from the Modern Web*. 3rd ed. Sebastopol, CA: O'Reilly Media, Inc., p. 331. URL: https://www.oreilly.com/catalog/errata.csp?isbn=0636920830702.

opencv.org (Apr. 28, 2024). *OpenCV*. Version 4.9.0-5. URL: https://github.com/opencv/opencv.

pandas.pydata.org (2024). *pandas*. Version 2.2.2. URL: https://github.com/pandas-dev/pandas.

Pilon, Gilles (July 26, 2024). *dawgdad*. Version 1.0.4. URL: https://github.com/gillespilon/dawgdad.

pypi.org (2023). *Requests*. Version 2.31.0. URL: https://github.com/psf/requests.

– (2024). *urllib*. Version 2.2.1. URL: https://github.com/urllib3/urllib3.

Python Software Foundation (2024). *Python*. URL: https://www.python.org/.

Richardson, Leonard (2024). *Beautiful Soup*. Version 4.12.3. URL: https://launchpad.net/beautifulsoup.

scrapy.org (Feb. 14, 2024). *Scapy*. Version 2.11.1. URL: https://github.com/scrapy/scrapy.

Stephenson, Peter (2024). *zsh*. Version 5.9. URL: https://sourceforge.net/p/zsh/code/ci/master/tree/.

Sweigart, Al (2023). *PyAutoGUI*. Version 0.9.54. URL: https://github.com/asweigart/pyautogui.

Tesseract OCR (Apr. 5, 2024). *Tesseract*. Version 5.3.4-1. URL: https://github.com/tesseract-ocr/tesseract.