REGULAR EXPRESSIONS

* use string methods for **simple text processing**

* string methods are more readable and simpler than regular expressions

REGULAR EXPRESSION

text pattern that a program uses to find substrings that will match the required pattern

expression that specify a set of strings

a **pattern matching mechanism**

also known as **Regex**

introduced in the 1950s as part of formal language theory

REGULAR EXPRESSIONS

very powerful! hundreds of code could be reduced to a **one-liner** elegant regular expression.

used to construct compilers, interpreters, text editors, …

used to **search** & **match** text patterns

used to **validate** text data formats especially input data

REGULAR EXPRESSIONS

Popular programming languages have **RegEx** capabilities:

Perl, JavaScript, PHP, Python, Ruby, Tcl,
Java, C, C++, C#, .Net, Ruby, …

REGEX

Popular programming languages have **RegEx** capabilities:

Perl, JavaScript, PHP, Python, Ruby, Tcl,
Java, C, C++, C#, .Net, Ruby, …

REGEX | General Concepts

- ❑ Alternative
- ❑ Grouping
- ❑ Quantification
- ❑ Anchors
- ❑ Meta-characters
- ❑ Character Classes

REGEX | General Concepts

❑ Alternative:	|
❑ Grouping:	()
❑ Quantification:	? + * {m,n}
❑ Anchors:	^ $
❑ Meta-characters:	. [ ] [-] [^]
❑ Character Classes:	\w \d \s \W …

REGEX | Alternative

“ranel|ranilio” == “ranel” or “ranilio”

“gray|grey” == “gray” or “grey”

REGEX | Grouping

“ran(el|ilio)” == “ranel” or “ranilio”

“gr(a|e)y” == “gray” or “grey”

“ra(mil|n(ny|el))” == “ramil” or “ranny” or “ranel”

REGEX | Quantification | ?

? == **zero** or **one** of the preceding element

“rani?el” == “raniel” or “ranel”

“colou?r” == “colour” or “color”

REGEX | Quantification | *

* == **zero** or **more** of the preceding element

“goo*gle” == “gogle” or “google” or “goooooogle”

“(ha)*” == “” or “ha” or “haha” or “hahahahaha”

“12*3” == “13” or “1223” or “12223”

REGEX | Quantification | +

+ == **one** or **more** of the preceding element

“goo+gle” == “google” or “gooogle” or “goooooogle”

“(ha)+” == “ha” or “haha” or “hahahahaha”

“12+3” == “123” or “1223” or “12223”

REGEX | Quantification | {m,n}

{m, n} == *m* **to** *n* times of the preceding element

“go{2, 3}gle” == “google” or “gooogle”

“6{3, 6}” == “666” or “6666” or “66666” or “666666”

“5{3}” == “555”

“a{2,}” == “aa” or “aaa” or “aaaa” or “aaaaa” …

REGEX | Anchors | ^

^ == matches the **starting position** within the string

“^laman” == “lamang” or “lamang-loob” or “lamang-lupa”

“^2013” == “2013”, “2013-12345”, “2013/1320”

REGEX | Anchors | $

$ == matches the **ending position** within the string

“laman**$**” == “halaman” or “kaalaman”

“2013**$**” == “2013”,“777-2013”,“0933-445-2013”

REGEX | Meta-characters | .

. == matches any single character

“ala.” == “ala” or “alat” or “alas” or “ala2”

“1.3” == “123” or “143” or “1s3”

REGEX | Meta-characters | [ ]

[ ] == matches a single character that is contained **within** the **brackets**.

“[abc]” == “a” or “b” or “c”

“[aoieu]” == any vowel

“[0123456789]” == any digit

REGEX | Meta-characters | [-]

[-] == matches a single character that is contained **within** the **brackets** and the specified range.

“[a-c]” == “a” or “b” or “c”

“[a-z]” == all alphabet letters (lowercase only)

“[a-zA-Z]” == all letters (lowercase & uppercase)

“[0-9]” == all digits

REGEX | Meta-characters | [^]

[^] == matches a single character that is **not contained** within the brackets.

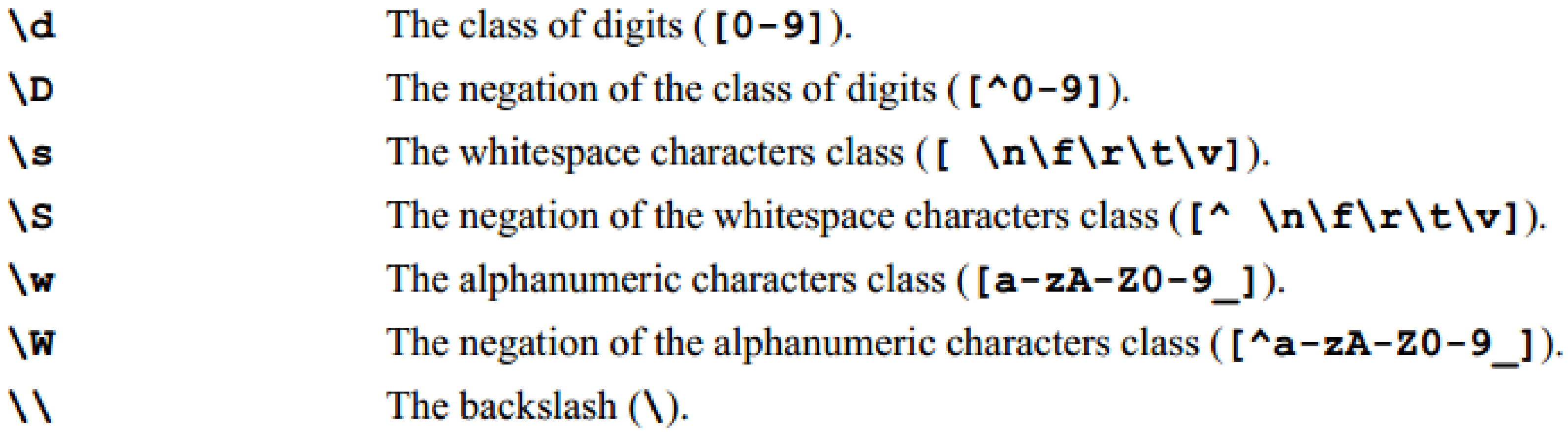
“[^aeiou]” == any non-vowel

“[^0-9]” == any non-digit

“[^abc]” == any character, but not “a”, “b”, or “c”

REGEX | Character Classes

Character classes specifies a **group of characters** to match in a string

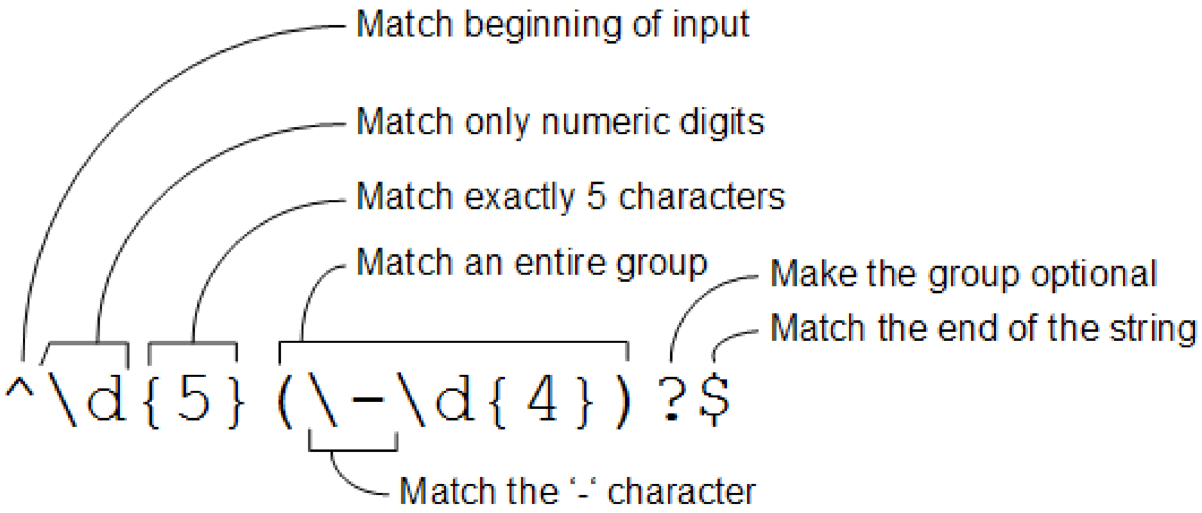


`\d`	The class of digits (`[0-9]`).
`\D`	The negation of the class of digits (`[^0-9]`).
`\s`	The whitespace characters class (`[ \n\f\r\t\v]`).
`\S`	The negation of the whitespace characters class (`[^ \n\f\r\t\v]`).
`\w`	The alphanumeric characters class (`[a-zA-Z0-9_]`).
`\W`	The negation of the alphanumeric characters class (`[^a-zA-Z0-9_]`).
`\\`	The backslash (\\).

REGEX | Summary

❑ Alternative:	|
❑ Grouping:	()
❑ Quantification:	? + * {m,n}
❑ Anchors:	^ $
❑ Meta-characters:	. [ ] [-] [^]
❑ Character Classes:	\w \d \s \W …

REGEX | Combo



```
^\d{5}(\-\d{4})?$
```

Match beginning of input

Match only numeric digits

Match exactly 5 characters

Match an entire group

Make the group optional

Match the end of the string

Match the '-' character

REGEX | Date Validation

“1/3/2013” or “24/2/2020”

```
(\d{1,2}\/\d{1,2}\/\d{4})
```

REGEX | Alphanumeric, -, & _

“rr2000” or “ranel_padon” or “Oblan-Padon”

([a-zA-Z0-9-_]+)

REGEX | Numbers in 1 to 50

“1” or “50” or “14”

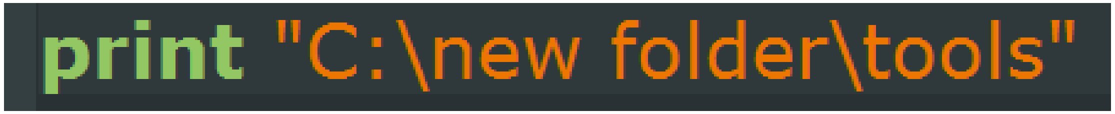
```
(^[1-9]{1}$|^[1-4]{1}[0-9]{1}$|^50$)
```

REGEX | HTML Tags

“<title>” or “<strong>” or “/body”

```
(<(/?[^>]+)>)
```

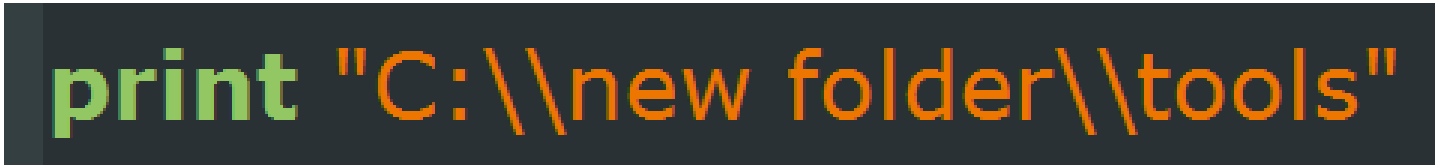
PYTHON REGEX | Raw String



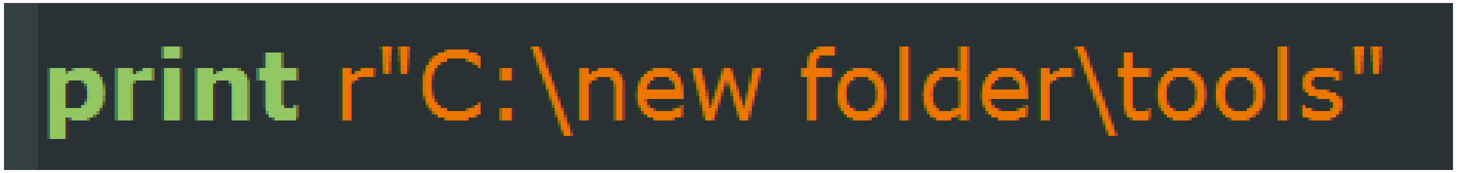
```
print "C:\new folder\tools"
```

PYTHON REGEX | Raw String r

Two Solutions:



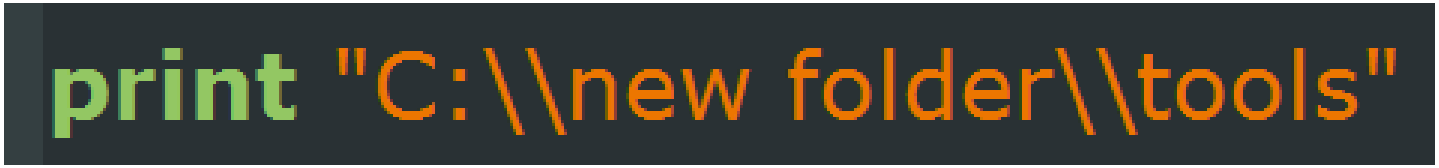
```
print "C:\\new folder\\tools"
```



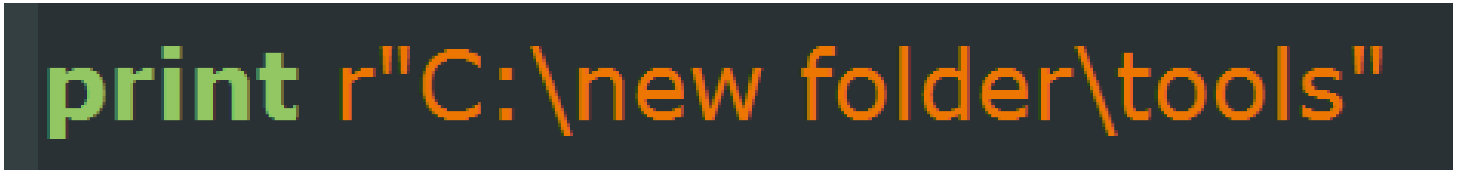
```
print r"C:\new folder\tools"
```

PYTHON REGEX | Raw String r

Raw Strings are used for enhancing readability.

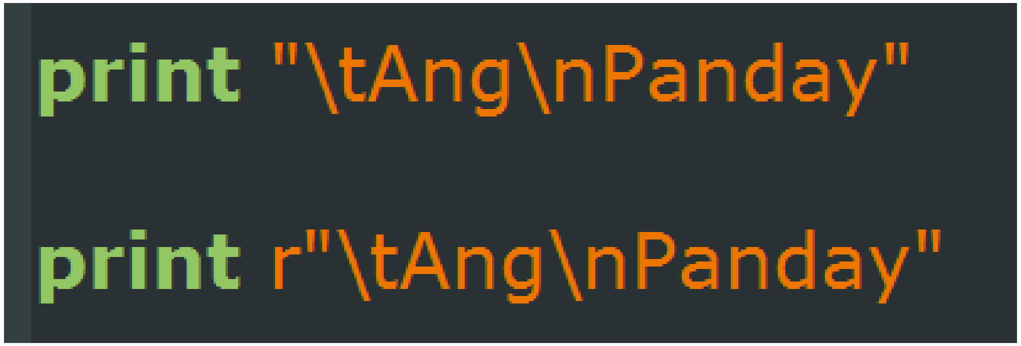


```
print "C:\\new folder\\tools"
```



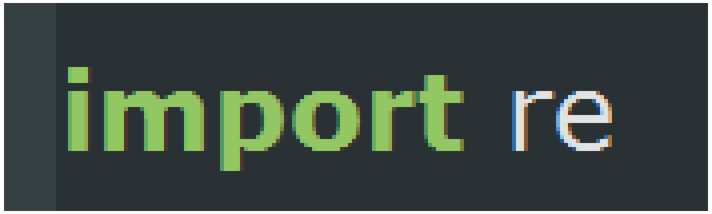
```
print r"C:\new folder\tools"
```

PYTHON REGEX | Raw String



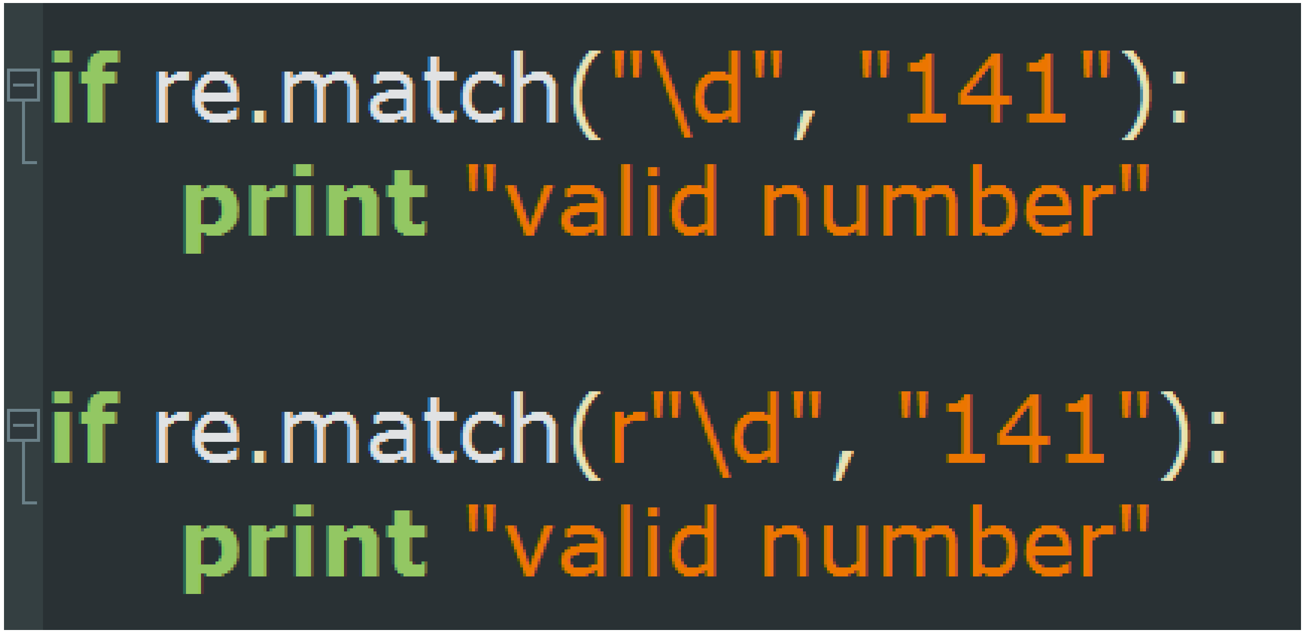
```
print "\tAng\nPanday"

print r"\tAng\nPanday"
```

PYTHON REGEX | The re Module



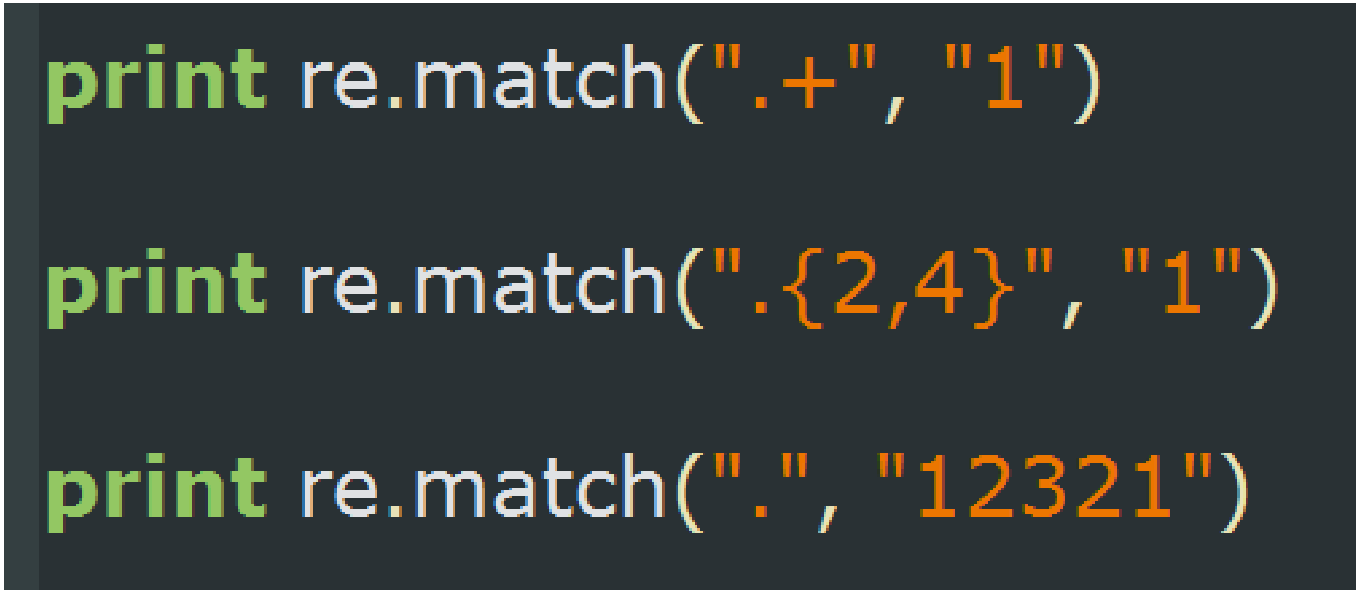
```
import re
```

PYTHON REGEX | Samples



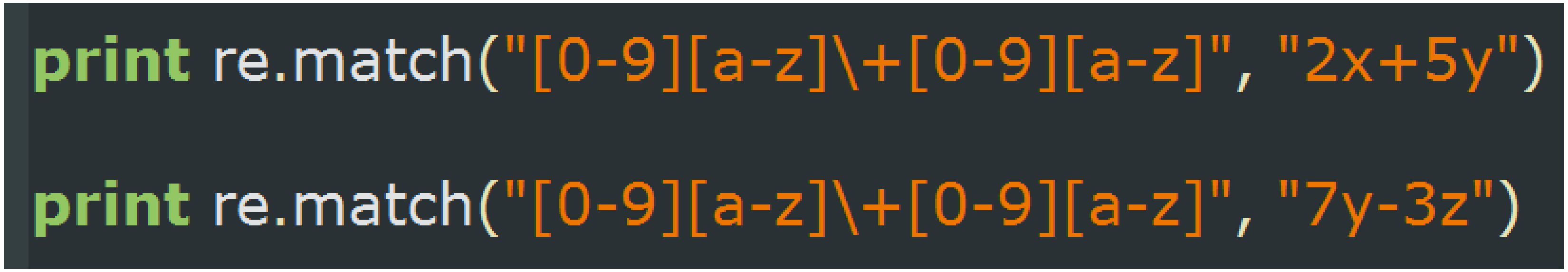
```
if re.match("\d", "141"):
    print "valid number"

if re.match(r"\d", "141"):
    print "valid number"
```

PYTHON REGEX | Samples



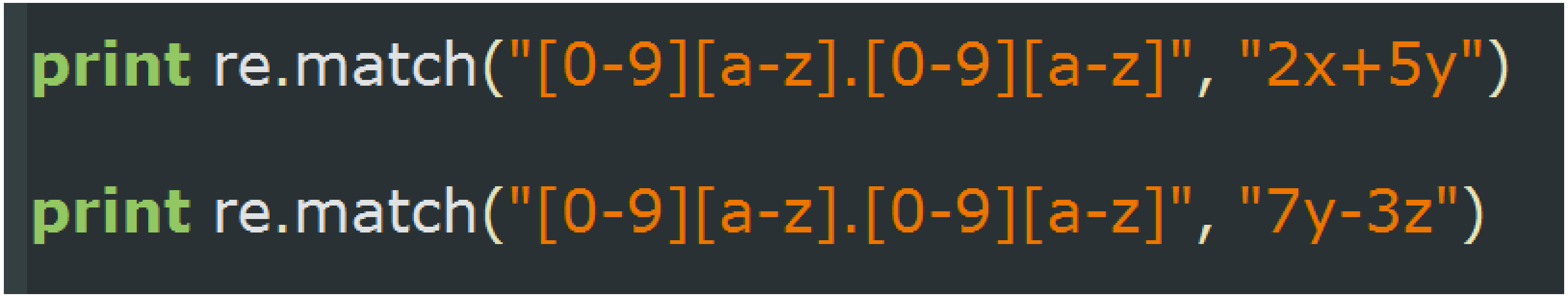
```
print re.match(".+", "1")

print re.match(".{2,4}", "1")

print re.match(".", "12321")
```

PYTHON REGEX | Samples



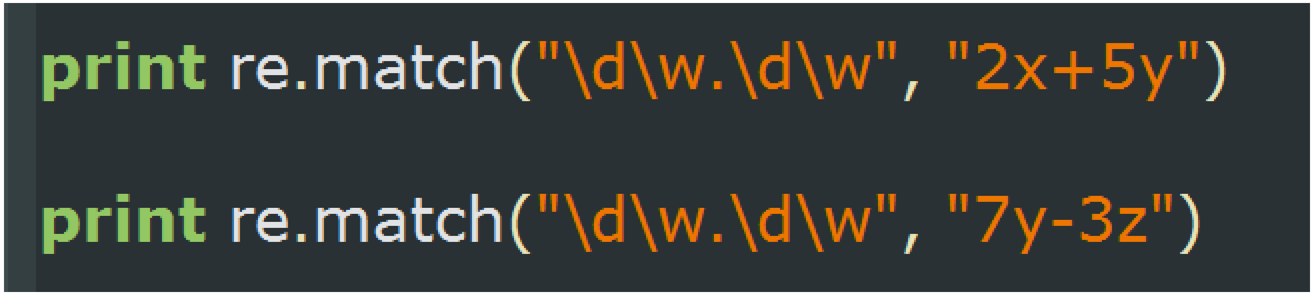
```
print re.match("[0-9][a-z]\+[0-9][a-z]", "2x+5y")

print re.match("[0-9][a-z]\+[0-9][a-z]", "7y-3z")
```

PYTHON REGEX | Samples



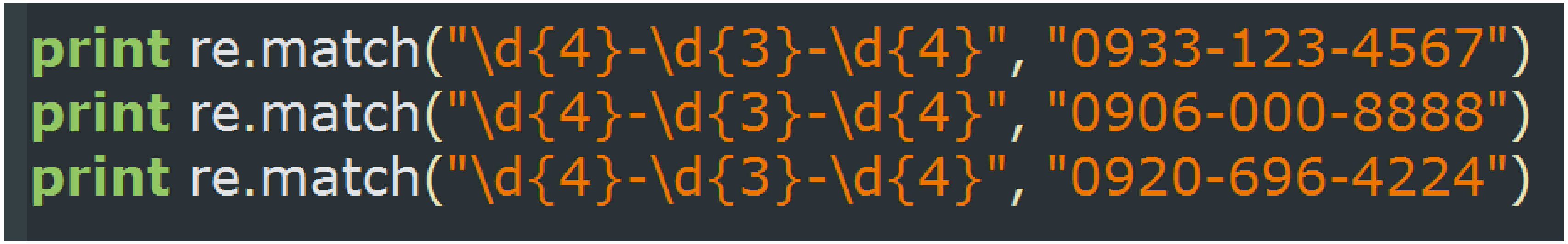
```
print re.match("[0-9][a-z].[0-9][a-z]", "2x+5y")

print re.match("[0-9][a-z].[0-9][a-z]", "7y-3z")
```

PYTHON REGEX | Samples



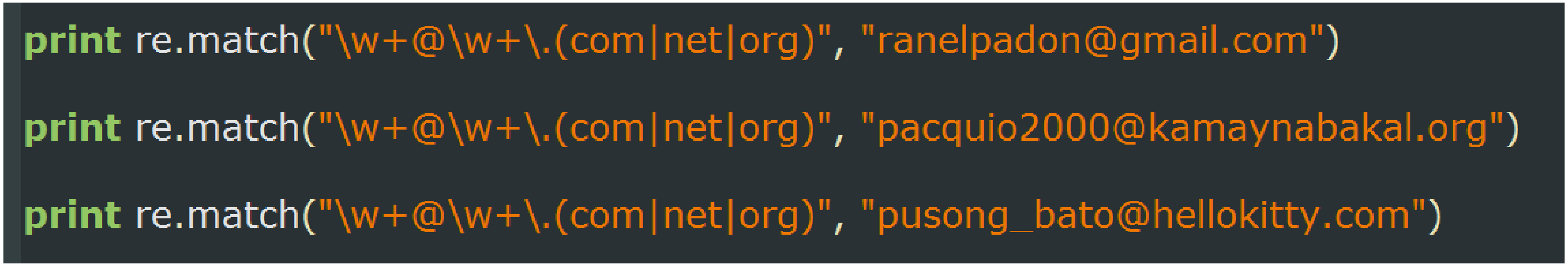
```
print re.match("\d\w.\d\w", "2x+5y")

print re.match("\d\w.\d\w", "7y-3z")
```

PYTHON REGEX | Samples



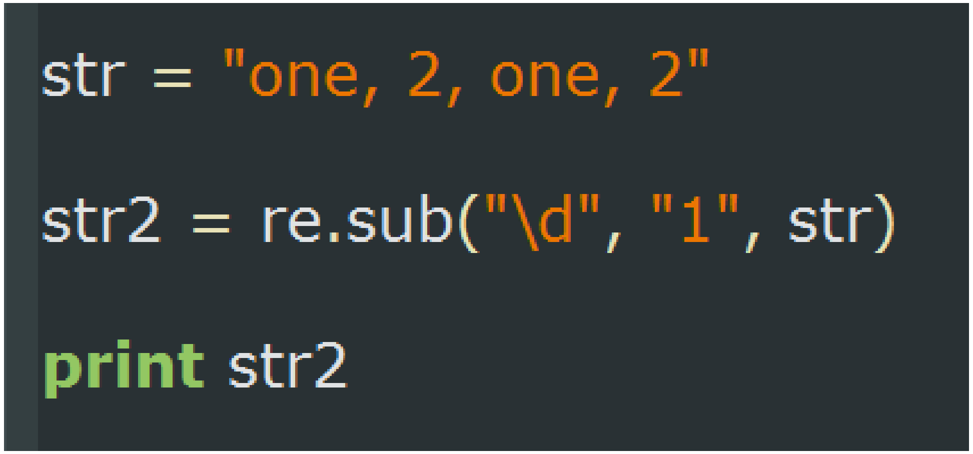
```
print re.match("\d{4}-\d{3}-\d{4}", "0933-123-4567")
print re.match("\d{4}-\d{3}-\d{4}", "0906-000-8888")
print re.match("\d{4}-\d{3}-\d{4}", "0920-696-4224")
```

PYTHON REGEX | Samples



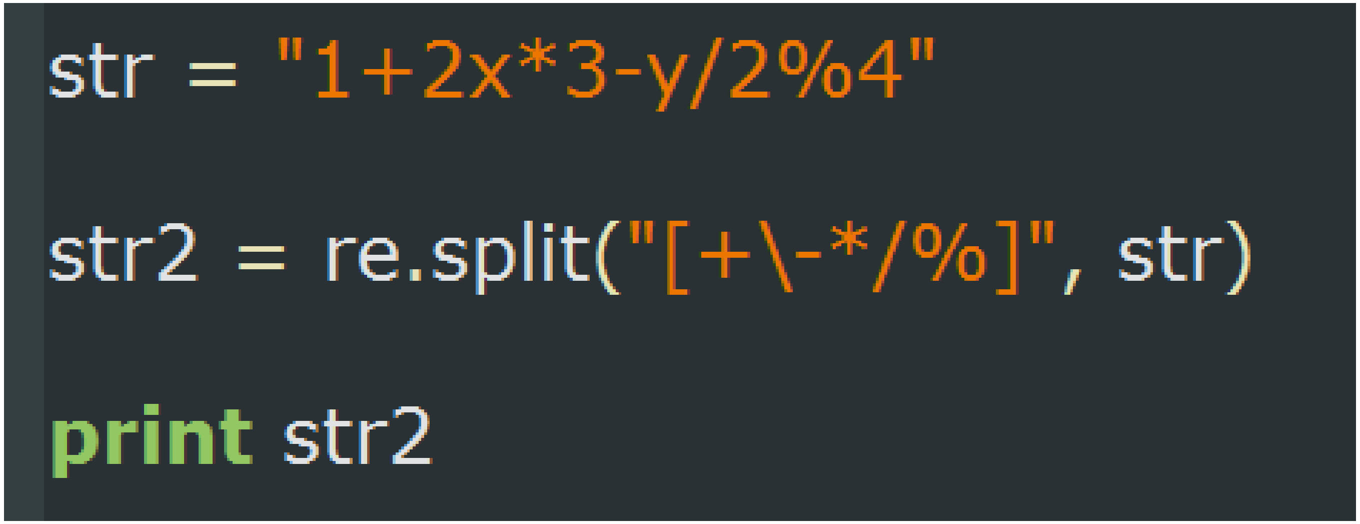
```
print re.match("\w+@\w+\.(com|net|org)", "ranelpadon@gmail.com")

print re.match("\w+@\w+\.(com|net|org)", "pacquio2000@kamaynabakal.org")

print re.match("\w+@\w+\.(com|net|org)", "pusong_bato@hellokitty.com")
```

PYTHON REGEX | Samples



```
str = "one, 2, one, 2"

str2 = re.sub("\d", "1", str)

print str2
```

PYTHON REGEX | Samples



```
str = "1+2x*3-y/2%4"

str2 = re.split("[+\-*/%]", str)

print str2
```

RegEx

Regular Expression

REFERENCES

❑ Deitel, Deitel, Liperi, and Wiedermann - Python: How to Program (2001).

❑ Disclaimer: Most of the images/information used here have no proper source citation, and I do not claim ownership of these either. I don't want to reinvent the wheel, and I just want to reuse and reintegrate materials that I think are useful or cool, then present them in another light, form, or perspective. Moreover, the images/information here are mainly used for illustration/educational purposes only, in the spirit of openness of data, spreading light, and empowering people with knowledge. ☺