  
Que:  
pattern:   
  
  
Note :we have to import re module to use regex in python.

# Getting started with RegEx in Python

The [**re**](https://docs.python.org/3/howto/regex.html) module provides an interface to the regular expression engine, allowing you to **compile regular expressions into objects and then perform matches with them**.

**import** re

## 1. Compiling Regular Expressions

Regular expressions are **compiled** into Pattern objects, which have methods for various operations such as searching for pattern matches or performing string substitutions.

### re.compile(pattern, flags=0)

Compile a regular expression pattern, returning a pattern object.

* The regular expression is passed to re.compile() as a **string**.

Regular expressions are handled as strings because regular expressions aren’t part of the core Python language, and no special syntax was created for expressing them.

Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C.

pattern **=** re**.**compile("hello")

print(pattern)

output:

re.compile(r'hello', re.UNICODE)

* re.compile() also accepts an optional flags argument, used to enable various special features and syntax variations. [More about flags](http://xahlee.info/python/python_regex_flags.html)

In the example below, we use the flag re.I (short for re.IGNORECASE) to ignore letter case in the regex pattern.

pattern **=** re**.**compile("hello", flags**=**re**.**I)

print(pattern)

Output:

re.compile(r'hello', re.IGNORECASE|re.UNICODE)

## 2. Performing Matches

So, we have created a Pattern object representing a compiled regular expression using re.compile() method.

Pattern objects have several methods and attributes.

Here is the list of different methods used for performing matches:

| **Method/Attribute** | **Purpose** |
| --- | --- |
| match() | Determine if the RE matches at the beginning of the string. |
| search() | Scan through a string, looking for any location where this RE matches. |
| findall() | Find all substrings where the RE matches, and returns them as a list. |
| finditer() | Find all substrings where the RE matches, and returns them as an iterator. |

Let us go through them one by one:

### match(string[, pos[, endpos]])

* A match is checked only at the beginning (by default).
* Checking starts from pos index of the string. (default is 0)
* Checking is done until endpos index of string. endpos is set as a very large integer (by default).
* Returns None if no match found.
* If a match is found, a Match object is returned, containing information about the match: where it starts and ends, the substring it matched, and more.

pattern **=** re**.**compile("hello")

match **=** pattern**.**match("hello world")

**match.**span() ## this will tell you the index inbetween which index ur pattern has matched.

(0, 5)

**match.**start()

0

**match.**end()

5

pattern**.**match("say hello", pos**=**4) **is** **None**

False

pattern**.**match("hello", endpos**=**4) **is** **None**

True

### search(string[, pos[, endpos]])

* A search is checked throughtout the string.
* Same behaviour of pos and endpos as the match() function.
* Returns None if no match found.
* If a match is found, a Match object is returned.

Pattern=”hello”

pattern**.**search("say hello")

OutPut : <\_sre.SRE\_Match object; span=(4, 9), match='hello'>

pattern**.**search("say hello hello")

O/P: <\_sre.SRE\_Match object; span=(4, 9), match='hello'>

### findall(string[, pos[, endpos]])

* Finds **all non-overlapping substrings** where the match is found, and returns them as a list.
* Same behaviour of pos and endpos as the match() and search() function.

In [15]:

Pattern=”hello”

pattern**.**findall("say hello hello")

Out[15]:

['hello', 'hello']

### finditer(string[, pos[, endpos]])

* Finds **all non-overlapping substrings** where the match is found, and returns them as an iterator of the Match objects.
* Same behaviour of pos and endpos as the match(), search() and findall() function.

In [16]:

matches **=** pattern**.**finditer("say hello hello")

In [17]:

**for** match **in** matches:

print(match**.**span())

(4, 9)

(10, 15)

In [18]:

**from** utils **import** highlight\_regex\_matches

highlight\_regex\_matches(pattern, "say hello hello")

say **hello** **hello**

By now, you must have noticed that match(), search() and finditer() return Match object(s) where as findall() returns a list of strings.

### Note:

It is not mandatory to create a Pattern object explicitly using re.compile() method in order to perform a regex operation.

You can direclty use the module level functions such as:

pattern=”hello”

* re.match(pattern, string, flags=0)
* re.search(pattern, string, flags=0)
* re.findall(pattern, string, flags=0)
* re.finditer(pattern, string, flags=0)

### **Key Differences:**

1. **Scope of Searching**:
   * match(): Only at the beginning of the string.
   * search(): Anywhere in the string.
   * findall(): All non-overlapping occurrences in the string.
   * finditer(): All non-overlapping occurrences in the string, but returns match objects.
2. **Return Type**:
   * match(): A match object or None(Only at the beginning of the string).
   * search(): A match object or None(Anywhere in the string).
   * findall(): A list of strings (all matches).
   * finditer(): An iterator yielding match objects, so you can iterate using loop.
3. **Use Cases**:
   * Use match() when you need to check if a string starts with a pattern.
   * Use search() when you need to find the first occurrence of a pattern.
   * Use findall() when you need all matches as a list.
   * Use finditer() when you need all matches as match objects, which provide additional information about the matches (such as their positions in the string).

and so on.

In a module level function, you can simply pass a **string** as your **regex pattern** as shown in the examples below.

In [19]:

re**.**match("hello", "hello")

Out[19]:

<\_sre.SRE\_Match object; span=(0, 5), match='hello'>

In [20]:

re**.**findall("hello", "say hello hello")

Out[20]:

['hello', 'hello']

### Important Example

Consider the example below:

In [21]:

txt **=** "This book costs $15."

Search for the pattern $15.

In [22]:

pattern **=** re**.**compile("$15")

In [23]:

pattern**.**search(txt)

### No match found. Why?

$ is a metacharacter and has a special meaning for regex engine. Here, we want to treat it like a literal.

In order to treat a metacharacter like a literal, you need to **escape** it using \ character.

In [24]:

pattern **=** re**.**compile("\$15")

In [25]:

pattern**.**search(txt)

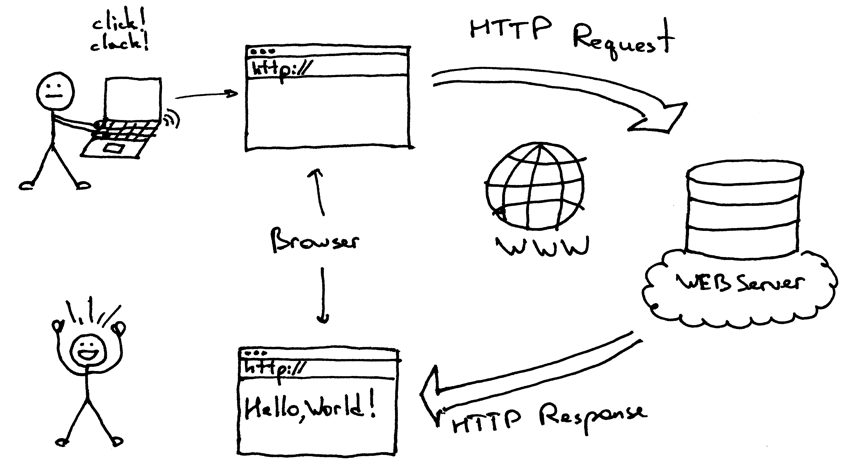
Out[25]:

<\_sre.SRE\_Match object; span=(16, 19), match='$15'>

In regular expressions, there are twelve metacharacters that should be escaped if they are to be used with their literal meaning:

* Backslash \
* Caret ^
* Dollar sign $
* Dot .
* Pipe symbol |
* Question mark ?
* Asterisk \*
* Plus sign +
* Opening parenthesis (
* Closing parenthesis )
* Opening square bracket [
* The opening curly brace {

###########Project relate to regex (Git- repo scrapper:-  
  
  
  
Each step to create a program that fetches repositories from a GitHub user, scans the files for specific patterns, and presents the findings. By the end, you will have a fully functional tool that can find URLs, emails, phone numbers, API keys, usernames, and passwords within GitHub repositories.  
**1.Introduction:**

  
Web scraping is a powerful technique for extracting information from websites. In this tutorial, we will use Python to build a tool that scrapes repositories from a GitHub user, scans files for specific patterns, and displays the results. This project is an excellent way to learn about APIs, regular expressions, and data processing.

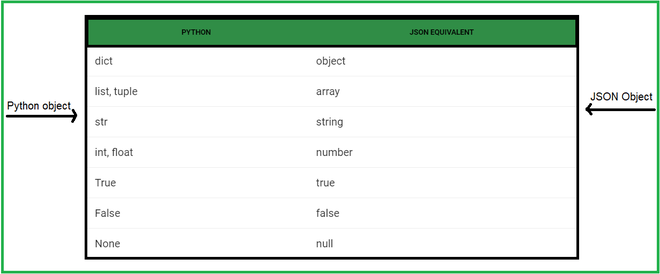
**2. Setting Up the Environment:**First, ensure you have Python installed on your system. You will also need the following libraries:

* requests: For making HTTP requests to the GitHub API
* re: For regular expression operations
* json: For handling JSON data
* time: For tracking execution time
* tqdm: For displaying progress bars (library).

Install the required libraries using pip:

pip install requests tqdm

# Note : what is Json in python ? Python JSON

**Python JSON JavaScript Object Notation** is a format for structuring data. It is mainly used for storing and transferring data between the browser and the server. [Python](https://www.geeksforgeeks.org/python-programming-language/learn-python-tutorial/) too supports JSON with a built-in package called JSON. This package provides all the necessary tools for working with JSON Objects including parsing, [serializing](https://www.geeksforgeeks.org/serializing-json-data-in-python/)(Serialization is the process of encoding the from naive data type to JSON format), deserializing(vise versa), and many more.   


**Let’s take a look at how we serialize Python data to JSON format with these methods:**

* Dump().
* Dumps().

## json.dump()

[**json.dump()**](https://www.geeksforgeeks.org/json-dump-in-python/) method can be used for writing to JSON file. Write data to a file-like object in json format.

***Syntax:****json.dump(dict, file\_pointer)*

***Parameters:***

* ***dictionary –****name of dictionary which should be converted to JSON object.*
* ***file pointer –****pointer of the file opened in write or append mode.*

**Below is the implementation:**

Converting python object and writing into json file

|  |
| --- |
| # import module  import json    # Data to be written  data = {      "user": {          "name": "satyam kumar",          "age": 21,          "Place": "Patna",          "Blood group": "O+"      }  }    # Serializing json and  # Writing json file  with open( "datafile.json" , "w" ) as write:      json.dump( data , write ) |

**Output:**

https://media.geeksforgeeks.org/wp-content/uploads/20201125190923/Capture1.PNG

*data\_file.json*

## json.dumps()

[**json.dumps()**](https://www.geeksforgeeks.org/json-dumps-in-python/) method can convert a Python object into a JSON string.

***Syntax:****json.dumps(dict)*

***Parameters:***

* ***dictionary –****name of dictionary which should be converted to JSON object.*

**Below is the implementation:**

Converting python object into json string.

|  |
| --- |
| # import module  import json    # Data to be written  data = {      "user": {          "name": "satyam kumar",          "age": 21,          "Place": "Patna",          "Blood group": "O+"      }  }    # Serializing json  res = json.dumps( data )  print( res ) |

**Output:**

# https://media.geeksforgeeks.org/wp-content/uploads/20201125191430/Capture-660x37.PNG

# 3. Fetching GitHub Repositories: We need a function to fetch all repositories of a given GitHub user:

# import requests def get\_repos(username): url = f"https://api.github.com/users/{username}/repos" response = requests.get(url) response.raise\_for\_status() # Raises an HTTPError if the status is 4xx or 5xx repos = response.json() # Proceed to process the data if the request was successful return [repo['name'] for repo in repos]

Note: the response.raise\_for\_status() method is used in the context of the requests library, which is a popular library for making HTTP requests. The raise\_for\_status() method is called on a Response object and is used to raise an HTTPError exception if the HTTP request returned an unsuccessful status code (i.e., a status code that indicates an error, such as 4xx or 5xx).

# 4. Retrieving Repository Files:

# Next, we create a function to get all files from a specific repository:

def get\_files(repo\_name, username):  
 url = f"https://api.github.com/repos/{username}/{repo\_name}/git/trees/main?recursive=1"  
 response = requests.get(url)  
 if response.status\_code == 404:  
 url = f"https://api.github.com/repos/{username}/{repo\_name}/git/trees/master?recursive=1"  
 response = requests.get(url)  
 response.raise\_for\_status()  
 files = response.json()  
 return [file['path'] for file in files['tree'] if file['type'] == 'blob']

# 5. Fetching File Content: Now, let’s write a function to retrieve the content of a specific file:

def get\_file\_content(repo\_name, username, file\_path):  
 url = f"https://raw.githubusercontent.com/{username}/{repo\_name}/main/{file\_path}"  
 response = requests.get(url)  
 if response.status\_code == 404:  
 url = f"https://raw.githubusercontent.com/{username}/{repo\_name}/master/{file\_path}"  
 response = requests.get(url)  
 response.raise\_for\_status()  
 return response.text

# 6. Pattern Matching in File Content:

# We need a function to find specific patterns in the content of files using regular expressions:

# import re def find\_patterns(content, patterns): matches = {} for pattern\_name, pattern in patterns.items(): matches[pattern\_name] = re.findall(pattern, content) return matches

# 7. Main Function to Orchestrate the Scraping:

# import time from tqdm import tqdm def main(username, patterns): start\_time = time.time() print("\033[94m" + "="\*50) print(" " \* 15 + "GitHub Scraper") print("="\*50 + "\033[0m") print("\033[92mDeveloped by: sahwe, smoke-wolf, JDX-50S\033[0m") print("\033[94m" + "="\*50 + "\033[0m") repos = get\_repos(username) all\_matches = {pattern\_name: {} for pattern\_name in patterns.keys()} files\_to\_scan = [] total\_lines\_scanned = 0 for repo in repos: try: files = get\_files(repo, username) files\_to\_scan.extend([(repo, file) for file in files]) except Exception as e: print(f"Failed to retrieve files for repo {repo}: {e}") with tqdm(total=len(files\_to\_scan), desc="Scanning files") as pbar: for repo, file in files\_to\_scan: try: content = get\_file\_content(repo, username, file) total\_lines\_scanned += len(content.split('\n')) matches = find\_patterns(content, patterns) for pattern\_name, match\_list in matches.items(): if match\_list: if repo not in all\_matches[pattern\_name]: all\_matches[pattern\_name][repo] = [] all\_matches[pattern\_name][repo].extend(match\_list) except Exception as e: print(f"Failed to process file {file} in repo {repo}: {e}") pbar.update(1) end\_time = time.time() elapsed\_time = end\_time - start\_time avg\_time\_per\_100\_lines = (elapsed\_time / total\_lines\_scanned) \* 100 if total\_lines\_scanned else 0 print("\033[94m" + "="\*50 + "\033[0m") for pattern\_name, repos in all\_matches.items(): print(f"\033[93mPattern: {pattern\_name}\033[0m") for repo, matches in repos.items(): print(f"\033[96mRepository: {repo}\033[0m") for match in matches: print(f"\033[96m{match}\033[0m") print("\033[94m" + "-"\*50 + "\033[0m") print("\033[94m" + "="\*50 + "\033[0m") print(f"\033[92mTotal time taken: {elapsed\_time:.2f} seconds\033[0m") print(f"\033[92mTotal lines scanned: {total\_lines\_scanned}\033[0m") print(f"\033[92mAverage time per 100 lines: {avg\_time\_per\_100\_lines:.2f} seconds\033[0m") while True: print("\033[91mMenu:\033[0m") print("1. Export data as JSON") print("2. Exit") choice = input("Enter your choice: ") if choice == '1': with open(f"{username}\_matches.json", "w") as f: json.dump(all\_matches, f, indent=4) print(f"Data exported to {username}\_matches.json") elif choice == '2': break else: print("Invalid choice, please try again.") 8. Displaying and Exporting Results

The main function includes options for displaying the matched patterns and exporting them to a JSON file.

# 9. Running the Program

To run the program, the user provides a GitHub username and selects patterns to search for:

if \_\_name\_\_ == "\_\_main\_\_":  
 username = input("Enter the GitHub username: ")  
 patterns = {  
 "URLs": r'(https?://\S+)',  
 "Emails": r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}',  
 "Phone Numbers": r'\+?\d[\d -]{8,}\d',  
 "API Keys": r'(?:api[\_-]?key|api[\_-]?token)[\'\"]?[:=][\'\"]?([a-zA-Z0-9-\_]+)[\'\"]?',  
 "Usernames": r'\buser(?:name)?[\'\"]?[:=][\'\"]?([a-zA-Z0-9-\_]+)[\'\"]?',  
 "Passwords": r'\bpass(?:word)?[\'\"]?[:=][\'\"]?([a-zA-Z0-9!@#$%^&\*()\_+={}[\]:;"\'<>.,?/`~\\-]+)[\'\"]?'  
 }  
 print("Select patterns to search for (comma separated):")  
 for idx, pattern\_name in enumerate(patterns.keys(), 1):  
 print(f"{idx}. {pattern\_name}")  
 selected\_indices = input("Enter your choices: ").split(",")  
 selected\_patterns = {name: pattern for idx, (name, pattern) in enumerate(patterns.items(), 1) if str(idx) in selected\_indices}  
 main(username, selected\_patterns)