Introduction to regular expressions

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and special metacharacters that describes patterns to find text or positions within a text*

What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of **normal characters** and special metacharacters that describes patterns to find text or positions within a text*

- Normal characters match themselves (`st`)

What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and **special metacharacters** that describes patterns to find text or positions within a text*

- Metacharacters represent types of characters (`\d` , `\s` , `\w`) or ideas (`{3,10}`)

What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and **special metacharacters** that describes patterns to find text or positions within a text*

- Metacharacters represent types of characters (`\d` , `\s` , `\w`) or ideas (`{3,10}`)

What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and **special metacharacters** that describes patterns to find text or positions within a text*

- Metacharacters represent types of characters (`\d` , `\s` , `\w`) or ideas (`{3,10}`)

What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and **special metacharacters** that describes patterns to find text or positions within a text*

- Metacharacters represent types of characters (`\d` , `\s` , `\w`) or ideas (`{3,10}`)

What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and special metacharacters that describes **patterns** to find text or positions within a text*

- Pattern: a sequence of characters that maps to words or punctuation

What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and special metacharacters that describes patterns* ***to find text or positions within a text***

- Pattern matching usage:
  - Find and replace text
  - Validate strings
- Very powerful and fast

The re module

```
import re
```

- Find all matches of a pattern:

re.findall(r"regex", string)

```
re.findall(r"#movies", "Love #movies! I had fun yesterday going to the #movies")
```

```
['#movies', '#movies']
```

The re module

```
import re
```

- Split string at each match:

re.split(r"regex", string)

```
re.split(r"!", "Nice Place to eat! I'll come back! Excellent meat!")
```

```
['Nice Place to eat', " I'll come back", ' Excellent meat', '']
```

The re module

```
import re
```

- Replace one or many matches with a string:

re.sub(r"regex", new, string)

```
re.sub(r"yellow", "nice", "I have a yellow car and a yellow house in a yellow neighborhood")
```

```
'I have a nice car and a nice house in a nice neighborhood'
```

Supported metacharacters

Metacharacter	Meaning
\d	Digit

```
re.findall(r"User\d", "The winners are: User9, UserN, User8")
```

```
['User9', 'User8']
```

Metacharacter	Meaning
\D	Non-digit

```
re.findall(r"User\D", "The winners are: User9, UserN, User8")
```

```
['UserN']
```

Supported metacharacters

Metacharacter	Meaning
\w	Word

```
re.findall(r"User\w", "The winners are: User9, UserN, User8")
```

```
['User9', 'UserN', 'User8']
```

Metacharacter	Meaning
\W	Non-word

```
re.findall(r"\W\d", "This skirt is on sale, only $5 today!")
```

```
['$5']
```

Supported metacharacters

Metacharacter	Meaning
\s	Whitespace

```
re.findall(r"Data\sScience", "I enjoy learning Data Science")
```

```
['Data Science']
```

Metacharacter	Meaning
\S	Non-Whitespace

```
re.sub(r"ice\Scream", "ice cream", "I really like ice-cream")
```

```
'I really like ice cream'
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Repetitions

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Science

datacamp

Repeated characters

Validate the following string:

password1234

Repeated characters

Validate the following string:

password1234

Repeated characters

Validate the following string:

password1234

Repeated characters

Validate the following string:


```
import re
password = "password1234"
```

```
re.search(r"\w\w\w\w\w\w\w\w\d\d\d\d", password)
```

```
<_sre.SRE_Match object; span=(0, 12), match='password1234'>
```

Repeated characters

Validate the following string:


```
import re
password = "password1234"
```

```
re.search(r"\w{8}\d{4}", password)
```

```
<_sre.SRE_Match object; span=(0, 12), match='password1234'>
```

> **Quantifiers:**

> A metacharacter that tells the regex engine how many times to match a character immediately to its left.

Quantifiers

- Once or more: `+`

```
text = "Date of start: 4-3. Date of registration: 10-04."
```

```
re.findall(r"       ", text)
```

Quantifiers

- Once or more: `+`

```
text = "Date of start: 4-3. Date of registration: 10-04."
```

```
re.findall(r"\d+-   ", text)
```

Quantifiers

- Once or more: `+`

```
text = "Date of start: 4-3. Date of registration: 10-04."
```

```
re.findall(r"\d+-\d+", text)
```

```
['4-3', '10-04']
```

Quantifiers

- Zero times or more: `*`

```
my_string = "The concert was amazing! @ameli!a @joh&&n @mary90"
re.findall(r"@\w+\W*\w+", my_string)
```

```
['@ameli!a', '@joh&&n', '@mary90']
```

Quantifiers

- Zero times or once: `?`

```
text = "The color of this image is amazing. However, the colour blue could be brighter."
re.findall(r"colou?r", text)
```

```
['color', 'colour']
```

Quantifiers

- n times at least, m times at most : `{n, m}`

```
phone_number = "John: 1-966-847-3131 Michelle: 54-908-42-42424"
```

```
re.findall(r"                              ", phone_number)
```

Quantifiers

- n times at least, m times at most : `{n, m}`

```
phone_number = "John: 1-966-847-3131 Michelle: 54-908-42-42424"
```

```
re.findall(r"\d{1,2}-                    ", phone_number)
```

Quantifiers

- n times at least, m times at most : `{n, m}`

```
phone_number = "John: 1-966-847-3131 Michelle: 54-908-42-42424"
```

```
re.findall(r"\d{1,2}-\d{3}-           ", phone_number)
```

Quantifiers

- n times at least, m times at most : `{n, m}`

```
phone_number = "John: 1-966-847-3131 Michelle: 54-908-42-42424"
```

```
re.findall(r"\d{1,2}-\d{3}-\d{2,3}-\d{4,}", phone_number)
```

```
['1-966-847-3131', '54-908-42-42424']
```

Quantifiers

- *Immediately to the left*
  - `r"apple+"` : `+` applies to e and not to apple

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Regex metacharacters

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

Looking for patterns

Two different operations to find a match:

re.search(r"regex", string)

```
re.search(r"\d{4}", "4506 people attend the show")
```

```
<_sre.SRE_Match object; span=(0, 4), match='4506'>
```

```
re.search(r"\d+", "Yesterday, I saw 3 shows")
```

```
<_sre.SRE_Match object; span=(17, 18), match='3'>
```

re.match(r"regex", string)

```
re.match(r"\d{4}", "4506 people attend the show")
```

```
<_sre.SRE_Match object; span=(0, 4), match='4506'>
```

```
re.match(r"\d+","Yesterday, I saw 3 shows")
```

```
None
```

Special characters

- Match any character (except newline): `.`


```
my_links = "Just check out this link: www.amazingpics.com. It has amazing photos!"
re.findall(r"www  com", my_links)
```

Special characters

- Match any character (except newline): `.`


```
my_links = "Just check out this link: www.amazingpics.com. It has amazing photos!"
re.findall(r"www.+com", my_links)
```

```
['www.amazingpics.com']
```

Special characters

- Start of the string: `^`

```
my_string = "the 80s music was much better that the 90s"
```

```
re.findall(r"the\s\d+s", my_string)
```

```
['the 80s', 'the 90s']
```

```
re.findall(r"^the\s\d+s", my_string)
```

```
['the 80s']
```

Special characters

- End of the string: `$`

```
my_string = "the 80s music hits were much better that the 90s"
```

```
re.findall(r"the\s\d+s$", my_string)
```

```
['the 90s']
```

Special characters

- Escape special characters: `\`

```
my_string = "I love the music of Mr.Go. However, the sound was too loud."
```

```
print(re.split(r".\s", my_string))
```

```
['', 'lov', 'th', 'musi', 'o', 'Mr.Go', 'However', 'th', 'soun', 'wa', 'to', 'loud.']
```

```
print(re.split(r"\.\s", my_string))
```

```
['I love the music of Mr.Go', 'However, the sound was too loud.']
```

OR operator

- Character: `|`

```
my_string = "Elephants are the world's largest land animal! I would love to see an elephant one day"
```

```
re.findall(r"Elephant|elephant", my_string)
```

```
['Elephant', 'elephant']
```

OR operator

- Set of characters: `[ ]`

```
my_string = "Yesterday I spent my afternoon with my friends: MaryJohn2 Clary3"
```

```
re.findall(r"[a-zA-Z]+\d", my_string)
```

```
['MaryJohn2', 'Clary3']
```

OR operator

- Set of characters: `[ ]`

```
my_string = "My&name&is#John Smith. I%live$in#London."
```

```
re.sub(r"[#$%&]", " ", my_string)
```

```
'My name is John Smith. I live in London.'
```

OR operand

- Set of characters: `[ ]`
  - `^` transforms the expression to negative

```
my_links = "Bad website: www.99.com. Favorite site: www.hola.com"
re.findall(r"www[^0-9]+com", my_links)
```

```
['www.hola.com']
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Greedy vs. non-greedy matching

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

Greedy vs. non-greedy matching

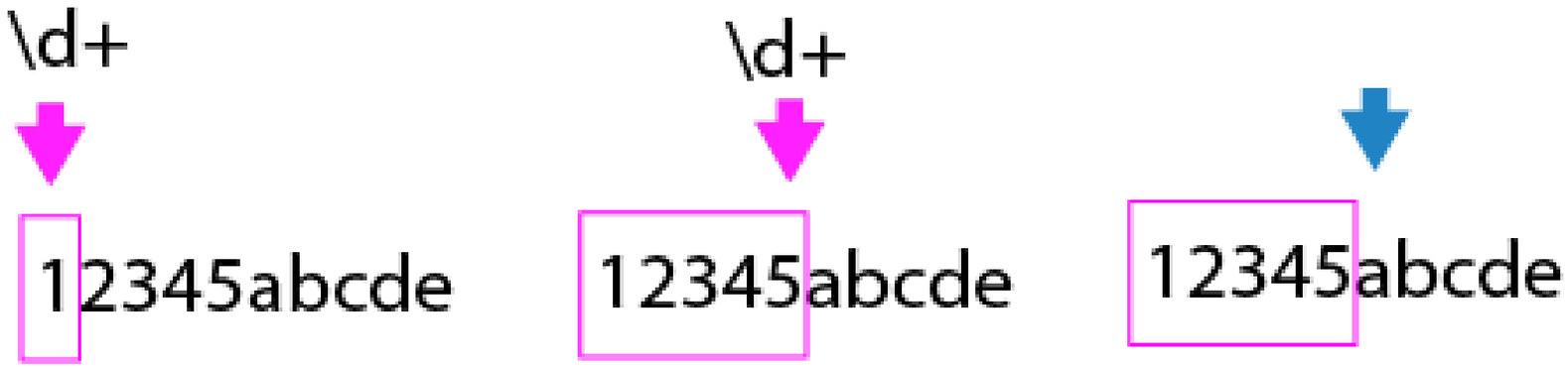
- Two types of matching methods:
  - Greedy
  - Non-greedy or lazy
- Standard quantifiers are greedy by default: `*`, `+`, `?`, `{num, num}`

Greedy matching

- **Greedy**: match as many characters as possible
- Return the *longest match*

```
import re
re.match(r"\d+", "12345bcada")
```

```
<_sre.SRE_Match object; span=(0, 5), match='12345'>
```

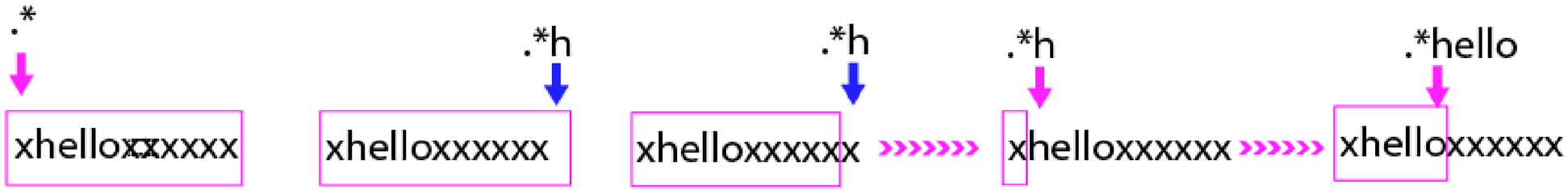


Greedy matching

- Backtracks when too many character matched
- Gives up characters one at a time

```
import re
re.match(r".*hello", "xhelloxxxxxx")
```

```
<_sre.SRE_Match object; span=(0, 6), match='xhello'>
```

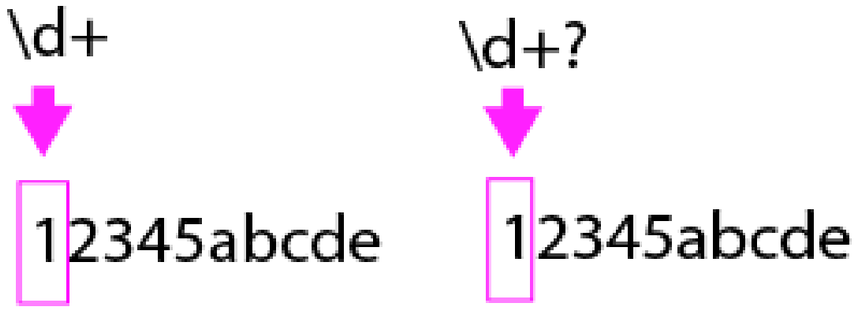


Non-greedy matching

- **Lazy**: match as few characters as needed
- Returns *the shortest match*
- Append `?` to greedy quantifiers

```
import re
re.match(r"\d+?", "12345bcada")
```

```
<_sre.SRE_Match object; span=(0, 1), match='1'>
```



Non-greedy matching

- Backtracks when too few characters matched
- Expands characters one a time

```
import re
re.match(r".*?hello", "xhelloxxxxxx")
```

```
<_sre.SRE_Match object; span=(0, 6), match='xhello'>
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON