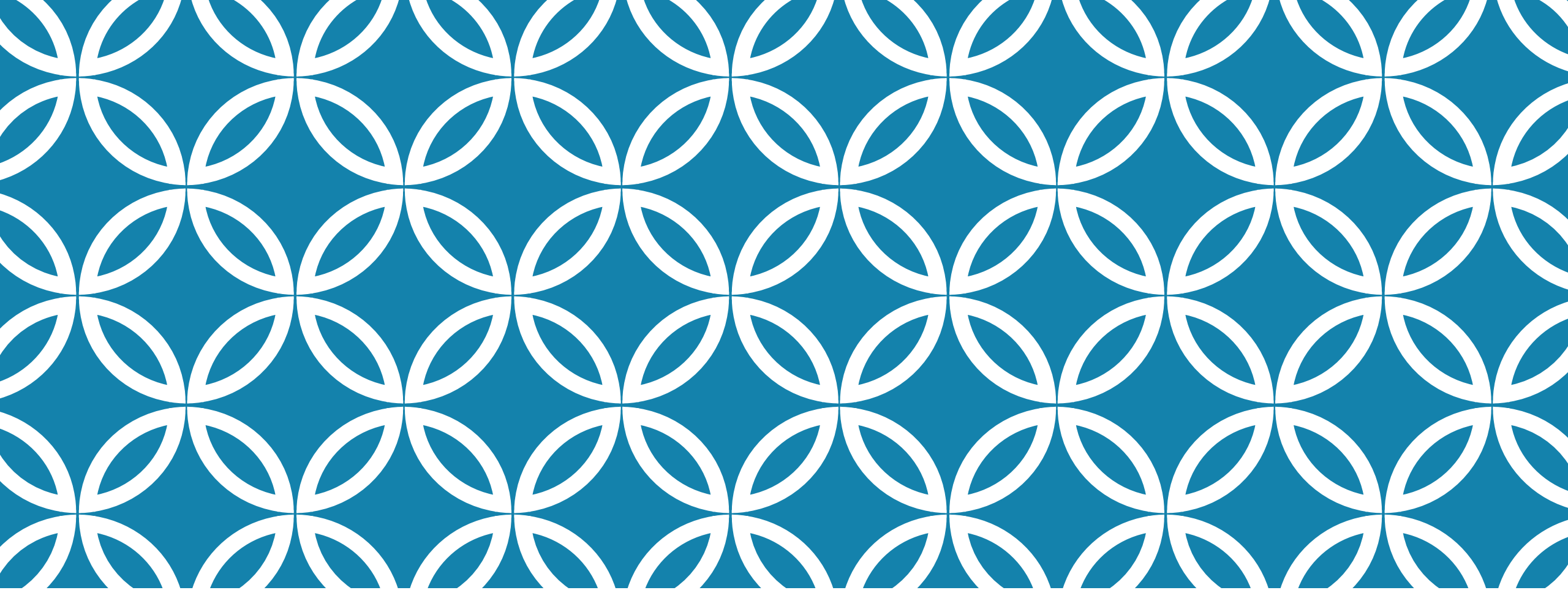


PROGRAMMING IN PYTHON

Gavrilut Dragos

Course 8

REGULAR EXPRESSIONS SUPPORT

Regular expression are implemented in Python in library “re”.

Usual usage:

- Regular expression (string form) is compiled into a binary form (usually an automata)
- The binary form is used for the following:
  - Checks if a string matches a regular expression
  - Checks if a sub-string from a string can be identified using a regular expression
  - Replace substrings from a string based on a regular expression

Documentation:

- Python 3: https://docs.python.org/3/library/re.html

REGULAR EXPRESSIONS SUPPORT

Regular expression special characters (here is a simple set of special characters)

Character	Match	Character	Match
.	All characters except new line	\d	Decimal characters 0,1,2,3,…9
^	Matches at the start of the string	\D	All except decimal characters
$	Matches at the end of the string	\s	Space, tab, new line (CR/LF) characters
*	>=0 repetition(s)	\S	All except characters designated by \s
?	0 or 1 occurrence	\w	Word characters a-z, A-Z, 0-9 and _
+	>=1 repetition(s)	\W	All except characters designated by \w
{x}	Matches <x> times	\	Escape character
{x,y}	Matches between <x> and <y> times	[^…]	Not specified group of characters
[]	Group of characters	(…)	Grouping
|	Or condition	[..-..]	'-' interval for a group of characters.

REGULAR EXPRESSIONS SUPPORT

Usage:

- use re.compile (regular_expression_string,flags) to compile a regular expression into its binary form
- Use the "match" method of the resulted object to check if a string matches the regular expression

Python 3.x

```
import re

r = re.compile("07[0-9]{8}")
if r.match("0740123456"):
        print("Match")
```

Output

```
Match
```

- The same result can be achieved by using the "match" function from the re module directly

Python 3.x

```
import re
if re.match("07[0-9]{8}", "0740123456"):
        print("Match")
```

REGULAR EXPRESSIONS SUPPORT

Pattern	String that will be match
\w+\s+\w+	“Gavrilut Dragos”, “Gavrilut Dragos Teodor”
^\w+\s+\w+$	“Gavrilut Dragos”
[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}	“192.168.0.1”, “999.999.999.999”
([0-9]{1,3}\.){3}[0-9]{1,3}	“192.168.0.1”, “999.999.999.999”
^((([0-9])|([1-9][0-9])|(1[0-9]{2})|(2[0-4][0-9])|(25[0-5]))\.){3}(([0-9])|([1-9][0-9])|(1[0-9]{2})|(2[0-4][0-9])|(25[0-5]))$	Will only match IP addresses
[12]\d{12}	CNP (but will not validate the correctness of the birth date
0x[0-9a-fA-F]+	A hex number
(if|then|else|while|continue|break)	A special keyword

REGULAR EXPRESSIONS SUPPORT

re.match starts the matching from the beginning of the string and stops once the matching ends and not when the string ends except for the case where regular expression pattern is using the "$" character:

Python 3.x

```
import re

if re.match("\d+","123 USD"):
        print ("Match")

if re.match("\d+","Price is 123 USD"):
        print ("Match")

if not re.match("\d+$","123 USD"):
        print ("NO Match")
```

Output

```
Match
NO Match
```

REGULAR EXPRESSIONS SUPPORT

If you want to check if a regular expression pattern is matching a part of a string, the "**search**" method can be used:

Python 3.x

```
import re

if re.search("\d+","Price is 123 USD"):
        print ("Found")
```

Output

```
Found
```

The same can be achieved using a compiled object:

Python 3.x

```
import re
r = re.compile("\d+")
if r.search("Price is 123 USD"):
        print ("Found")
```

REGULAR EXPRESSIONS SUPPORT

search method stops after the first match is achieved.

The object returned by the **search** or **match** method is called a match object. A match object is always evaluated to **true**. If the search does not find any match, None is returned and will be evaluated to false. A match object has several members:

- **group**(index) → returns the substring that matches that specific group. If *index* is 0, the substring refers to the entire string that was matched by the regular expression
- **lastindex** → returns the index of the last object that was matched by the regular expression. To create a group within the regular expression, one must use (…).

Python 3.x

```
import re

result = re.search("\d+","Price is 123 USD")
if result:
        print (result.group(0))
```

Output

```
123
```

REGULAR EXPRESSIONS SUPPORT

In case of some operators (like * or **+**) they can be preceded by **?**. This will specify a NON-greedy behavior.

Python 3.x

```
import re

result = re.search(".*(\d+)",  "File size if 12345 bytes")
if result:
        print (result.group(1))

result = re.search(".*?(\d+)", "File size if 12345 bytes")
if result:
        print (result.group(1))
```

Output

```
5
12345
```

REGULAR EXPRESSIONS SUPPORT

In case of some operators (like * or +) they can be preceded by **?**. This will specify a NON-greedy behavior.

Python 3.x

```
import re

result = re.search(".*(\d+)",  "File size if 12345 bytes")
if result:
        print (result.group(1))

result = re.search(".*?(\d+)", "File size if 12345 bytes")
if result:
        print (result.group(1))
```

Output

```
5
12345
```

REGULAR EXPRESSIONS SUPPORT

(...) is usually used to delimit specific sequences of sub-string within the regular expression pattern:

Python 3.x

```
import re

result = re.search("(\d+)[^\d]*(\d+)","Price is 123 USD aprox 110 EUR")
if result:
        print (result.lastindex)
        for i in range(0,result.lastindex+1):
                print (i, "=>", result.group(i))
```

Output

```
2
0 => 123 USD, aprox. 110
1 => 123
2 => 110
```

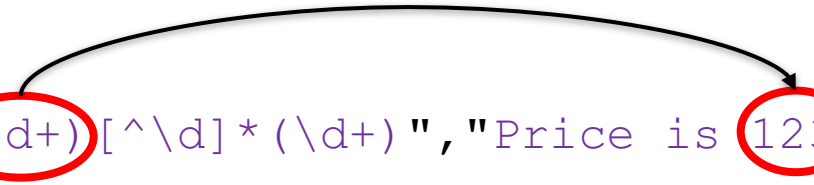
REGULAR EXPRESSIONS SUPPORT

(...) is usually used to delimit specific sequences of sub-string within the regular expression pattern:

Python 3.x

```
import re

result = re.search("(\d+)[^\d]*(\d+)","Price is 123 USD aprox 110 EUR")
if result:
        print (result.lastindex)
        for i in range(0,result.lastindex+1):
                print (i, "=>", result.group(i))
```



Output

```
2
0 => 123 USD, aprox. 110
1 => 123
2 => 110
```

REGULAR EXPRESSIONS SUPPORT

(...) is usually used to delimit specific sequences of sub-string within the regular expression pattern:

Python 3.x

```
import re

result = re.search("(\d+)[^\d]*(\d+)","Price is 123 USD aprox 110 EUR")
if result:
        print (result.lastindex)
        for i in range(0,result.lastindex+1):
                print (i, "=>", result.group(i))
```

Output

```
2
0 => 123 USD, aprox. 110
1 => 123
2 => 110
```

REGULAR EXPRESSIONS SUPPORT

(...) is usually used to delimit specific sequences of sub-string within the regular expression pattern:

Python 3.x

```
import re

result = re.search("((\d+),(\d+))[^\d]*(\d+)",
                   "Color from pixel 20,30 is 123")
if result:
        print (result.lastindex)
        for i in range(0,result.lastindex+1):
                print (i, "=>", result.group(i))
```

Output

```
4
0 => 20,30 is 123
1 => 20,30
2 => 20
3 => 30
4 => 123
```

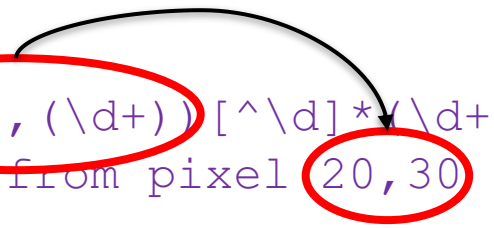
REGULAR EXPRESSIONS SUPPORT

(...) is usually used to delimit specific sequences of sub-string within the regular expression pattern:

Python 3.x

```
import re

result = re.search("((\d+),(\d+))[^\d]*(\d+)",
                   "Color from pixel 20,30 is 123")
if result:
        print (result.lastindex)
        for i in range(0,result.lastindex+1):
                print (i, "=>", result.group(i))
```



Output

```
4
0 => 20,30 is 123
1 => 20,30
2 => 20
3 => 30
4 => 123
```

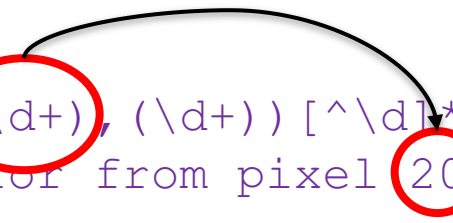
REGULAR EXPRESSIONS SUPPORT

(...) is usually used to delimit specific sequences of sub-string within the regular expression pattern:

Python 3.x

```
import re

result = re.search("((\d+),(\d+))[^\d]*(\d+)",
                   "Color from pixel 20 30 is 123")
if result:
        print (result.lastindex)
        for i in range(0,result.lastindex+1):
                print (i, "=>", result.group(i))
```



Output

```
4
0 => 20,30 is 123
1 => 20,30
2 => 20
3 => 30
4 => 123
```

REGULAR EXPRESSIONS SUPPORT

(...) is usually used to delimit specific sequences of sub-string within the regular expression pattern:

Python 3.x

```
import re

result = re.search("((\d+),(\d+))[^\d]*(\d+)",
                   "Color from pixel 20,30 is 123")
if result:
        print (result.lastindex)
        for i in range(0,result.lastindex+1):
                print (i, "=>", result.group(i))
```

Output

```
4
0 => 20,30 is 123
1 => 20,30
2 => 20
3 => 30
4 => 123
```

REGULAR EXPRESSIONS SUPPORT

search stops after the first match. To find all substring that match a specific regular expression from a string, use **findall** method.

Python 3.x

```
import re

result = re.findall("\d+","Color from pixel 20,30 is 123")
if result:
        print (result)
```

Output

['20', '30', '123']

The result is a vector containing all substrings that matched the regular expression.

REGULAR EXPRESSIONS SUPPORT

search stops after the first match. To find all substring that match a specific regular expression from a string, use **findall** method.

Using groups (...) is also allowed (in this case they will be converted to a tuple in each list element.

Python 3.x

```
import re

result = re.findall("(\d)(\d+)","Color from pixel 20,30 is 123")
if result:
        print (result)
```

Output

```
[('2', '0'), ('3', '0'), ('1', '23')]
```

REGULAR EXPRESSIONS SUPPORT

search stops after the first match. To find all substring that match a specific regular expression from a string, use **findall** method.

Using groups (...) is also allowed (in this case they will be converted to a tuple in each list element.

Python 3.x

```
import re

result = re.findall("(\d)(\d+)","Color from pixel 20,30 is 123")
if result:
        print (result)
```

Output

```
[('2', '0'), ('3', '0'), ('1', '23')]
```

REGULAR EXPRESSIONS SUPPORT

search stops after the first match. To find all substring that match a specific regular expression from a string, use **findall** method.

Using groups (...) is also allowed (in this case they will be converted to a tuple in each list element.

Python 3.x

```
import re

result = re.findall("(\d)(\d+)","Color from pixel 20,30 is 123")
if result:
        print (result)
```

Output

```
[('2', '0'), ('3', '0'), ('1', '23')]
```

REGULAR EXPRESSIONS SUPPORT

split method can be used to split a string using a regular expression.

The result is a vector with all elements that substrings that were obtained after the split occurred.

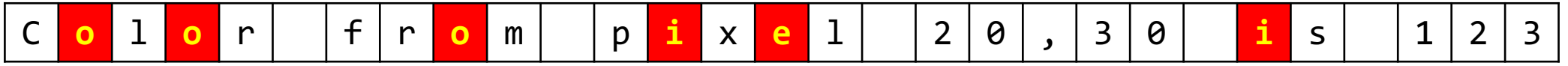
Python 3.x

```
import re

result = re.split("[aeiou]+","Color from pixel 20,30 is 123")
print (result)
```

Output

['C', 'l', 'r fr', 'm p', 'x', 'l 20,30 ', 's 123']



REGULAR EXPRESSIONS SUPPORT

Groups can also be used. In this case the split is done after each group that matches.

Python 3.x

```
import re
print (re.split("\d\d","Color from pixel 20,30 is 123"))
```

Elements

'Color from pixel '	','	' is '	'3'

Python 3.x

```
import re
print (re.split("(\d)(\d)","Color from pixel 20,30 is 123"))
```

Elements

'Color from pixel '	'2'	'0'	','	'3'	'0'	' is '	'1'	'2'	'3'

REGULAR EXPRESSIONS SUPPORT

Groups can also be used. In this case the split is done after each group that matches.

Python 3.x

```
import re
print (re.split("\d\d+","Color from pixel 20,30 is 123"))
```

Output

```
['Color from pixel ', ',', ' is ', '']
```

C	o	l	o	r		f	r	o	m		p	i	x	e	l		2	0	,	3	0		i	s		1	2	3

REGULAR EXPRESSIONS SUPPORT

Groups can also be used. In this case the split is done after each group that matches.

Python 3.x

```
import re
print (re.split("\d\d+", "12345"))
```

Output

```
['', '']
```

Python 3.x

```
import re
print (re.split("\d", "12345"))
```

Output

```
['', '', '', '', '', '']
```

Python 3.x

```
import re
print (re.split("(\d)", "12345"))
```

Output

```
['', '1', '', '2', '', '3', '', '4', '', '5', '']
```

REGULAR EXPRESSIONS SUPPORT

Groups can also be used. In this case the split is done after each group that matches.

"If capturing parentheses are used in pattern, then the text of all groups in the pattern are also returned as part of the resulting list."

https://docs.python.org/3/library/re.html#re.split

Python 3.x

```
import re
print (re.split("(\d\d+)", "Color from pixel 20,30 is 123"))
```

Output

```
['Color from pixel ', '20', ',', '30', ' is ', '123', '']
```

REGULAR EXPRESSIONS SUPPORT

split method also allow flags and to specify how many times a split can be performed.
The full format is *split (pattern, string, maxsplit=0, flags=0)*

Python 3.x

```
import re
s = "Today I'm having a python course"
print (re.split("[^a-z']+", s))
print (re.split("[^a-z']+", s, 2))
print (re.split("[^a-z']+", s, flags = re.IGNORECASE))
print (re.split("[^a-z']+", s, 2, flags = re.IGNORECASE))
print (re.split("[^a-z'A-Z]+", s))
```

Output

```
['', 'oday', "'m", 'having', 'a', 'python', 'course']
['', 'oday', "'m having a python course"]
['Today', "I'm", 'having', 'a', 'python', 'course']
['Today', "I'm", 'having a python course']
['Today', "I'm", 'having', 'a', 'python', 'course']
```

REGULAR EXPRESSIONS SUPPORT

Regexp can also be used to replace a matched string with another string using the method **sub**.
format is *sub (pattern, replace, string, count=0, flags=0)*

- ***pattern*** is a regular expression to search for
- ***replace*** is either a string or a function
- ***string*** is the string where you are going to search the pattern
- ***count*** represents how many time the replacement can occur. If missing or 0 means for all matches.
- ***flags*** represents some flags (like re.IGNORECASE)

Python 3.x

```
import re
s = "Today I'm having a python course"
print (re.sub("having\s+a\s+\w+\s+course", "not doing anything", s))
```

Output

Today I'm **not doing anything**

REGULAR EXPRESSIONS SUPPORT

Regexp can also be used to replace a match with another string using the method **sub.**
format is *sub (pattern, replace, string, count=0, flags=0)*

If ***replace*** parameter is a string, there is a special operator (\<**number**>) that if found within the replacement string will be replace with the group from the match search (for example \3 will be replaced with .group(3)).

Python 3.x

```
import re
s = "Today I'm having a python course"
print (re.sub("having\s+a\s+(\w+)\s+course",
              r"not doing the \1 course",
               s))
```

Output

Today I'm **not doing the python course**

REGULAR EXPRESSIONS SUPPORT

Regexp can also be used to replace a match with another string using the method **sub.** format is *sub (pattern, replace, string, count=0, flags=0)*

If ***replace*** parameter is a string, there is a special operator (**\<number>**) that if found within the replacement string will be replace with the group from the match search (for example \3 will be replaced with .group(3)).

Python 3.x

```
import re
s = "Today I'm having a python course"
print (re.sub("having\s+a\s+(\w+)\s+course",
              r"not doing the \1 course",
               s))
```

You can also use \g<number> with the same effect. In this case **\g<1>**

REGULAR EXPRESSIONS SUPPORT

Regexp can also be used to replace a match with another string using the method **sub**. format is *sub (pattern, replace, string, count=0, flags=0)*

If ***replace*** parameter is a function it receives the match object. Usually that function will use .group(0) method to get the string that was matched and convert it to the replacement value.

Python 3.x

```
import re

def ConvertToHex(s):
        return hex(int(s.group(0)))

s = "File size is 12345 bytes"
print (re.sub("\d+",ConvertToHex, s))
```

Output

File size is 0x3039 bytes

EXTENSIONS

Python regular expressions supports extensions. The form of the extension is **(?...)**

- **(?P<name>...)** will set the name of a group to a given string. In case of a match, that group can be accessed based on its name.

Python 3.x

```
import re
s = "File size if 12345 bytes"
result = re.search("(?P<file_size>\d+)",s)
if result:
        print ("Size is ",result.group("file_size"))
```

Output

```
Size is 12345
```

EXTENSIONS

Python regular expressions supports extensions. The form of the extension is **(?...)**

- **(?P<name>...)** →The match object also has a **groupdict** method that returns a dictionary with all the keys and strings that match the specified regular expression

Python 3.x

```
import re

s = "File config.txt was create on 2016-10-20 and has 12345 bytes"
result = re.search("File\s+(?P<name>[a-z\.]+)\s.*(?P<date>\d{4}-\d{2}-
\d{2}).*\s(?P<size>\d+)",s)
if result:
        print (result.groupdict())
```

Result

```
{
     'date'  : '2016-10-20',
     'name'  : 'config.txt',
     'size'  : '12345‘
}
```

EXTENSIONS

Python regular expressions supports extensions. The form of the extension is **(?...)**

- **(?i)(...)** ignore case will be applied for the current block match
- **(?s)(...)** "." (dot) will match everything

Python 3.x

```
import re
s = "12345abc54321"
result = re.search("(?i)([A-Z]+))",s)
if result:
        print (result.group(1))
```

Output

```
abc
```

EXTENSIONS

Python regular expressions supports extensions. The form of the extension is **(?...)**

- **(?=...)** will match the previous expression only if next expression is … (this is called look ahead assertion)
- **(?!...)** similarly, will match only if the next expression will **not** match …

Python 3.x

```
import re
s = "Python Course"
result = re.search("(Python)\s+(?=Course)",s)
if result:
        print (result.group(1))
```

Output

```
Python
```

EXTENSIONS

Python regular expressions supports extensions. The form of the extension is **(?...)**

- **(?#...)** represents a comment / information that can be added in the regular expression to reflect the purpose of a specific group

Python 3.x

```
import re
s = "Size is 1234 bytes"
result = re.search("(?# file size)(\d+)",s)
if result:
        print ("Size is ",result.group(1))
```

Output

```
Size is 1234
```

BUILDING A TOKENIZER

Python has a way to iterate through a string applying different regular expression. Because of this, a tokenizer can be built for different languages. Use method **finditer** for this.

Python 3.x

```
import re
number = "(?P<number>\d+)"
operation = "(?P<operation>[+\-\*\/])"
braket = "(?P<braket>[\(\)])"
space = "(?P<space>\s)"
other = "(?P<other>.)"
r = re.compile(number+"|"+operation+"|"+braket+
               "|"+space+"|"+other)
expr = "10 * (250+3)"
for matchobj in r.finditer(expr):
        key = matchobj.lastgroup
        print (matchobj.group(key)+" => "+key)
```

Output

```
10 => number
  => space
* => operation
  => space
( => braket
250 => number
+ => operation
3 => number
) => braket
```

REGULAR EXPRESSIONS SUPPORT

Recommendations:

1. If the same regular expression is used multiple times using it in the compile form will improve the performance of the script
2. Even if Python recognizes some escape sequences (such as \d or \w) it is better to either use a raw string r"…" or to duplicate the escape character
   - Instead of "\d" → use r"\d" or "\\d"
3. Regular expression need memory. If all you need is to search a substring within another substring or perform simple string operation, don't use regular expression for this.
4. If you are trying to use the regular expression in a portable way, don't use some features like (?P=name) → other languages or regular expression engines might not support this.