Computación y Estructuras Discretas III

Andrés A. Aristizábal P.
aaaristizabal@icesi.edu.co
Ángela Villota
apvillota@icesi.edu.co

Departamento de Computación y Sistemas Inteligentes

2024-1

Agenda

1. **Regular languages and Automata theory**
   - Languages
   - Regular Languages
   - Regular expressions
   - Exercises
   - An overview of regular expressions
   - Regular expressions in Python

Agenda

1. **Regular languages and Automata theory**
   - Languages
   - Regular Languages
   - Regular expressions
   - Exercises
   - An overview of regular expressions
   - Regular expressions in Python

Languages

What is a language?

Languages

What is a language?

- Is a **set** of strings formed with the symbols in a particular alphabet.
- Languages are denoted with capital letters.
- As sets, languages can be finite or infinite.
- there's two types of language operations
  - Set (also called boolean)
  - Linguistic (power, reverse, concat, closure)

Activity

Given the languages $L_1 = \{\lambda, 0, 10, 11\}$ and $L_2 = \{\lambda, 1, 01, 11\}$ over the alphabet $\Sigma = \{0, 1\}$, obtain: $L_1 \cdot L_2$, $L_1 \cup L_2$. $L_1 \cap L_2$, $L_1 - L_2$, L_1^2 y L_2^R.

Languages

What is the power of a language?

Languages

What is the power of a language?

Definition

Given a language A over Σ, ($A \subseteq \Sigma^*$), *and a natural numberl* $n \in \mathbb{N}$, A^n *is defined in the following way:*

Languages

What is the power of a language?

Definition

Given a language A over Σ, ($A \subseteq \Sigma^*$), *and a natural numberl* $n \in \mathbb{N}$, A^n *is defined in the following way:*

$$\begin{aligned} A^0 &= \{\lambda\}, \\ A^n &= \underbrace{AA\cdots A}_{n \text{ times}} = \{u_1 \cdots u_n \mid u_i \in A, \text{ for all } i,\ 1 \leq i \leq n\}. \end{aligned}$$

Languages

What is the power of a language?

Definition

Given a language A over Σ, ($A \subseteq \Sigma^*$), *and a natural numberl* $n \in \mathbb{N}$, A^n *is defined in the following way:*

$$\begin{aligned} A^0 &= \{\lambda\}, \\ A^n &= \underbrace{AA\cdots A}_{n \text{ times}} = \{u_1 \cdots u_n \mid u_i \in A, \text{ for all } i,\ 1 \leq i \leq n\}. \end{aligned}$$

A^2 is the set of double string concatenations of A, A^3 the set of triple string concatenations of A and in general A^n is the set of n string concatenations of A.

Languages

What is the Kleene closure of a language?

Languages

What is the Kleene closure of a language?

Definition

The Kleene closure of A, $A \subseteq \Sigma^*$, is the union of all powers of A and it is represented by A^*.

What is the Kleene closure of a language?

Definition

The Kleene closure of A, $A \subseteq \Sigma^*$, is the union of all powers of A and it is represented by A^*.

$$\begin{aligned} A^* &= \textit{set of all string concatenations of } A, \\ &\quad \textit{including } \lambda \\ &= \{u_1, \cdots u_n \mid u_i \in A,\ n \geq 0\} \end{aligned}$$

Languages

What is the positive closure of a language?

Languages

What is the positive closure of a language?

Definition

The positive closure of a language A, $A \subseteq \Sigma^*$, is the union of all powers of A, without including λ and is represented by A^+.

What is the positive closure of a language?

Definition

The positive closure of a language A, $A \subseteq \Sigma^*$, is the union of all powers of A, without including λ and is represented by A^+.

$$\begin{aligned} A^+ &= \text{set of all string concatenations of } A, \\ &\quad \text{without including } \lambda \\ &= \{u_1, \cdots u_n \mid u_i \in A,\ n \geq 1\} \end{aligned}$$

Languages

What is the inverse of a language?

Languages

What is the inverse of a language?

Definition

Given A, a language over Σ, we define A^R in the following way:

$$A^R = \{u^R \mid u \in A\}$$

What is the inverse of a language?

Definition

Given A, a language over Σ, we define A^R in the following way:

$$A^R = \{u^R \mid u \in A\}$$

A^R is the inverse of A.

Agenda

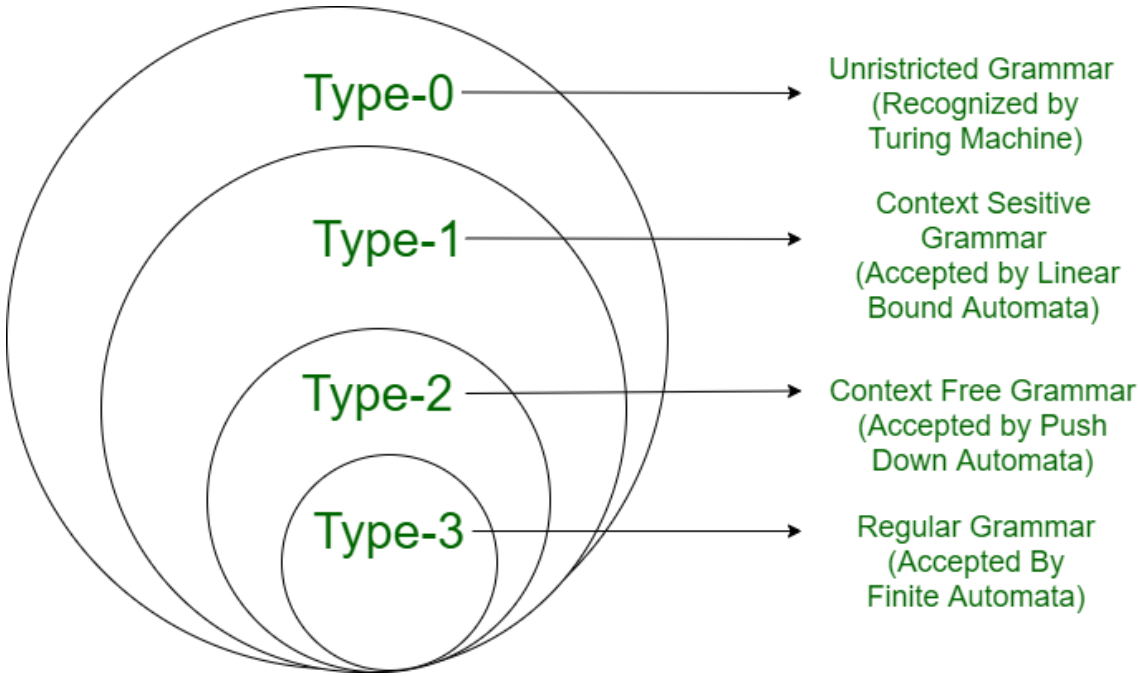
1. **Regular languages and Automata theory**
   - Languages
   - Regular Languages
   - Regular expressions
   - Exercises
   - An overview of regular expressions
   - Regular expressions in Python

Languages

What are regular languages?

Languages

What are regular languages?



Languages

What are regular languages useful for?

Languages

What are regular languages useful for?

- Lexical analysis (Scanning): In compiler construction, regular languages are used to define the tokens of a programming language, and lexical analyzers (scanners) are built to recognize these tokens using regular expressions.
- Language engineering: building tools for developers (compilers, IDEs, etc). For instance, regular expressions are applied to perform syntax highlighting, making it easier for programmers to read and understand code.

Languages

Formally, What are regular languages?

Languages

Formally, What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Languages

Formally, What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Languages

Formally, What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*

Languages

Formally, What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*
2. *If A and B are regular languages over* Σ, *then the following are also regular:*

Languages

Formally, What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*
2. *If A and B are regular languages over* Σ, *then the following are also regular:*

$$A \cup B \quad \textit{(union)},$$

Languages

Formally, What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*
2. *If A and B are regular languages over* Σ, *then the following are also regular:*

$$\begin{array}{ll} A \cup B & \textit{(union)}, \\ A \cdot B & \textit{(concatenation)}, \end{array}$$

Languages

Formally, What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*
2. *If* A *and* B *are regular languages over* Σ, *then the following are also regular:*

$$\begin{array}{ll} A \cup B & \textit{(union)}, \\ A \cdot B & \textit{(concatenation)}, \\ A^* & \textit{(Kleene closure)}. \end{array}$$

Languages

What are regular languages?

Languages

What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Languages

What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Languages

What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*

Languages

What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*
2. *If A and B are regular languages over* Σ, *then the following are also regular:*

Languages

What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*
2. *If A and B are regular languages over* Σ, *then the following are also regular:*

$$A \cup B \quad \text{(union)},$$

Languages

What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*
2. *If* A *and* B *are regular languages over* Σ, *then the following are also regular:*

$$\begin{array}{ll} A \cup B & \textit{(union)}, \\ A \cdot B & \textit{(concatenation)}, \end{array}$$

Languages

What are regular languages?

Regular languages from a given alphabet Σ are all those languages that can be built from the basic languages $\emptyset, \{\lambda\}, \{a\}, a \in \Sigma$, using union, concatenation and Kleene closure.

Given an alphabet Σ

Definition

1. $\emptyset, \{\lambda\}, \{a\}$, *for all* $a \in \Sigma$, *are regular languages over* Σ. *These are the so called basic regular languages.*
2. *If* A *and* B *are regular languages over* Σ, *then the following are also regular:*

$$\begin{array}{ll} A \cup B & \textit{(union)}, \\ A \cdot B & \textit{(concatenation)}, \\ A^* & \textit{(Kleene closure)}. \end{array}$$

Languages

Example

Given $\Sigma = \{a, b\}$. *The following ones are regular languages over* Σ :

Languages

Example

Given $\Sigma = \{a, b\}$. *The following ones are regular languages over* Σ :

1. *The language A of all strings that have just one a* :

$$A = \{b\}^* \cdot \{a\} \cdot \{b\}^*.$$

Languages

Example

Given $\Sigma = \{a, b\}$. *The following ones are regular languages over* Σ :

1. *The language A of all strings that have just one a* :

$$A = \{b\}^* \cdot \{a\} \cdot \{b\}^*.$$

2. *The language B of all strings that start with b* :

$$B = \{b\} \cdot \{a, b\}^*.$$

Languages

Example

Given $\Sigma = \{a, b\}$. The following ones are regular languages over Σ :

1. *The language A of all strings that have just one a :*

$$A = \{b\}^* \cdot \{a\} \cdot \{b\}^*.$$

2. *The language B of all strings that start with b :*

$$B = \{b\} \cdot \{a, b\}^*.$$

3. *The language C of all strings that contain the string ba :*

$$C = \{a, b\}^* \cdot \{ba\} \cdot \{a, b\}^*.$$

Languages

Example

Given $\Sigma = \{a, b\}$. *The following ones are regular languages over* Σ :

1. *The language A of all strings that have just one a* :
$$A = \{b\}^* \cdot \{a\} \cdot \{b\}^*.$$
2. *The language B of all strings that start with b* :
$$B = \{b\} \cdot \{a, b\}^*.$$
3. *The language C of all strings that contain the string ba* :
$$C = \{a, b\}^* \cdot \{ba\} \cdot \{a, b\}^*.$$
4. $(\{a\} \cup \{b\}^*) \cdot \{a\}$.

Languages

Example

Given $\Sigma = \{a, b\}$. *The following ones are regular languages over* Σ :

1. *The language A of all strings that have just one a* :
$$A = \{b\}^* \cdot \{a\} \cdot \{b\}^*.$$
2. *The language B of all strings that start with b* :
$$B = \{b\} \cdot \{a, b\}^*.$$
3. *The language C of all strings that contain the string ba* :
$$C = \{a, b\}^* \cdot \{ba\} \cdot \{a, b\}^*.$$
4. $(\{a\} \cup \{b\}^*) \cdot \{a\}$.
5. $[(\{a\}^* \cup \{b\}^*) \cdot \{b\}]^*$.

Agenda

1. **Regular languages and Automata theory**
   - Languages
   - Regular Languages
   - Regular expressions
   - Exercises
   - An overview of regular expressions
   - Regular expressions in Python

Languages

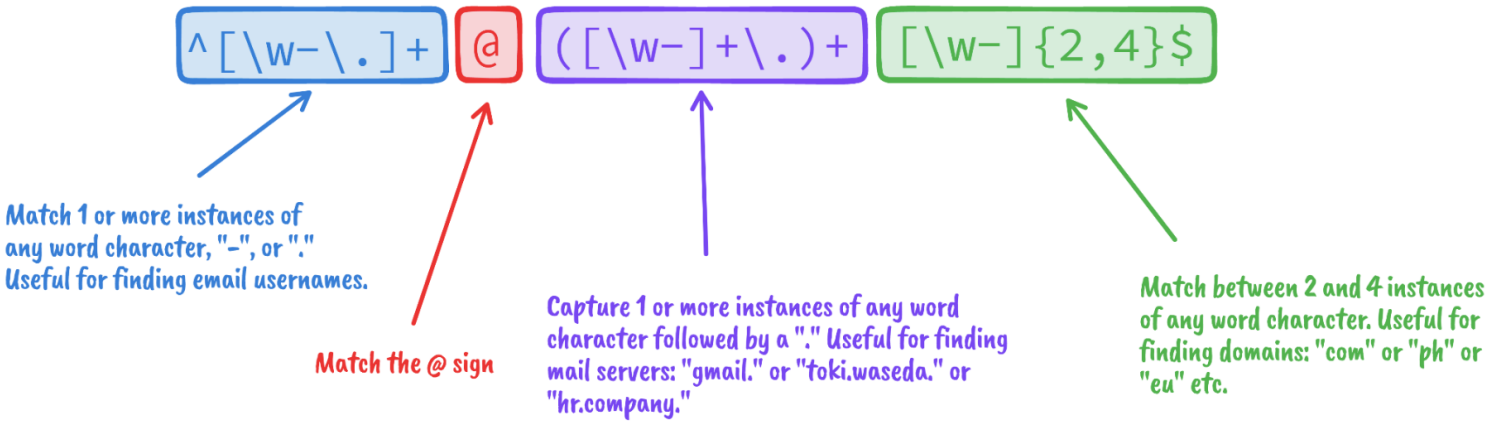
What are regular expressions?

Languages

What are regular expressions?
Regular expressions are text patterns that define the form a text string should have.

Languages

What are regular expressions?
Regular expressions are text patterns that define the form a text string should have.



Languages

What can we do with these regular expressions?

Languages

What can we do with these regular expressions?

- Check if an input honors a given pattern.
- Look for a pattern appearance in a piece of text.
- Extract specific portions of a text.
- Replace portions of text.
- Split a larger text into smaller pieces.

Languages

Formally, what are the regular expressions?

Languages

Formally, what are the regular expressions?

Definition

1. *Basic regular expressions:*

Formally, what are the regular expressions?

Definition

1. *Basic regular expressions:*

   $\emptyset$ *is a regular expression that represents the language* $\emptyset$

Formally, what are the regular expressions?

Definition

1. *Basic regular expressions:*

   $\emptyset$ *is a regular expression that represents the language* $\emptyset$
   λ *is a regular expression that represents the language* $\{\lambda\}$.

Formally, what are the regular expressions?

Definition

1. *Basic regular expressions:*

   $\emptyset$ *is a regular expression that represents the language* $\emptyset$
   λ *is a regular expression that represents the language* $\{\lambda\}$.
   a *is a regular expression that represents the language* $\{a\}$, $a \in \Sigma$.

Formally, what are the regular expressions?

Definition

1. *Basic regular expressions:*

   $\emptyset$ *is a regular expression that represents the language* $\emptyset$
   λ *is a regular expression that represents the language* $\{\lambda\}$.
   a *is a regular expression that represents the language* $\{a\}$, $a \in \Sigma$.

2. *If R and S are regular expressions over Σ, the following ones are also regular:*

Formally, what are the regular expressions?

Definition

1. *Basic regular expressions:*

   $\emptyset$ *is a regular expression that represents the language* $\emptyset$
   λ *is a regular expression that represents the language* $\{\lambda\}$.
   a *is a regular expression that represents the language* $\{a\}$, $a \in \Sigma$.

2. *If* R *and* S *are regular expressions over* Σ, *the following ones are also regular:*

$$(R)(S)$$

Formally, what are the regular expressions?

Definition

1. *Basic regular expressions:*

   $\emptyset$ *is a regular expression that represents the language* $\emptyset$
   λ *is a regular expression that represents the language* $\{\lambda\}$.
   a *is a regular expression that represents the language* $\{a\}$, $a \in \Sigma$.

2. *If* R *and* S *are regular expressions over* Σ, *the following ones are also regular:*

$$(R)(S)$$
$$(R \cup S)$$

Formally, what are the regular expressions?

Definition

1. *Basic regular expressions:*

   $\emptyset$ *is a regular expression that represents the language* $\emptyset$
   λ *is a regular expression that represents the language* $\{\lambda\}$.
   a *is a regular expression that represents the language* $\{a\}$, $a \in \Sigma$.

2. *If* R *and* S *are regular expressions over* Σ, *the following ones are also regular:*

$$(R)(S)$$
$$(R \cup S)$$
$$(R)^*$$

Languages

Example

Given the alphabet $\Sigma = \{a, b, c\}$,

Example

Given the alphabet $\Sigma = \{a, b, c\}$,

$$(a \cup b^*)a^*(bc)^*$$

Example

Given the alphabet $\Sigma = \{a, b, c\}$,

$$(a \cup b^*)a^*(bc)^*$$

is the regular expression that represents

Example

Given the alphabet $\Sigma = \{a, b, c\}$,

$$(a \cup b^*)a^*(bc)^*$$

is the regular expression that represents

$$(\{a\} \cup \{b\}^*) \cdot \{a\}^* \cdot \{bc\}^*$$

Activity

Give some examples of strings in the language defined by the regular expression

Agenda

1. **Regular languages and Automata theory**
   - Languages
   - Regular Languages
   - Regular expressions
   - Exercises
   - An overview of regular expressions
   - Regular expressions in Python

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1, 2\}$ *of all strings that start with* 2 *and end with* 1.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ of all strings that start with two consecutive ones.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ of all strings that have a number of zeros divisible by three.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ of all strings that have at least two consecutive ones.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ of all strings that have at most two zeros.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ of all strings that only have one occurrence of three consecutive zeros.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ of all strings that have length of five or less.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ *of all strings that start or end in* 00 *or* 11.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{a, b\}$ of all strings that have string ab an even number of times.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{a, b\}$ of all strings that have an even number of a's and an odd number of b's.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1, 2\}$ of all strings that do not have two consecutive ones.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ *of all strings that have at least one* 0 *and at least one* 1.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ of all strings that have at most two consecutive ones.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ *of all strings which fifth symbol, from left to right, is a* 1.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ *of all strings which length is* ≥ 4.

Exercises

Exercise

Find the regular expression for the language over $\Sigma = \{0, 1\}$ of all strings which do not have four consecutive zeros.

Agenda

1. **Regular languages and Automata theory**
   - Languages
   - Regular Languages
   - Regular expressions
   - Exercises
   - An overview of regular expressions
   - Regular expressions in Python

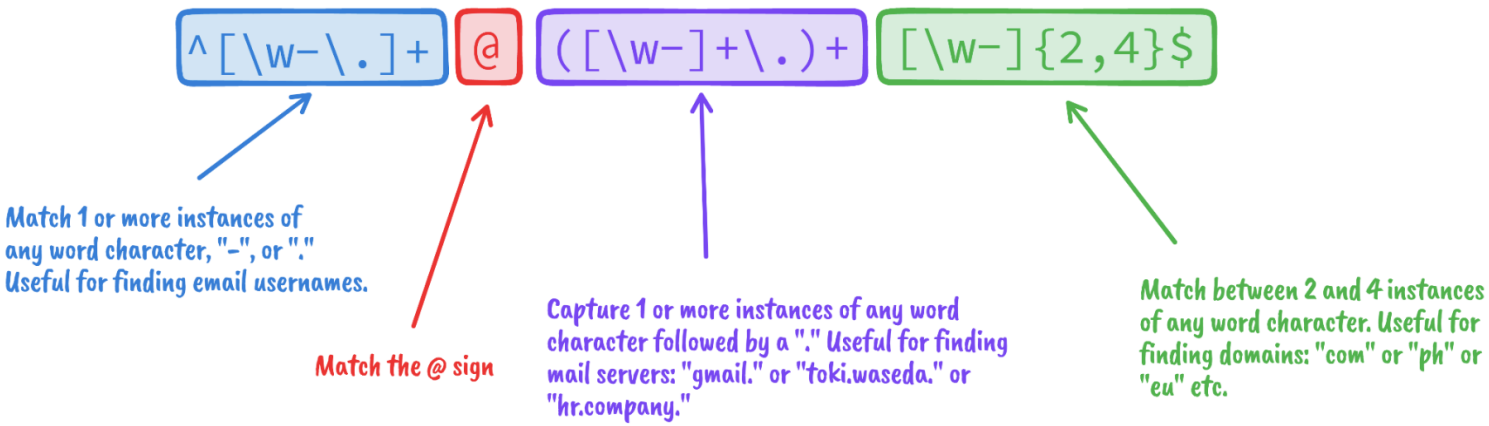
An overview of regular expressions

What else can we say about regular expressions?

An overview of regular expressions

What else can we say about regular expressions?

- A regular expression is a pattern of text that consists of ordinary characters **literals** and special characters known as **metacharacters**.



An overview of regular expressions

What are literals?

What are literals?

- Are the simplest form of pattern matching in regular expression.
- They will simply succeed whenever that literal is found.
- If we apply the regular expression **fox** to search the phrase:
  The quick brown fox jumps over the lazy dog, we will find one match.
- We can also obtain several results instead of just one, if we apply the regular expression `be` to the following phrase:
  To be, or not to be.

What are character classes?

What are character classes?

- The character classes (also known as character sets) allow us to define a character that will match if any of the defined characters on the set is present.
- To define a character class, we should use the opening square bracket metacharacter [, then any accepted characters, and finally close with a closing square bracket].
- e.g. `licen[cs]e`
- It is possible to also use the range of a character. This is done by leveraging the hyphen symbol (-) between two related characters.
- e.g. To match any lowercase letter we can use `[a-z]`.
- e.g. To match any single digit we can define the character set `[0-9]`.

How to work with several character classes?

How to work with several character classes?

- The character classes' ranges can be combined to be able to match a character against many ranges.

How to work with several character classes?

- The character classes' ranges can be combined to be able to match a character against many ranges.
- We just need to put one range after the other.

How to work with several character classes?

- The character classes' ranges can be combined to be able to match a character against many ranges.
- We just need to put one range after the other.
- e.g. If we want to match any lowercase or uppercase alphanumeric character, we can use `[0-9a-zA-Z]`

How to work with several character classes?

- The character classes' ranges can be combined to be able to match a character against many ranges.
- We just need to put one range after the other.
- e.g. If we want to match any lowercase or uppercase alphanumeric character, we can use `[0-9a-zA-Z]`
- This can be alternatively written using the union mechanism: `[0-9[a-z[A-Z]]]`.

What do we mean by negation of ranges?

What do we mean by negation of ranges?

- We can invert the meaning of a character set by placing a caret (^) symbol right after the opening square bracket metacharacter ([).

What do we mean by negation of ranges?

- We can invert the meaning of a character set by placing a caret (^) symbol right after the opening square bracket metacharacter ([).
- If we have a character class such as `[0-9]` meaning any digit, the negated character class `[^0-9]` will match anything that is not a digit.

An overview of regular expressions

Some character classes supported at this moment in Python

Some character classes supported at this moment in Python

- . This element matches any character except newline `\n`

An overview of regular expressions

Some character classes supported at this moment in Python

- `.` This element matches any character except newline `\n`
- `\d` This matches any decimal digit.

An overview of regular expressions

Some character classes supported at this moment in Python

- `.` This element matches any character except newline `\n`
- `\d` This matches any decimal digit.
- `\D` This matches any non-digit character.

An overview of regular expressions

Some character classes supported at this moment in Python

- `.` This element matches any character except newline `\n`
- `\d` This matches any decimal digit.
- `\D` This matches any non-digit character.
- `\s` This matches any whitespace character.

An overview of regular expressions

Some character classes supported at this moment in Python

- `.` This element matches any character except newline `\n`
- `\d` This matches any decimal digit.
- `\D` This matches any non-digit character.
- `\s` This matches any whitespace character.
- `\S` This matches any non whitespace character.

An overview of regular expressions

Some character classes supported at this moment in Python

- `.` This element matches any character except newline `\n`
- `\d` This matches any decimal digit.
- `\D` This matches any non-digit character.
- `\s` This matches any whitespace character.
- `\S` This matches any non whitespace character.
- `\w` This matches any alphanumeric character.

An overview of regular expressions

Some character classes supported at this moment in Python

- `.` This element matches any character except newline `\n`
- `\d` This matches any decimal digit.
- `\D` This matches any non-digit character.
- `\s` This matches any whitespace character.
- `\S` This matches any non whitespace character.
- `\w` This matches any alphanumeric character.
- `\W` This matches any non alphanumeric character.

An overview of regular expressions

How to match against a set of regular expressions?

An overview of regular expressions

How to match against a set of regular expressions?

- This is accomplished by using the pipe symbol `|`.

An overview of regular expressions

How to match against a set of regular expressions?

- This is accomplished by using the pipe symbol `|`.
- It alternates between regular expressions.

An overview of regular expressions

How to match against a set of regular expressions?

- This is accomplished by using the pipe symbol `|`.
- It alternates between regular expressions.
- e.g. `yes|no`

How to match against a set of regular expressions?

- This is accomplished by using the pipe symbol `|`.
- It alternates between regular expressions.
- e.g. `yes|no`
- e.g. `yes|no|maybe`

What are quantifiers?

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

- `?` 0 or 1 repetitions

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

- `?` 0 or 1 repetitions
- `*` 0 or more times

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

- `?` 0 or 1 repetitions
- `*` 0 or more times
- `+` 1 or more times

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

- `?` 0 or 1 repetitions
- `*` 0 or more times
- `+` 1 or more times
- `{n,m}` Between *n* and *m* times.

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

- `?` 0 or 1 repetitions
- `*` 0 or more times
- `+` 1 or more times
- `{n,m}` Between *n* and *m* times.
  - `{n}` Exactly *n* times.

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

- `?` 0 or 1 repetitions
- `*` 0 or more times
- `+` 1 or more times
- `{n,m}` Between *n* and *m* times.
  - `{n}` Exactly *n* times.
  - `{n,}` *n* or more times.

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

- `?` 0 or 1 repetitions
- `*` 0 or more times
- `+` 1 or more times
- `{n,m}` Between *n* and *m* times.
  - `{n}` Exactly *n* times.
  - `{n,}` *n* or more times.
  - `{,n}` At most *n* times.

What are quantifiers?

- The mechanisms to define how a character, metacharacter, or character set can be repeated.

Which are the three basic quantifiers?

- `?` 0 or 1 repetitions
- `*` 0 or more times
- `+` 1 or more times
- `{n,m}` Between *n* and *m* times.
  - `{n}` Exactly *n* times.
  - `{n,}` *n* or more times.
  - `{,n}` At most *n* times.
- e.g. `cars?` Singular or plural.

Example

How to match a telephone number that can be in the format 555-555-555, 555 555 555, or 555555555.

```
r'\b\d{3}[-\s]?\d{3}[-\s]?\d{3}\b'
```

What are boundary matchers?

- The boundary matchers are a number of identifiers that will correspond to a particular position inside of the input.
- `^` Matches at the beginning of a line
- `$` Matches at the end of a line
- `\b` Matches a word boundary
- `\B` The opposite of \b
- `\A` Matches the beginning of the input
- `\Z` Matches the end of the input

Example

Write a regular expression that will match lines that start with `Name:` *and make sure that after the name, there are only alphabetic characters or spaces until the end of the line.*

r'^Name:\s*[A-Za-z]*$'

Example

What is the difference between the regular expressions `hello` *and* `\bhello\b`*?*

- ‘hello’ is a pattern that matches the substring "hello" anywhere in the text, regardless of what characters come before or after it.
- r’\bhello\b’ uses the word boundary anchors \b to match the substring "hello" as a whole word, ensuring it is not part of another word.

Agenda

1. **Regular languages and Automata theory**
   - Languages
   - Regular Languages
   - Regular expressions
   - Exercises
   - An overview of regular expressions
   - Regular expressions in Python

Regular expressions in Python

How are regular expressions supported in Python?

- They are supported by the `re` module.
- We only need to import it to start using it.

How to start using them to match a pattern?

- We compile a pattern with `pattern = re.compile(r'foo')`
- We can try to match it against a string `pattern.match("foo bar")`

Regular expressions in Python

What are the building blocks for Python Regex?

- RegexObject
  - Also known as Pattern Object.
  - Represents a compiled regular expression.
- MatchObject
  - Represents the matched pattern.

What is a RegexObject?

- Before matching patterns we need to compile the regex.
- The compilation produces a reusable pattern object.
- This object provides all the operations that can be done (i.e. matching a pattern and finding all substrings that match a particular regex).
- e.g. `pattern = re.compile(r'<HTML>')`
- `pattern.match("<HTML>")`

Regular expressions in Python

What are two ways of matching a pattern?

- We can compile a pattern, which gives us a RegexObject.
- We can just use the module operations.
- If we compile a pattern we are able to reuse it.
- e.g. `pattern = re.compile(r'<HTML>')`
- `pattern.match("<HTML>")`
- e.g. `re.match(r'<HTML>', "<HTML>")`

How to search for string that match a pattern?

- In python we have two operations match and search.

How does match work?

- This method tries to match the compiled pattern only at the beginning of the string.
- If there is a match, then it returns a MatchObject.
- It has two optional parameters, pos and endpos.
- pos determines the position from where to search the pattern in the string.
- endpos determines the position until where the pattern is searched in the string.
- `pattern.match(string,pos,endpos)`

Example

Given `pattern = re.compile(r'^<HTML>')` *what is the result of:*

Example

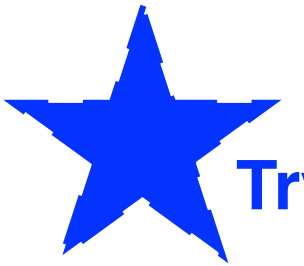
Given `pattern = re.compile(r'^<HTML>')` *what is the result of:*

- `pattern.match("<HTML>")`

Example

Given `pattern = re.compile(r'^<HTML>')` *what is the result of:*

- `pattern.match("<HTML>")`
- `pattern.match(" <HTML>")`



Try the following instructions and find the answer

Example

Given `pattern = re.compile(r'^<HTML>')` *what is the result of:*

- `pattern.match("<HTML>")`
- `pattern.match(" <HTML>")`
- `pattern.match(" <HTML>"[2:])`

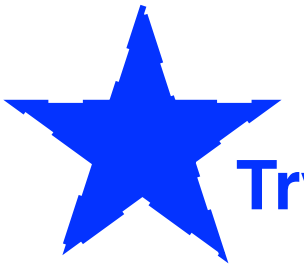
Example

Given `pattern = re.compile(r'<HTML>$')` *what is the result of:*

Example

Given `pattern = re.compile(r'<HTML>$')` *what is the result of:*

- `pattern.match("<HTML> ",0,6)`



Try the following instructions and find the answer

Example

Given `pattern = re.compile(r'<HTML>$')` *what is the result of:*

- `pattern.match("<HTML> ",0,6)`
- `pattern.match("<HTML> "[:6])`

Regular expressions in Python

How does search work?

How does search work?

- This operation would be like the match of many languages.
- It tries to match the pattern at any location of the string and not just at the beginning.
- If there is a match, then it returns a MatchObject.
- The pos and endpos parameters have the same meaning as that in the match operation.
- pos determines the position from where to search the pattern in the string.
- endpos determines the position until where the pattern is searched in the string.
- `pattern.match(string,pos,endpos)`

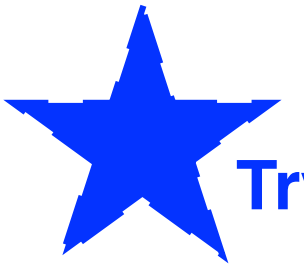
Example

Given `pattern = re.compile(r"world")` *what is the result of:*

Example

Given `pattern = re.compile(r"world")` *what is the result of:*

- `pattern.match("hello world")`



Try the following instructions and find the answer

Example

Given `pattern = re.compile(r"world")` *what is the result of:*

- `pattern.search("hello world")`
- `pattern.search("hola mundo ")`

Example

Given `pattern = re.compile(r'^<HTML>', re.MULTILINE)`
what is the result of:

Example

Given `pattern = re.compile(r'^<HTML>', re.MULTILINE)` *what is the result of:*

- `pattern.search("<HTML>")`

Example

Given `pattern = re.compile(r'^<HTML>', re.MULTILINE)` *what is the result of:*

- `pattern.search("<HTML>")`
- `pattern.search(" <HTML>")`

Example

Given `pattern = re.compile(r'^<HTML>', re.MULTILINE)` *what is the result of:*

- `pattern.search("<HTML>")`
- `pattern.search(" <HTML>")`
- `pattern.search(" \n<HTML>")`

Example

Given `pattern = re.compile(r'^<HTML>', re.MULTILINE)` *what is the result of:*

- `pattern.search("<HTML>")`
- `pattern.search(" <HTML>")`
- `pattern.search(" \n<HTML>")`
- `pattern.search(" \n<HTML>",3)`

Example

Given `pattern = re.compile(r'^<HTML>', re.MULTILINE)` *what is the result of:*

- `pattern.search("<HTML>")`
- `pattern.search(" <HTML>")`
- `pattern.search(" \n<HTML>")`
- `pattern.search(" \n<HTML>",3)`
- `pattern.search("</div></body>\n<HTML>",4)`

Try the following instructions and find the answer

Example

Given `pattern = re.compile(r'^<HTML>', re.MULTILINE)` *what is the result of:*

- `pattern.search("<HTML>")`
- `pattern.search(" <HTML>")`
- `pattern.search(" \n<HTML>")`
- `pattern.search(" \n<HTML>",3)`
- `pattern.search("</div></body>\n<HTML>",4)`
- `pattern.search(" \n<HTML>",4)`

How does findall work?

How does findall work?

- It returns a list with all the non-overlapping occurrences of a pattern and not the MatchObject like search and match do.

How does findall work?

- It returns a list with all the non-overlapping occurrences of a pattern and not the MatchObject like search and match do.
- e.g. `pattern = re.compile(r"\w+")`
- `pattern.findall("hello world")`
- `['hello', 'world']`

Regular expressions in Python

Example

- `pattern = re.compile(r"(\w+) (\w+)")`
- `pattern.findall("Hello world hola mundo")`
- `[('Hello', 'world'), ('hola', 'mundo')]`

How does finditer work?

How does finditer work?

- Works essentially as as findall.
- It returns an iterator in which each element is a MatchObject.
- We can use the operations provided by this object.
- Useful when you need information for every match.

Regular expressions in Python

Example

- `pattern = re.compile(r"(\w+) (\w+)")`
- `it = pattern.finditer("Hello world hola mundo")`
- `match = it.next()`
- `match.groups()`
- `('Hello','world')`
- `match.span()`
- `(0,11)`

Regular expressions in Python

Which are some operations to modify strings?

Which are some operations to modify strings?

- split(string, maxsplit=0): A string can be split based on the matches of the pattern.
- sub(repl, string, count=0): Returns the resulting string after replacing the matched pattern in the original string with the replacement.

Regular expressions in Python

Example

- `pattern = re.compile(r"\W")`
- `pattern.split("Beautiful is better than ugly", 2)`
- `['Beautiful', 'is', 'better than ugly']`

Example

- *pattern = re.compile(r'[0-9]+)*
- *pattern.sub("-","order0, order1 order13")*
- *'order- order- order-'*

What is a MatchObject?

What is a MatchObject?

- An object that represents the matched pattern.
- We will get one every time you execute one of these operations:
  - match
  - search
  - finditer
- Provides a set of operations for working with the captured groups.

Which are the main operations for a MatchObject?

Which are the main operations for a MatchObject?

- group
- groups
- groupdict
- start
- end
- span

What is the group operation?

What is the group operation?

- Gives the subgroups of the match.
- If invoked with no arguments or zero, returns the entire match.
- If one or more group identifiers are passed, the corresponding groups' matches will be returned.

Try the following instructions and find the answer

Example

- `pattern = re.compile("(\w+) (\w+)")`
- `match = pattern.search("Hello world")`
- `match.group()` → `'Hello world'`
- `match.group(0)` → `'Hello world'`
- `match.group(1)` → `'Hello'`
- `match.group(2)` → `'world'`
- `match.group(0,2)` → `('Hello world','world')`

Example

- *Groups can be named.*
- *If the pattern has named groups, they can be accessed using the names or the index:*
- `pattern = re.compile(r"(?P<first>\w+) (?P<second>\w+)")`
- `match = pattern.search("Hello world")`
- `match.group('first')` → `'Hello'`
- `match.group(1)` → `'Hello'`
- `match.group(0,'first',2)` → `('Hello world','Hello','world')`

What is the groups operation?

What is the groups operation?

- It returns a tuple with all the subgroups in the match instead of giving one or some of the groups.

Regular expressions in Python

Example

- `pattern = re.compile("(\w+) (\w+)")`
- `match = pattern.search("Hello world")`
- `match.groups()` → `('Hello', 'world')`

What is the groupdict operation?

What is the groupdict operation?

- It is used in the cases where named groups have been used.
- It will return a dictionary with all the groups that were found.
- `pattern = re.compile(r"(?P<first>\w+) (?P<second>\w+)")`
- `match = pattern.search("Hello world")`
- `{'first': 'Hello', 'second': 'world'}`
- If there aren't named groups, then it returns an empty dictionary.