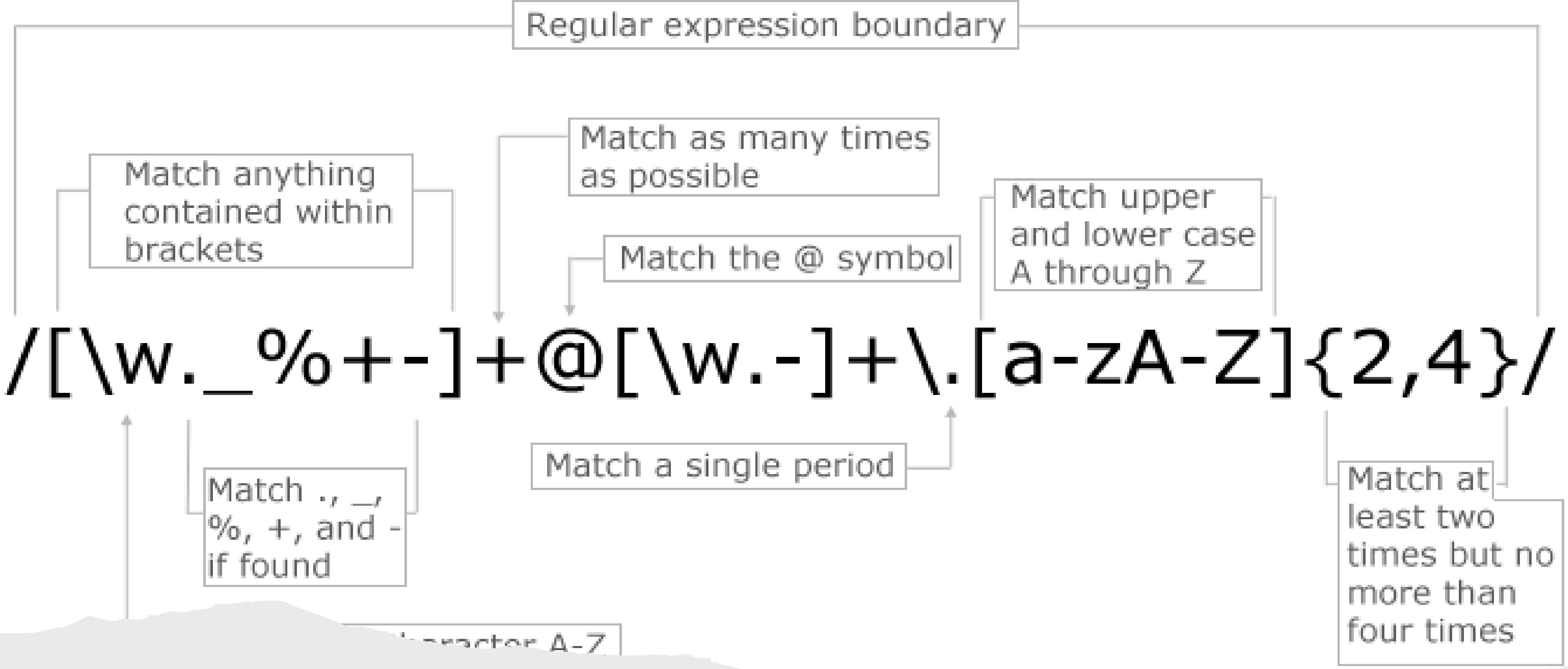
Lisbon School of Economics & Management
Universidade de Lisboa

LISBOA
UNIVERSIDADE DE LISBOA

REGULAR EXPRESSIONS

Carlos J. Costa


Regular expression boundary
Match anything contained within brackets
Match as many times as possible
Match the @ symbol
Match upper and lower case A through Z
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
Match ., _, %, +, and - if found
Match a single period
Match at least two times but no more than four times

Regular Expressions

- Regular expressions are a powerful language for matching text patterns.

Regular Expressions

```
import re
text = "The author's name is Porter"
result = re.search("^The.*Porter$", text)
if result:
  print("ok!")
else:
  print("Nop…")
```

Regular Expressions

findall - Returns a list containing all matches

search - Returns a Match object if there is a match anywhere in the string

split- Returns a list where the string has been split at each match

sub- Replaces one or many matches with a string

Regular Expressions

Literals: Characters that match themselves.

- Example: The pattern hello will match the string "hello" exactly.

```
import re
text = "hello world"
pattern = r'hello'
match = re.search(pattern, text)
print(match.group())  # Output: hello
```

Regular Expressions

Character Classes: Specify a set of characters to match.

- Square brackets [ ] are used to define a character class.
- Example: [aeiou] matches any vowel.

```
import re

text = "apple banana cherry"
pattern = r'[aeiou]'
matches = re.findall(pattern, text)
print(matches)

# Output: ['a', 'e', 'a', 'a', 'e']
```

Regular Expressions

Quantifiers: Indicate how many times a character or group can occur.

*: Matches zero or more occurrences.

+: Matches one or more occurrences.

?: Matches zero or one occurrence.

{n}: Matches exactly n occurrences.

{n,}: Matches at least n occurrences.

{n,m}: Matches between n and m occurrences.

Example: \d{3} matches exactly three digits.

```
import re

text = "12345 67890"

pattern = r'\d{3}’

matches = re.findall(pattern, text)

print(matches)  # Output: ['123', '678']
```

Regular Expressions

Anchors: Specify the position in the string where the match should occur.

^: Matches the start of the string.

$: Matches the end of the string.

\b: Matches a word boundary.

Example: ^\d{3} matches a string starting with three digits.

Other Example: Matches "start" at the beginning of the string

```
import re

text = "start middle end"

pattern = r'^start'  # match = re.search(pattern, text)

print(match.group())  # Output: start
```

Regular Expressions

Escape Sequences: Special sequences that match specific characters.

- \d: Matches any digit (equivalent to [0-9]).
- \w: Matches any alphanumeric character (equivalent to [a-zA-Z0-9_]).
- \s: Matches any whitespace character.
- Example: Matches a digit, whitespace, and non-word character

```
import re
text = "abc 123 !@#"
pattern = r'\d\s\W'  match = re.search(pattern, text)
print(match.group())  # Output: 3  !
```

Regular Expressions

Grouping and Capturing: Use parentheses () to group characters.

( ): Groups characters together.

(?:): Non-capturing group.

Example: (ab)+ matches one or more occurrences of "ab".

```
import re

text = "abababab"

pattern = r'(ab)+'

match = re.search(pattern, text)

print(match.group())  # Output: abababab
```

Regular Expressions

Alternation: Match one of several patterns.

- |: Alternation operator.
- Example: cat|dog matches either "cat" or "dog".

```
text = "cat dog bird"
pattern = r'cat|dog'
match = re.search(pattern, text)
print(match.group())  # Output: cat
```

Regular Expressions

Modifiers: Change the behavior of a pattern.

i: Case-insensitive matching.

m: Multiline mode.

s: Dot matches newline characters.

Example: (?i)hello matches "hello" case-insensitively.

```
import re
text = "HELLO\nworld"
pattern = r'(?i)hello’
match = re.search(pattern, text)
print(match.group())  # Output: HELLO
```

Regular Expressions

Special Characters:

Have special meanings in regular expressions.

., ^, $, *, +, ?, {, }, [,], (,), \, |

Example: Matches one or more digits

```
import re
text = "I have 10 dollars."
pattern = r'\d+'
match = re.search(pattern, text)
print(match.group())  # Output: 10
```

Example 01

- Matching a specific pattern in a string:
- Matches words that are exactly 5 characters long

```
import re
text = "The quick brown fox jumps over the lazy dog"
pattern = r'\b\w{5}\b'
matches = re.findall(pattern, text)
print(matches)
```

- Output: ['quick', 'brown']

Example 02

- Replacing patterns in a string::
- Matches email addresses

```
import re
text = "Hello, my email is user@example.com"
pattern = r'\b\w+@\w+\.\w+\b'
new_text = re.sub(pattern, 'your_email@example.com', text)
print(new_text)
```

- Output:

Hello, my email is your_email@example.com

Example 03

- Finding and extracting specific information from a string
- Matches names and phone numbers

```
import re
text = "John: 555-1234, Lisa: 555-9876, Mike: 555-5678"
pattern = r'(\w+): (\d{3}-\d{4})'
matches = re.findall(pattern, text)
print(matches)
```

- Output:

```
[('John', '555-1234'), ('Lisa', '555-9876'), ('Mike', '555-5678')]
```

Example 04

- Splitting a string based on a pattern
- Matches whitespace characters

```
text = "The quick brown fox jumps over the lazy dog"
pattern = r'\s+'
words = re.split(pattern, text)
print(words)
```

- Output:

```
['The', 'quick', 'brown', 'fox', 'jumps',
'over', 'the', 'lazy', 'dog']
```

Example 05

- Validating input format
- Matches a standard email format

```
import re
def is_valid_email(email):
    pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
    return bool(re.match(pattern, email))

print(is_valid_email("invalid-email"))
```

- Output:

```
False
```

More information

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