Python Programming

RGM College of Engineering & Technology (Autonomous)

Department of Computer Science & Engineering

Academic Year : 2020-2021

REGULAR EXPRESSIONS - 1

UNIT – V:

Regular Expressions: Character matching in regular expressions, Extracting data using regular expressions, Combining searching and extracting, Escape character.

Topics Covered:

1. Introduction
2. ‘re’ module
3. Character classes
4. Pre defined Character classes
5. Quantifiers
6. Important functions of 're' module
7. Example applications using Regular expressions

Guido Van Rossum

<u>Learning Mantra</u>

If you really strong in the basics, then remaining things will become so easy.

Agenda

1.Introduction

2. ‘re’ Module

1.INTRODUCTION

1. Introduction

- So far we have been reading through files, looking for patterns and extracting various bits of lines that we find interesting.
- We have been using string methods like split() and find() and using lists and string slicing to extract portions of the lines.
- This task of searching and extracting is so common that Python has a very powerful library called **regular expressions** that handles many of these tasks quite elegantly.
- So far, we have not introduced regular expressions, because they are very powerful and also they are little bit complicated.

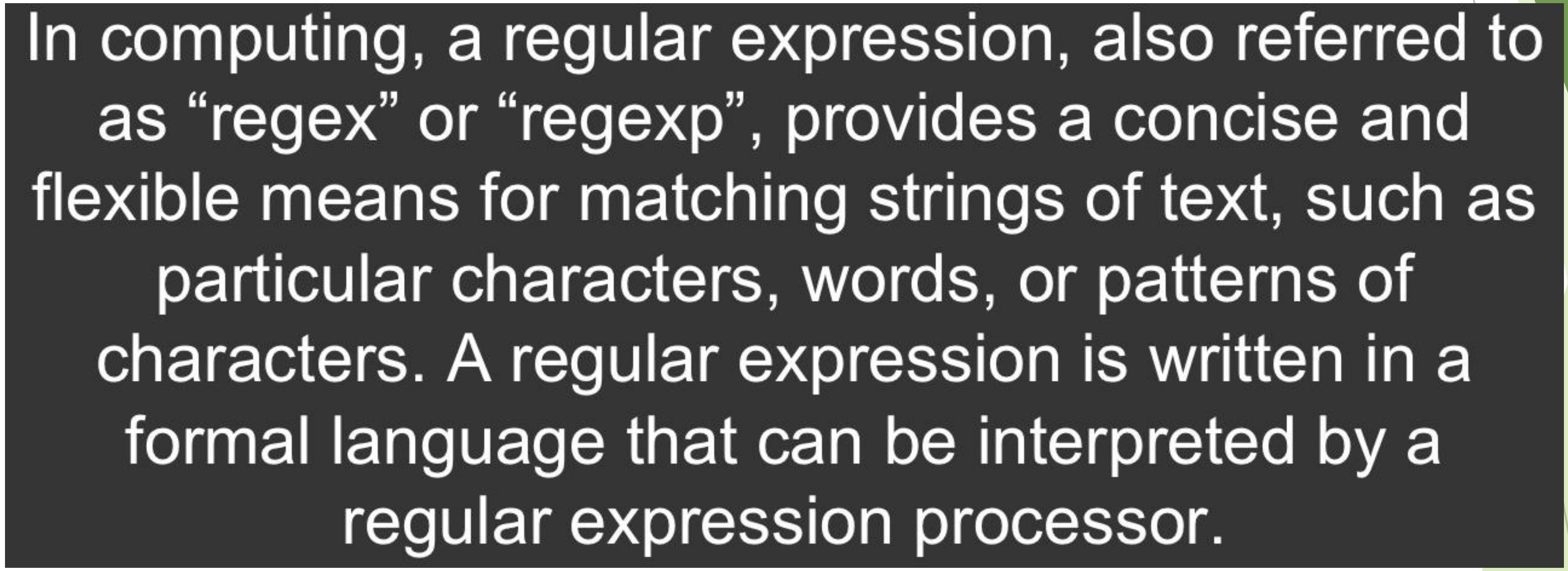
- Regular expressions are almost their own little programming language for searching and parsing strings.
- As a matter of fact, lot of books have been written on the topic of regular expressions.
- In this chapter, we will only cover the basics of regular expressions.

For more detail on regular expressions, see:

https://en.wikipedia.org/wiki/Regular_expression

https://docs.python.org/library/re.html

Regular expressions



In computing, a regular expression, also referred to as “regex” or “regexp”, provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor.

- If we want to represent a group of Strings according to a particular format/pattern then we should go for Regular Expressions. i.e., Regular Expressions is a declarative mechanism to represent a group of Strings according to particular format/pattern.

Eg 1:

- We can write a regular expression to represent all mobile numbers. (i.e., All mobile numbers having a particular format i.e., exactly 10 numbers only)

Eg 2:

- We can write a regular expression to represent all mail ids.

Eg 3:

- We can write a regular expression to represent all java/python/C identifiers.

Note: Regular Expressions is language independent concept.

The main important application areas of Regular Expressions are as follows:

- To develop Validation frameworks/Validation logic. For example, mail id validation, mobile number validation etc.
- To develop Pattern matching applications ('ctrl+f' in windows, 'grep' in UNIX etc.,).
- To develop Translators like compilers, interpreters etc. In compiler design, Lexical analysis phase is internally implemented using Regular expressions only.
- To develop digital circuits. For example, Binary Incrementor, Binary adder, Binary subtractor etc.
- To develop communication protocols like TCP/IP, UDP etc. (Protocol means set of rules, to follow the rules during communication, we use regular expressions).

2. ‘re’ module

- We can develop Regular Expression Based applications by using python module known as **re**.
- This module contains several in-built functions to use Regular Expressions very easily in our applications.

1. compile():

- **re** module contains compile() function to compile a pattern into RegexObject.

```
import re
pattern = re.compile("python")
print(type(pattern))
```

```
<class 're.Pattern'>
```

2. finditer():

- Returns an Iterator object which yields Match object for every Match.

matcher = pattern.finditer("Learning python is very easy...")

On Match object we can call the following methods.

1. start() ➔Returns start index of the match
2. end() ➔ Returns end+1 index of the match
3. group() ➔ Returns the matched string

Eg: Write a python program to find whether the given pattern is available in the given string or not?

```
import re
count=0
pattern=re.compile("python")
matcher=pattern.finditer("Learning python is very easy...") # We are searching for a word i
for match in matcher:
    count+=1
    print(match.start(),"...",match.end(),"...",match.group())
print("The number of occurrences: ",count)
```

Output:

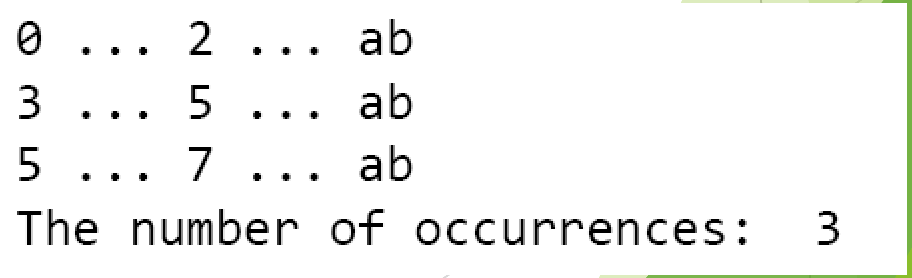
```
9 ... 15 ... python
The number of occurrences: 1
```

Note: More Simplified form:

- We can pass pattern directly as argument to finditer() function.

Eg:

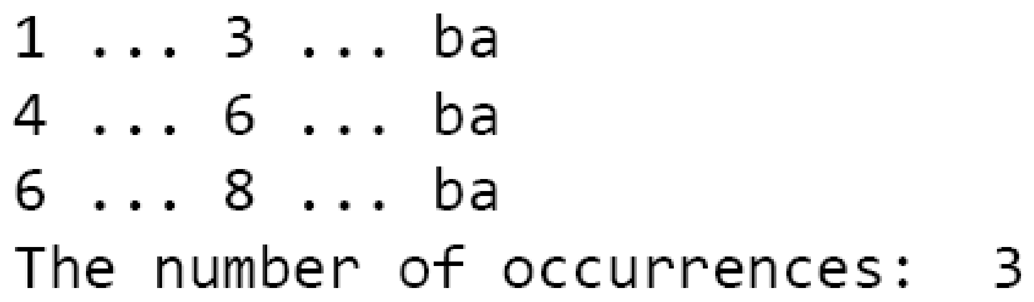
```
import re
count=0
matcher=re.finditer("ab","abaababa")
for match in matcher:
    count+=1
    print(match.start(),"...",match.end(),"...",match.group())
print("The number of occurrences: ",count)
```



```
0 ... 2 ... ab
3 ... 5 ... ab
5 ... 7 ... ab
The number of occurrences:  3
```

Eg:

```
import re
count=0
matcher=re.finditer("ba","abaababa")
for match in matcher:
    count+=1
    print(match.start(),"...",match.end(),"...",match.group())
print("The number of occurrences: ",count)
```



```
1 ... 3 ... ba
4 ... 6 ... ba
6 ... 8 ... ba
The number of occurrences:  3
```

Eg:

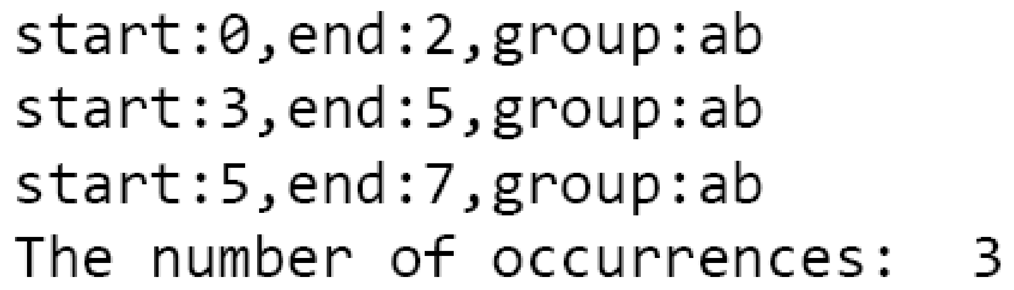
```
import re
count=0
matcher=re.finditer("bb","abaababa")
for match in matcher:
    count+=1
    print("start:{},end:{},group:{}".format(match.start(),match.end(),match.group()))
print("The number of occurrences: ",count)
```

Output:

```
The number of occurrences: 0
```

Eg:

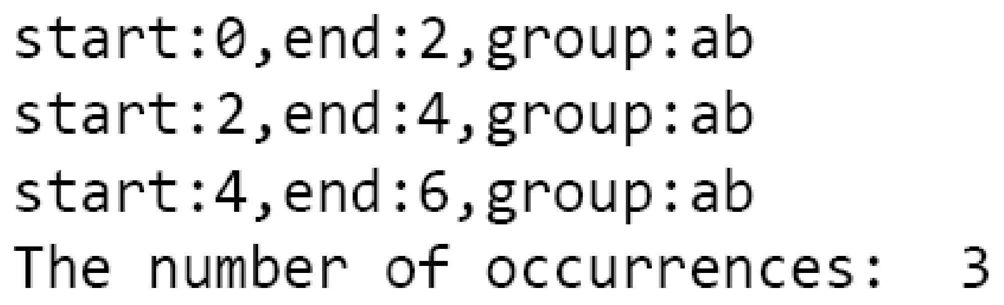
```
import re
count=0
matcher=re.finditer("ab","abaababa")
for match in matcher:
    count+=1
    print("start:{},end:{},group:{}".format(match.start(),match.end(),match.group()))
print("The number of occurrences: ",count)
```



```
start:0,end:2,group:ab
start:3,end:5,group:ab
start:5,end:7,group:ab
The number of occurrences:  3
```

Eg:

```
import re
count=0
matcher=re.finditer("ab","abababa")
for match in matcher:
    count+=1
    print("start:{},end:{},group:{}".format(match.start(),match.end(),match.group()))
print("The number of occurrences: ",count)
```



```
start:0,end:2,group:ab
start:2,end:4,group:ab
start:4,end:6,group:ab
The number of occurrences:  3
```

Any question?

If you try to practice programs yourself, then you will learn many things automatically

Spend few minutes and then enjoy the study

Thank You