Unit 4
REGULAR EXPRESSIONS IN PYTHON

Python RegEx

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

RegEx Module

Python has a built-in package called `re`, which can be used to work with Regular Expressions.

Import the `re` module:

```
import re
```

RegEx in Python

When you have imported the `re` module, you can start using regular expressions:

```
import re

#Check if the string starts with "The" and ends with "Spain":

txt = "The rain in Spain"
x = re.search("^The", txt)

if (x):
  print("YES! We have a match!")
else:
  print("No match")
```

output

YES! We have a match!"

RegEx Functions

The `re` module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

Metacharacters

Metacharacters are characters with a special meaning:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Special Sequences

\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word	r"\Bain" r"ain\B"

\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Sets

A set is a set of characters inside a pair of square brackets `[]` with a special meaning:

Set	Description
[arn]	Returns a match where one of the specified characters (`a`, `r`, or `n`) are present
[a-n]	Returns a match for any lower case character, alphabetically between `a` and `n`
[^arn]	Returns a match for any character EXCEPT `a`, `r`, and `n`
[0123]	Returns a match where any of the specified digits (`0`, `1`, `2`, or `3`) are present
[0-9]	Returns a match for any digit between `0` and `9`
[0-5][0-9]	Returns a match for any two-digit numbers from `00` and `59`
[a-zA-Z]	Returns a match for any character alphabetically between `a` and `z`, lower case OR upper case
[+]	In sets, `+`, `*`, `.`, `|`, `()`, `$`,`{}` has no special meaning, so `[+]` means: return a match for any `+` character in the string

The findall() Function

The `findall()` function returns a list containing all matches.

Example

Print a list of all matches:

```
import re

str = "The rain in Spain"
x = re.findall("ai", str)
print(x)
```

output

['ai', 'ai']

The search() Function

The `search()` function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

Example

Search for the first white-space character in the string:

```
import re

str = "The rain in Spain"
x = re.search("\s", str)

print("The first white-space character is located in position:", x.start())
```

output:

The first white-space character is located in position: 3

If no matches are found, the value `None` is returned:

Example

Make a search that returns no match:

```
import re

str = "The rain in Spain"
x = re.search("Portugal", str)
print(x)
```

output

None

The split() Function

The `split()` function returns a list where the string has been split at each match:

Example

Split at each white-space character:

```
import re

str = "The rain in Spain"
x = re.split("\s", str, 1)
print(x)
```

output

['The', 'rain in Spain']

The sub() Function

The `sub()` function replaces the matches with the text of your choice:

Example

Replace every white-space character with the number 9:

```
import re

str = "The rain in Spain"

x = re.sub("\s", "9", str)

print(x)
```

output:

The9rain9in9Spain

You can control the number of replacements by specifying the `count` parameter:

Example

Replace the first 2 occurrences:

```
import re
```

```
str = "The rain in Spain"

x = re.sub("\s", "9", str, 2)

print(x)
```

output

The9rain9in Spain

Match Object

A Match Object is an object containing information about the search and the result.

Note: If there are no match, the value `None` will be returned, instead of the Match Object.

Example

Do a search that will return a Match Object:

```
import re

str = "The rain in Spain"
x = re.search("ai", str)
print(x) #this will print an object
```

output

<_sre.SRE_Match object; span=(5, 7), match='ai'>

The Match object has properties and methods used to retrieve information about the search, and the result:

`.span()` returns a tuple containing the start-, and end positions of the match.
`.string` returns the string passed into the function
`.group()` returns the part of the string where there was a match

Example

Print the position (start- and end-position) of the first match occurrence.

The regular expression looks for any words that starts with an upper case "S":

```
import re
```

```
str = "The rain in Spain"

x = re.search(r"\bS\w+", str)

print(x.span())
```

output:

(12, 17)

----------programs----------‘

```
import re

str = "The rain in Spain"

#Find all lower case characters alphabetically between "a" and "m":

x = re.findall("[a-m]", str)
print(x)
```

output

['h', 'e', 'a', 'i', 'i', 'a', 'i']

```
import re

str = "That will be 59 dollars"

#Find all digit characters:

x = re.findall("\d", str)
print(x)
```

output

['5', '9']

```
import re

str = "hello world"

#Search for a sequence that starts with "he", followed by two (any) characters, and an "o":

x = re.findall("he..o", str)
print(x)
```

output

['hello']

```
import re

str = "hello world"

#Check if the string starts with 'hello':

x = re.findall("^hello", str)
if (x):
  print("Yes, the string starts with 'hello'")
else:
  print("No match")
```

output

Yes, the string starts with 'hello'

```
import re

str = "hello world"

#Check if the string ends with 'world':

x = re.findall("world$", str)
if (x):
  print("Yes, the string ends with 'world'")
else:
  print("No match")
```

output

Yes, the string ends with 'world'

```
import re

str = "The rain in Spain falls mainly in the plain!"
#Check if the string contains "ai" followed by 0 or more "nl" characters:
x = re.findall("ainl*", str)
print(x)

if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

output

```
['ain', 'ain', 'ainl', 'ain']
Yes, there is at least one match!
```

```
import re

str = "The rain in Spain falls mainly in the plain! alkll"

#Check if the string contains "a" followed by exactly two "l" characters:

x = re.findall("al{2}", str)

print(x)

if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

output

```
['all']
Yes, there is at least one match!
```

```
import re

str = "The rain in Spain falls mainly in the plain!"

#Check if the string contains either "falls" or "stays":
```

```
x = re.findall("falls|stays", str)

print(x)

if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

output

['falls']
Yes, there is at least one match!

```
import re

str = "The rain in Spain"

#Check if the string starts with "The":

x = re.findall("\AThe", str)

print(x)

if (x):
  print("Yes, there is a match!")
else:
  print("No match")
```

output

['The']
Yes, there is a match!

```
import re

str = "The rain in Spain"

#Check if "ain" is present at the beginning of a WORD:

x = re.findall(r"\bain", str)

print(x)
```

```
if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

output

```
[]
No match
```

```
import re

str = "The rain in Spain"
#Check if "ain" is present at the end of a WORD:
x = re.findall(r"ain\b", str)
print(x)

if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

output

```
['ain', 'ain']
Yes, there is at least one match!
```

```
import re

str = "The rain in Spain"
#Check if the string has any a, r, or n characters:
x = re.findall("[arn]", str)
print(x)

if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

output

```
['r', 'a', 'n', 'n', 'a', 'n']
Yes, there is at least one match!
```

```
import re
str = "8 times before 11:45 AM"
#Check if the string has any characters from a to z lower case, and A to Z upper case:
x = re.findall("[a-zA-Z]", str)
print(x)

if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

output

['t', 'i', 'm', 'e', 's', 'b', 'e', 'f', 'o', 'r', 'e', 'A', 'M']
Yes, there is at least one match!

Extra:

Prog: A python program to create a regular expression to retrieve all words starting with a numeric digit.

```
import re
str="The meeting will be conducted on 1st and 21st of every month"
result=re.findall(r"\d[\w]*",str)
print(result)
```

output
[1st, 21st]

prog: A Python program to create a regular expression to retrieve all words having 5 characters length (Using findall)

```
import re
str="one two three four five six seven 8 9 10"
result=re.findall(r"\b\w{5}\b",str)
print(result)
```

output
['three', 'seven']

Or (Using search)

```
import re
str="one two three four five six seven 8 9 10"
result=re.search(r"\b\w{5}\b",str)
print(result.group())
```

prog: A Python program to create a regular expression to retrieve all the words that are having the length of at least 4 characters.

```
import re
str="one two three four five six seven 8 9 10"
result=re.findall(r"\b\w{4,}\b",str)
print(result)
```

output

['three', 'four', 'five', 'seven']

Or

#if retrieve 3 or 4 or 5 character {3,5}

#if retrieve only digit r'\d'

Prog: A python program to create a regular expression to retrieve the phone number of a person.

```
import re
str="vipul baldha: 9664650498"
result=re.search(r"\d+",str)
print(result.group())
```

output:

9664650498

Prog: A Python program to create a regular expression to extract only name but not number from a string.

#re.search(r"\D+",str)

Prog: A Python program to create a regular expression to find all words starting with 'an' or 'ak'

```
import re
str="anil akhil anant arun arati arundhati abhijit ankur akruti"
result=re.findall(r"a[nk][\w]*",str)
print(result)
```

output:

['anil', 'akhil', 'anant', 'ankur','akruti']

Prog: Retrieve Date of Birth

```
import re
str="Vipul 33 03-10-1985 Rakesh 30 14-05-1987"
result=re.findall(r"\d{2}-\d{2}-\d{4}",str)
print(result)
```

Prog: A Python program to create a regular expression to search at the ending of a string by ignoring the case.

```
import re
str="Hello World"
res=re.search(r"world$",str,re.IGNORECASE)
if res:
   print("String ends with 'world'")
else:
   print("String does not end with 'world'")
```

Prog: A Python program to create a regular expression to retrieve marks and names from a given string.

```
import re
str="Rahul got 95 marks, Vijay got 55 marks, wheres Vikas got 98 marks."

#extract only marks having 2 digits
marks=re.findall("\d{2}",str)
print(marks)

#extract names starting with a capital letter and remaining alphabets
names=re.findall("[A-Z][a-z]*",str)
print(names)
```

output

```
['95', '55', '98']
['Rahul', 'Vijay', 'Vikas']
```

Using Regular Expression on Files: (IMP)

Prog: A Python program to create a regular expression that reads email-ids from a text file.

```
import re
#open the file for reading
f=open("mails.txt","r")

#repeat for each line of the file
for line in f:
   res=re.findall(r"\S+@\S+",line)

#display if there are some elements in result
if len(res)>0:
   print(res)
```

```
#close the file
f.close()
```

Prog: A Python program to retrieve data from a file using regular expressions and then write that data Into a file.

```
import re
#open the files
f1=open("salaries.txt","r")
f2=open("newfile.txt","w")

for line in f1:
   res1=re.search(r"\d{4}",line) #extract id no from f1
   res2=re.search(r"\d{4,}.\d{2}",line) # extract salary from f1
   print(res1.group(),res2.group()) #display
   f2.write(res1.group()+"\t")
   f2.write(res2.group()+"\n")

f1.close()
f2.close()
```

Retrieving Information from a HTML File (IMP)

Prog: A Python program to retrieve information from a HTML file using a regular expression.

```
import urllib.request
import re

url = 'file:///D:/python_prog/breakfast.html'

req = urllib.request.Request(url)
resp = urllib.request.urlopen(req)
respData = resp.read()

paragraphs = re.findall(r'<td>(.*?)</td>',str(respData))

for eachP in paragraphs:
   print(eachP)
```

Prog: Extract the user id, domain name and suffix from the following email addresses.

```
import re
#open the file for reading
emails = """zuck26@facebook.com
page33@google.com
jeff42@amazon.com"""

# Solution
pattern=r'(\w+)@([A-Z0-9]+)\.([A-Z]{2,4})'
result=re.findall(pattern, emails, flags=re.IGNORECASE)
print(result)
```

Python JSON (JavaScript Object Notation)

JSON stands for JavaScript Object Notation

JSON is a lightweight format for storing and transporting data

JSON is often used when data is sent from a server to a web page

JSON is "self-describing" and easy to understand

JSON is syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

Uses of JSON

- It is used while writing JavaScript based applications that include browser extensions and websites.
- JSON format is used for serializing and transmitting **structured** data over network connection.
- It is primarily used to transmit data between a server and web applications.
- JSON is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute-value pairs and array data types (or any other serializable value).

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects

- Square brackets hold arrays

JSON in Python

Python has a built-in package called json, which can be use to work with JSON data.

Example

Import the json module:

```
import json
```

Method	Description
dumps()	encoding to JSON objects
dump()	encoded string writing on file
loads()	Decode the JSON string
load()	Decode while JSON file read

Parse JSON - Convert from JSON to Python

If you have a JSON string, you can parse it by using the json.loads() method.

```
import json

# some JSON:
x='{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
print(y["name"])
print(y["age"])
print(y["city"])
```

Example

Convert Python objects into JSON strings, and print the values: (**dumps** and loads convert between strings and objects)

```
import json
print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
```

```
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print(json.dumps(42))
print(json.dumps(31.76))
print(json.dumps(True))
print(json.dumps(False))
print(json.dumps(None))
```

Python	JSON
dict	object
list, tuple	array
str	string
int, long, float	number
True	true
False	false
None	null

PYTHON AND XML:

Python is an ideal language for manipulating XML. The combination of Python and XML brings great power to the developer. While XML is a potent technology, it requires the programmer to use objects, interfaces, and strings. Python does so as well, and therefore provides an excellent playpen for XML development. The number of XML tools for Python is growing all the time. XML is used as a **serialization format** for RDFs (RDF/XML) in a semantic web context. In the publishing industry, XML is used throughout the **document processing work flow**. It is also the standard for Office file formats such as Word, Excel, PowerPoint or the Google Docs equivalents.

Key Advantages of XML

1. Application Neutrality
2. Hierarchical Structure
3. International Language Support
4. Platform Neutrality

The Power of Python and XML

Now that we've introduced you to the world of XML, we'll look at what Python brings to the table. We'll review the Python features that apply to XML, and then we'll give some specific examples of Python

with XML. As a very high-level language, Python includes many powerful data structures as part of the core language and libraries. The more recent versions of Python, from 2.0 onward, include excellent support for Unicode and an impressive range of encodings, as well as an excellent (and fast!) XML parser that provides character data from XML as Unicode strings. Python's standard library also contains implementations of the industry-standard DOM.

Python Tools for XML

Three major packages provide Python tools for working with XML. These are, from the most commonly used to the largest: The Python standard library

PyXML, 4Suite.

xml.dom.* – a standard DOM API

xml.sax. – a standard SAX API (Simple API for XML)*

import xml.sax.handler

import xml.dom.minidom

XML vs. JSON – a first glimpse

```
<?xml version="1.0"?>
<book id="123">
<title>Object Thinking</title>
<author>David West</author>
<published>
<by>Microsoft Press</by>
<year>2004</year>
</published>
</book>
```

```
{
"id": 123,
"title": "Object Thinking",
"author": "David West",
"published": {
"by": "Microsoft Press",
"year": 2004
}
}
```

Screen scraper in python

The action of using a computer program to copy data from a website. Screen scraping is the process of collecting screen display data from one application and translating it so that another application can display it. This is normally done to capture data from a legacy application in order to display it using a more modern user interface.

Screen scraping usually refers to a legitimate technique used to translate screen data from one application to another. It is sometimes confused with content scraping, which is the use of manual or automatic means to harvest content from a website without the approval of the website owner.

Screen scraping is sometimes referred to as terminal emulation.

What is Beautiful Soup?

Beautiful Soup is a Python library for pulling data out of HTML and XML files.

```
Cmd-> pip install beautifulsoup4
```

Once you have raw HTML in front of you, you can start to select and extract. For this purpose, you will be using `BeautifulSoup`. The `BeautifulSoup` constructor parses raw HTML strings and produces an object that mirrors the HTML document's structure. The object includes a slew of methods to select, view, and manipulate DOM nodes and text content.

Consider the following quick and contrived example of an HTML document:

```
<!DOCTYPE html>
<html>
<head>
  <title>Contrived Example</title>
</head>
<body>
<p id="hardworker"> I am the hardworker </p>
<p id="simple"> I am the simple man </p>
</body>
</html>
```

If the above HTML is saved in the file `contrived.html`, then you can use `BeautifulSoup` like this:

```
from bs4 import BeautifulSoup
raw_html = open('contrived.html').read()
html = BeautifulSoup(raw_html, 'html.parser')
for p in html.select('p'):
    if p['id'] == 'simple':
        print(p.text)
```

Using BeautifulSoup to Get Mathematician Names (Screen Scraping)

Open **http://www.fabpedigree.com/james/mathmen.html**

Save it in your program folder with magician.html.

If you spend a minute or two looking at this page's source, you can see that each mathematician's name appears inside the text content of an <li> tag. To make matters even simpler, <li> tags on this page seem to contain nothing but names of mathematicians.

```
from bs4 import BeautifulSoup
raw_html=open('magician.html').read()
html=BeautifulSoup(raw_html, 'html.parser')
for i, li in enumerate(html.select('li')):
    print(i, li.text)
```

Mailmerge in Python:

- Python Program to Merge Mails

  To understand this example, you should have the knowledge of following Python programming topics:

  - Python String Methods
  - Python File I/O

  When we want to send the same invitations to many people, the body of the mail does not change. Only the name (and maybe address) needs to be changed.

  Mail merge is a process of doing this. Instead of writing each mail separately, we have a template for body of the mail and a list of names that we merge together to form all the mails.

Source Code to Merge Mails

```
# Python program to mail merger
# Names are in the file names.txt
# Body of the mail is in body.txt

# open names.txt for reading
with open("names.txt",'r',encoding = 'utf-8') as names_file:

   # open body.txt for reading
   with open("body.txt",'r',encoding = 'utf-8') as body_file:

       # read entire content of the body
       body = body_file.read()

       # iterate over names
       for name in names_file:
           mail = "Hello "+name+body

           # write the mails to individual files
           with open(name.strip()+".txt",'w',encoding = 'utf-8') as
mail_file:
               mail_file.write(mail)
```