# Business Programming (using Python)

Zhaohu (Jonathan) Fan

Information Technology Management

Scheller College of Business

Georgia Institute of Technology

September 21, 2023

# Main topics

- Data Structures
  - Strings & Regular Expressions (II)
  - Exercises

# Data Structures - Strings & Regular Expressions (II)

# Business context question

- Suppose you have a database of fictional animal names, and you want to ensure that each entry starts with a name of **at least three characters**, followed by **an optional single character and space (representing a title or honorific)**, and then ends with **a name of at least three characters**.

- Sample Entries:

  - Cat A Dog
  - Elephant B Mouse
  - Tiger Cheetah
  - Ko Lion
  - Ant Eater

# Why the Regular Expression (Regex)?

The regular expression (Regex) is

- **a sequence of characters defining a search pattern**.
- crucial for precise string matching and data validation.
- **Question**:
    - **Determine the validity of given animal names based on specific criteria**.

# Step-by-step explanation: step 1

- Each entry **starts** with a name of **at least three characters**
  - The caret symbol `^` is used to check if a string **starts with** a certain character.
  - `\w` matches any alphanumeric character (digits and alphabets). Equivalent to `[a-zA-Z0-9_]`. By the way, underscore `_` is also considered an alphanumeric character.
  - `\w{3,}` matches any word character (letters and numbers) 3 or more times.
    - It ensures the first name has at least three characters.
    - So, **"Ko" in "Ko Lion" would not match because "Ko" has only two characters.**

**Code**

```
^\w{3,}
```

# Step 2

- `\s` - Matches where a string contains any whitespace character.
    - In our example, there is **a space between the first name and the second name or title**.

**Code**

```
^\w{3,}\s
```

# Step 3:followed by an optional single character and space

- **`(\w\s)?`**: matches **a single word character followed by a whitespace character**, but this entire group is **optional because of the ? at the end**.

  -In our toy example, it checks for titles or honorifics like `"A"` in `"Cat A Dog"` but does not mandate it.

**Code**

```
^\w{3,}\s(\w\s)?
```

# Step 4: ends with a name of at least three characters

- `\w{3,}`: matches any word **character 3 or more times**.
    - In our example, it ensures the second name (or the only name if there's no title) has at least three characters.

**Code**

```
^\w{3,}\s(\w\s)?\w{3,}
```

# Python script

- Let's write a Python script that tests the given animal names against the specified regular expression to determine if they match the desired pattern.
  - Run this script, and it will print out whether each animal name in the list matches the regular expression pattern or not.

```
import re
# List of sample entries
animal_names = [
    "Cat A Dog",
    "Elephant B Mouse",
    "Tiger Cheetah",
    "Ko Lion",
    "Ant Eater"
]
# Regular expression pattern
pattern = '^\w{3,}\s(\w\s)?\w{3,}'
# Iterate over the animal names and test against the pattern
for name in animal_names:
    if re.match(pattern, name):
        print(f"'{name}' matches the pattern!")
    else:
        print(f"'{name}' does NOT match the pattern!")
```

## Output

```
'Cat A Dog' matches the pattern!
'Elephant B Mouse' matches the pattern!
'Tiger Cheetah' matches the pattern!
'Ko Lion' does NOT match the pattern!
'Ant Eater' matches the pattern!
```

# Results for our example

- Suppose you have a database of fictional animal names, and you want to ensure that each entry starts with a name of **at least three characters**, followed by **an optional single character and space (representing a title or honorific)**, and then ends with **a name of at least three characters**.

- Sample Entries:

  - **Cat A Dog** matches the pattern. Starts with a word of three characters, followed by a title and another word with three characters.
  - **Elephant B Mouse** matches the pattern.
  - **Tiger Cheetah** matches the pattern. Starts and ends with a word of three or more characters.
  - **Ko Lion** doesn't match because the first word "Ko" is less than three characters.
  - **Ant Eater** matches the pattern.

# Cheat sheet for metaCharacters

### Regular Expression Basics

| | |
|---|---|
| . | Any character except newline |
| a | The character a |
| ab | The string ab |
| a\|b | a or b |
| a* | 0 or more a's |
| \ | Escapes a special character |

### Regular Expression Quantifiers

| | |
|---|---|
| * | 0 or more |
| + | 1 or more |
| ? | 0 or 1 |
| {2} | Exactly 2 |
| {2, 5} | Between 2 and 5 |
| {2,} | 2 or more |

Default is greedy. Append ? for reluctant.

### Regular Expression Groups

| | |
|---|---|
| (...) | Capturing group |
| (?:...) | Non-capturing group |
| \Y | Match the Y'th captured group |

### Regular Expression Character Classes

| | |
|---|---|
| [ab-d] | One character of: a, b, c, d |
| [^ab-d] | One character except: a, b, c, d |
| [\b] | Backspace character |
| \d | One digit |
| \D | One non-digit |
| \s | One whitespace |
| \S | One non-whitespace |
| \w | One word character |
| \W | One non-word character |

### Regular Expression Assertions

| | |
|---|---|
| ^ | Start of string |
| $ | End of string |
| \b | Word boundary |
| \B | Non-word boundary |
| (?=...) | Positive lookahead |
| (?!...) | Negative lookahead |

### Regular Expression Flags

| | |
|---|---|
| g | Global Match |
| i | Ignore case |
| m | ^ and $ match start and end of line |

### Regular Expression Special Characters

| | |
|---|---|
| \n | Newline |
| \r | Carriage return |
| \t | Tab |
| \0 | Null character |
| \YYY | Octal character YYY |
| \xYY | Hexadecimal character YY |
| \uYYYY | Hexadecimal character YYYY |
| \cY | Control character Y |

### Regular Expression Replacement

| | |
|---|---|
| $$ | Inserts $ |
| $& | Insert entire match |
| $` | Insert preceding string |
| $' | Insert following string |
| $Y | Insert Y'th captured group |

# Exercises

- Please click on the link provided below.
  - In-Class Exercise