Grouping and capturing

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

Group characters

Clary has 2 friends who she spends a lot time with. Susan has 3 brothers while John has 4 sisters.

Group characters

Clary has 2 friends who she spends a lot time with. Susan has 3 brothers while John has 4 sisters.

```
re.findall(r'[A-Za-z]+\s\w+\s\d+\s\w+', text)
```

```
['Clary has 2 friends', 'Susan has 3 brothers', 'John has 4 sisters']
```

Capturing groups

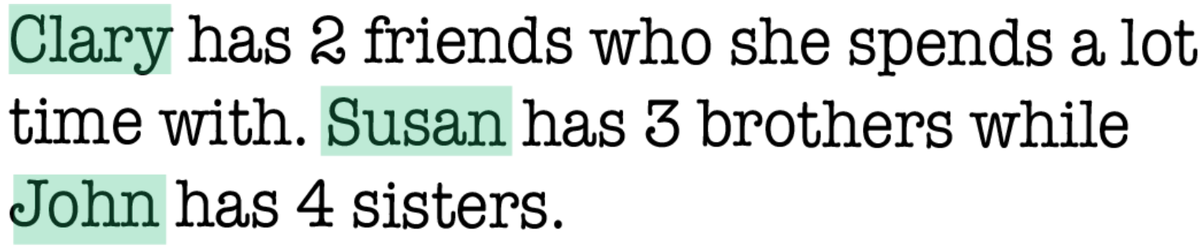
Clary has 2 friends who she spends a lot time with. Susan has 3 brothers while John has 4 sisters.

- Use parentheses to **group** and **capture** characters together

```
([A-Za-z]+)\s\w+\s\d+\s\w+
```

Group

Capturing groups



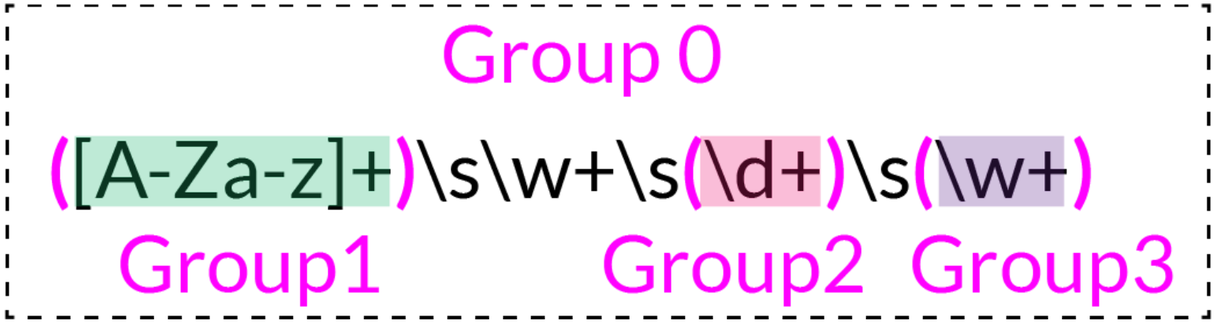
- Use parentheses to group and capture characters together

```
re.findall(r'([A-Za-z]+)\s\w+\s\d+\s\w+', text)
```

```
['Clary', 'Susan', 'John']
```

Capturing groups

Clary has 2 friends who she spends a lot time with. Susan has 3 brothers while John has 4 sisters.



Capturing groups

Clary has 2 friends who she spends a lot time with. Susan has 3 brothers while John has 4 sisters.

```
re.findall(r'([A-Za-z]+)\s\w+\s(\d+)\s(\w+)', text)
```

```
[('Clary', '2', 'friends'),
 ('Susan', '3', 'brothers'),
 ('John', '4', 'sisters')]
```

Capturing groups

- Match a specific subpattern in a pattern
- Use it for further processing

Capturing groups

- Organize the data

```
pets = re.findall(r'([A-Za-z]+)\s\w+\s(\d+)\s(\w+)', "Clary has 2 dogs but John has 3 cats")
pets[0][0]
```

```
'Clary'
```

Capturing groups

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

- Apply a quantifier to the entire group

```
re.search(r"(\d[A-Za-z])+", "My user name is 3e4r5fg")
```

```
<_sre.SRE_Match object; span=(16, 22), match='3e4r5f'>
```

Capturing groups

- Capture a repeated group `(\d+)` vs. repeat a capturing group `(\d)+`

```
my_string = "My lucky numbers are 8755 and 33"
re.findall(r"(\d)+", my_string)
```

```
['5', '3']
```

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

```
['8755', '33']
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Alternation and non-capturing groups

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

Pipe

- Vertical bar or pipe: `|`

```
my_string = "I want to have a pet. But I don't know if I want a cat, a dog or a bird."
re.findall(r"cat|dog|bird", my_string)
```

```
['cat', 'dog', 'bird']
```

Pipe

- Vertical bar or pipe: `|`

```
my_string = "I want to have a pet. But I don't know if I want 2 cats, 1 dog or a bird."
re.findall(r"\d+\scat|dog|bird", my_string)
```

```
['2 cat', 'dog', 'bird']
```

Alternation

- Use groups to choose between optional patterns


```
my_string = "I want to have a pet. But I don't know if I want 2 cats, 1 dog or a bird."
re.findall(r"\d+\s(cat|dog|bird)", my_string)
```

```
['cat', 'dog']
```

Alternation

- Use groups to choose between optional patterns


```
my_string = "I want to have a pet. But I don't know if I want 2 cats, 1 dog or a bird."
re.findall(r"(\d)+\s(cat|dog|bird)", my_string)
```

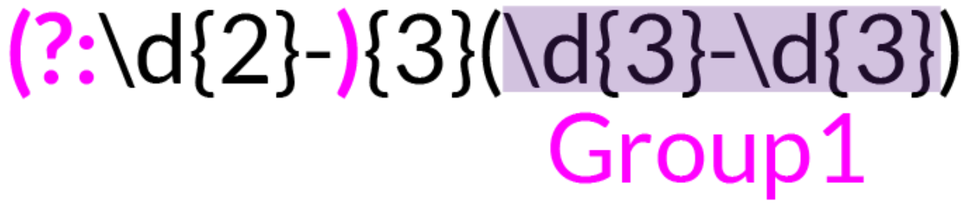
```
[('2', 'cat'), ('1', 'dog')]
```

Non-capturing groups

- Match but **not capture** a group
  - When group is not backreferenced
  - Add `?:` : `(?:regex)`

Non-capturing groups

- Match but **not capture** a group




```
my_string = "John Smith: 34-34-34-042-980, Rebeca Smith: 10-10-10-434-425"
re.findall(r"(?:\d{2}-){3}(\d{3}-\d{3})", my_string)
```

```
['042-980', '434-425']
```

Alternation

- Use non-capturing groups for alternation

```
my_date = "Today is 23rd May 2019. Tomorrow is 24th May 19."
re.findall(r"(\d+)(?:th|rd)", my_date)
```

```
['23', '24']
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Backreferences

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

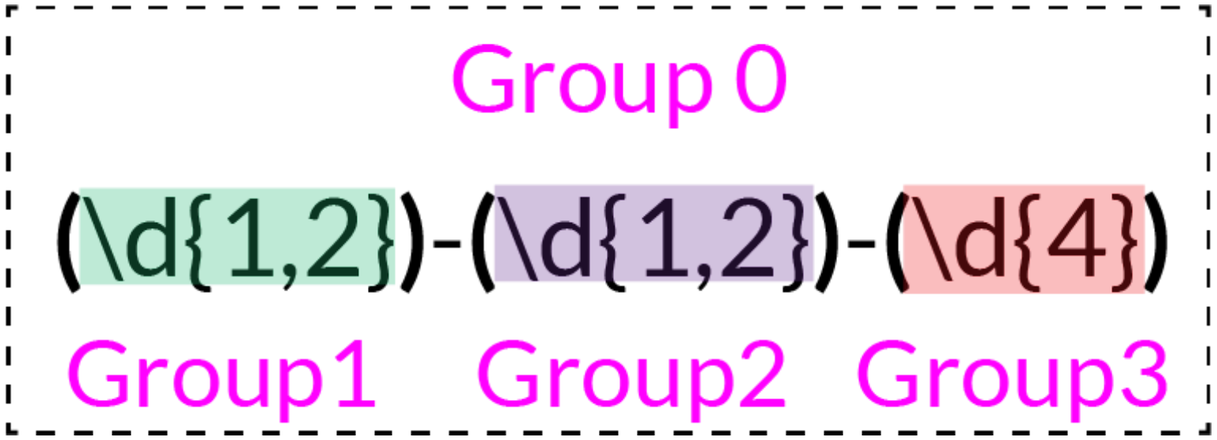
Numbered groups

Python 3.0 was released on 12-03-2008. It was a major revision of the language. Many of its major features were backported to Python 2.6.x and 2.7.x version series.

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

Numbered groups

Python 3.0 was released on 12-03-2008. It was a major revision of the language. Many of its major features were backported to Python 2.6.x and 2.7.x version series.



Numbered groups

```
text = "Python 3.0 was released on 12-03-2008."
```

```
information = re.search('(\d{1,2})-(\d{2})-(\d{4})', text)
information.group(3)
```

```
'2008'
```

```
information.group(0)
```

```
'12-03-2008'
```

Named groups

- Give a name to groups

(?P<name>regex)

Named groups

- Give a name to groups

```
text = "Austin, 78701"
cities = re.search(r"(?P<city>[A-Za-z]+).*?(?P<zipcode>\d{5})", text)
cities.group("city")
```

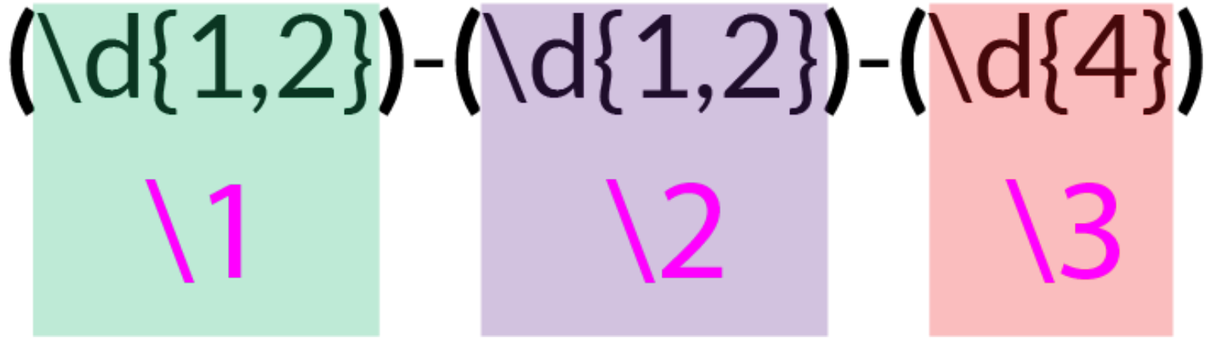
```
'Austin'
```

```
cities.group("zipcode")
```

```
'78701'
```

Backreferences

- Using capturing groups to reference back to a group



Backreferences

- Using numbered capturing groups to reference back

```
sentence = "I wish you a happy happy birthday!"
re.findall(r"(\w+)\s  ", sentence)
```

Backreferences

- Using numbered capturing groups to reference back

```
sentence = "I wish you a happy happy birthday!"
re.findall(r"(\w+)\s\1", sentence)
```

```
['happy']
```

Backreferences

- Using numbered capturing groups to reference back

```
sentence = "I wish you a happy happy birthday!"
re.sub(r"(\w+)\s\1", r"\1", sentence)
```

```
'I wish you a happy birthday!'
```

Backreferences

- Using named capturing groups to reference back

(?P<name>regex)

?P=name

```
sentence = "Your new code number is 23434. Please, enter 23434 to open the door."
re.findall(r"(?P<code>\d{5}).*?(?P=code)", sentence)
```

```
['23434']
```

Backreferences

- Using named capturing groups to reference back

(?P<name>regex)

\g<name>

```
sentence = "This app is not working! It's repeating the last word word."
re.sub(r"(?P<word>\w+)\s(?P=word)", r"\g<word>", sentence)
```

```
'This app is not working! It's repeating the last word.'
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Lookaround

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

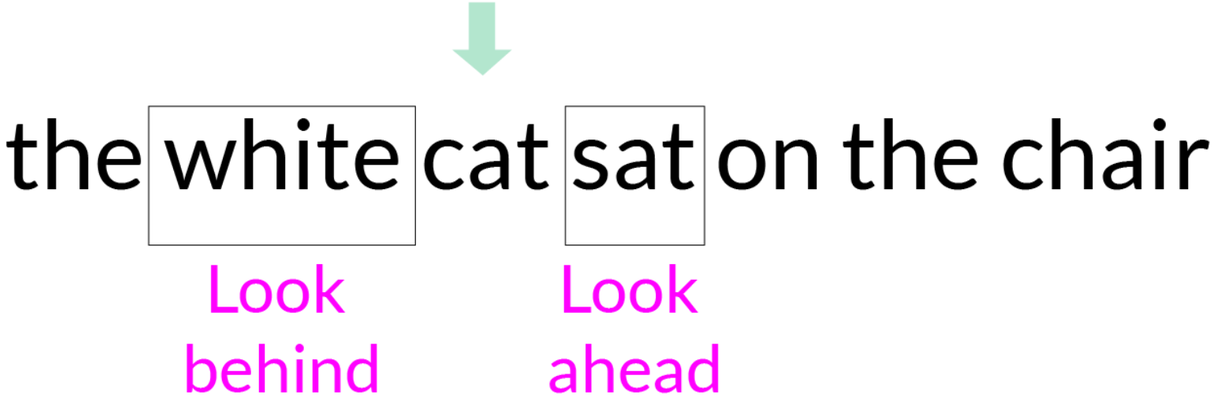
Looking around

- Allow us to confirm that sub-pattern is ahead or behind main pattern

the white cat sat on the chair

Looking around

- Allow us to confirm that sub-pattern is ahead or behind main pattern



> At my current position in the matching process, look ahead or behind and examine whether some pattern matches or not match before continuing.

Look-ahead

- Non-capturing group
- Checks that the first part of the expression is followed or not by the lookahead expression
- Return only the first part of the expression

the white cat sat on the chair

Look ahead

positive (?=sat)

negative (?!run)

Positive look-ahead

- Non-capturing group
- Checks that the first part of the expression is followed by the lookahead expression
- Return only the first part of the expression

```
my_text = "tweets.txt transferred, mypass.txt transferred, keywords.txt error"
re.findall(r"\w+\.txt                  ", my_text)
```

Positive look-ahead

- Non-capturing group
- Checks that the first part of the expression is followed by the lookahead expression
- Return only the first part of the expression

```
my_text = "tweets.txt transferred, mypass.txt transferred, keywords.txt error"
re.findall(r"\w+\.txt(?=\stransferred)", my_text)
```

```
['tweets.txt', 'mypass.txt']
```

Negative look-ahead

- Non-capturing group
- Checks that the first part of the expression is **not** followed by the lookahead expression
- Return only the first part of the expression

```
my_text = "tweets.txt transferred, mypass.txt transferred, keywords.txt error"
re.findall(r"\w+\.txt                  ", my_text)
```

Negative look-ahead

- Non-capturing group
- Checks that the first part of the expression is **not** followed by the lookahead expression
- Return only the first part of the expression

```
my_text = "tweets.txt transferred, mypass.txt transferred, keywords.txt error"
re.findall(r"\w+\.txt(?!\stransferred)", my_text)
```

```
['keywords.txt']
```

Look-behind

- Non-capturing group
- Get all the matches that are preceded or not by a specific pattern.
- Return pattern after look-behind expression

the white cat sat on the chair

Look behind

positive (?<=white)

negative (?<!brown)

Positive look-behind

- Non-capturing group
- Get all the matches that are preceded by a specific pattern.
- Return pattern after look-behind expression

```
my_text = "Member: Angus Young, Member: Chris Slade, Past: Malcolm Young, Past: Cliff Williams."
re.findall(r"                \w+\s\w+", my_text)
```

Positive look-behind

- Non-capturing group
- Get all the matches that are preceded by a specific pattern.
- Return pattern after look-behind expression

```
my_text = "Member: Angus Young, Member: Chris Slade, Past: Malcolm Young, Past: Cliff Williams."
re.findall(r"(?<=Member:\s)\w+\s\w+", my_text)
```

```
['Angus Young', 'Chris Slade']
```

Negative look-behind

- Non-capturing group
- Get all the matches that are **not** preceded by a specific pattern.
- Return pattern after look-behind expression

```
my_text = "My white cat sat at the table. However, my brown dog was lying on the couch."
re.findall(r"(?<!brown\s)(cat|dog)", my_text)
```

```
['cat']
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Finishing line

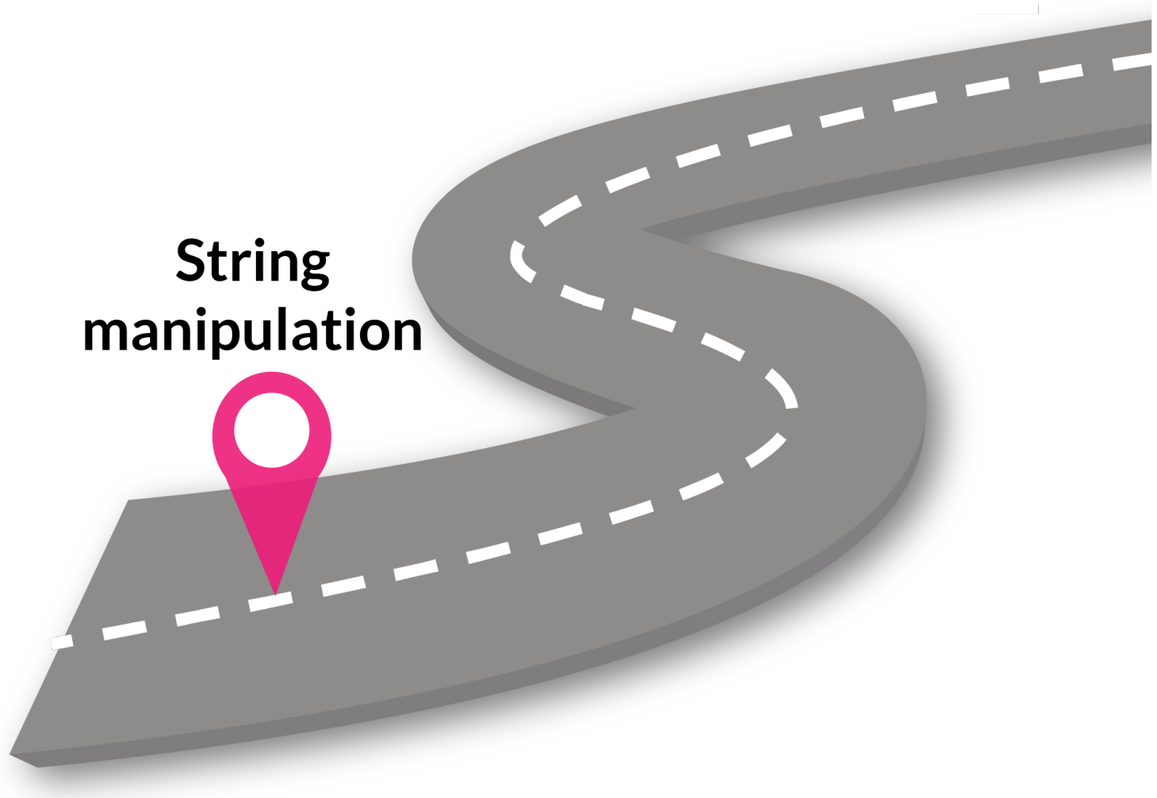
REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

```
r"(Congratulations!)+"
```

Our journey



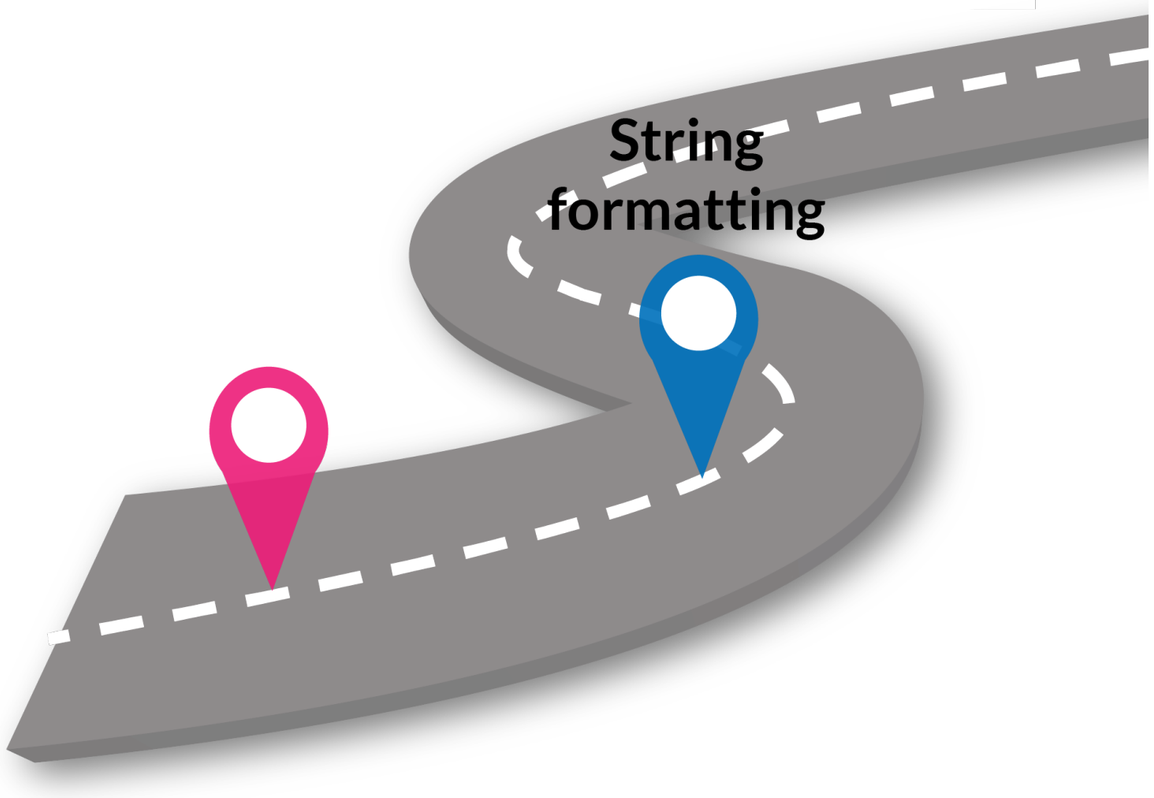
Key concepts

Concatenate and split

Index and slice strings

Replace and remove characters

Our journey

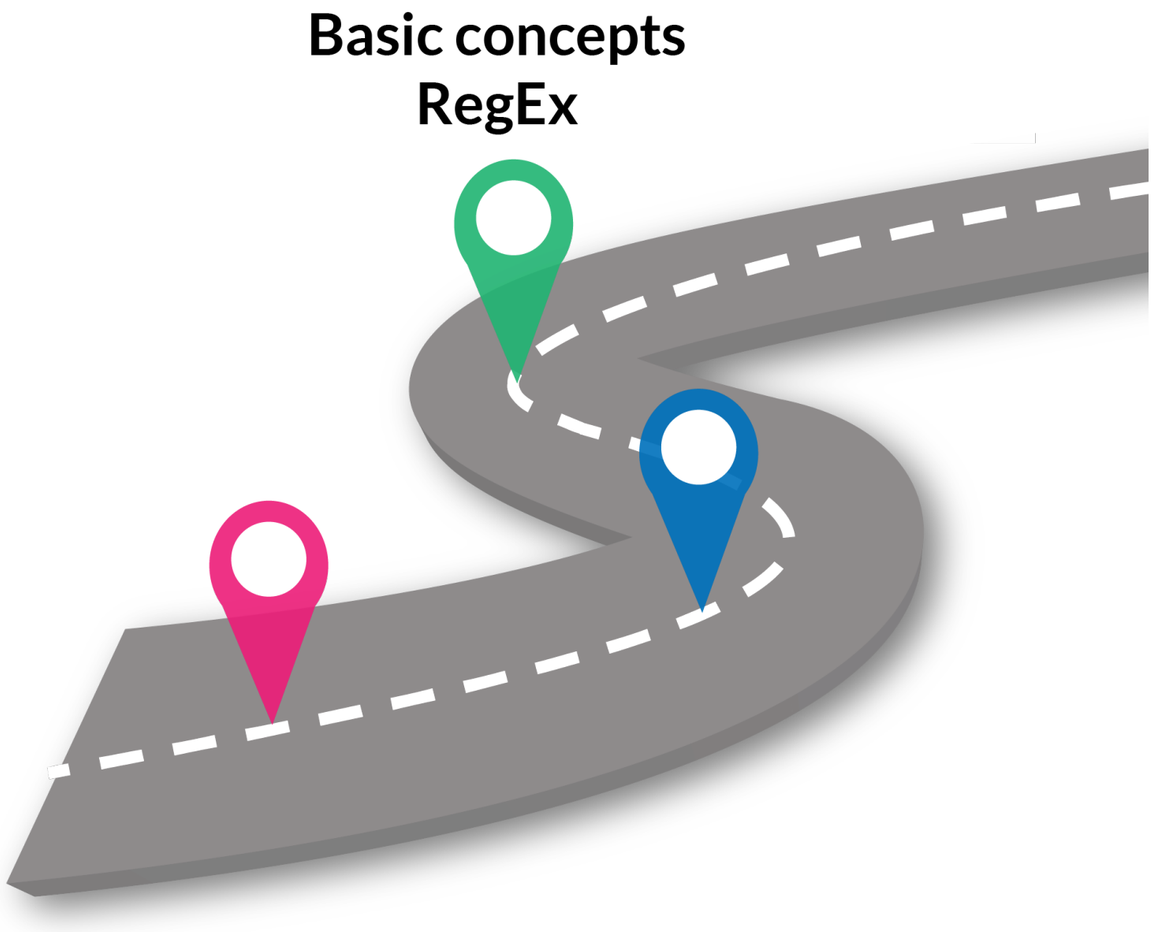


Insert custom strings into a predefined text

Three string formatting methods

Best approach according to situation

Our journey



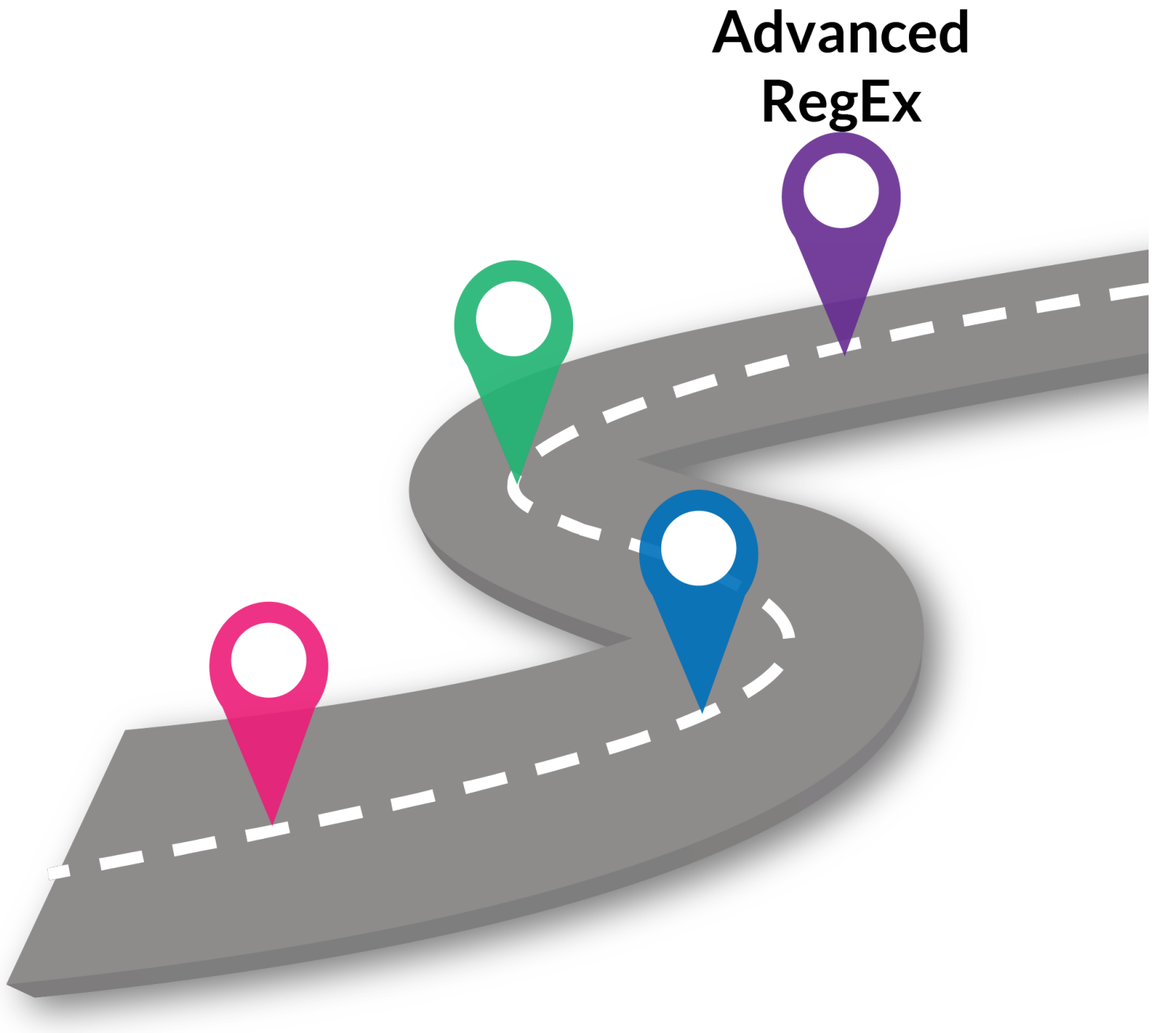
Basic syntax

Normal characters

Metacharacters

Greedy and non-greedy quantifiers

Our journey



Capturing and non-capturing groups

Backreference a pattern

Lookaround an expression

Last tips

- ✓ Practice
- ✓ Apply
- ✓ Have fun

Thank you!

REGULAR EXPRESSIONS IN PYTHON