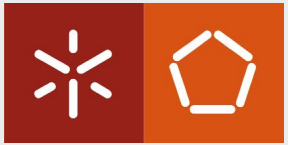
Processamento de Linguagem Natural

Luís Filipe Cunha
lfc@di.uminho.pt



Processamento de Linguagem Natural

“Natural language processing (NLP) is concerned with giving computer program the ability to understand human language as it is spoken and written -- referred to as natural language.”

What is Language

Language is a structured system of communication that uses **symbols** and **rules** to convey **meaning**.

Language enables:

- Communication
- Thought representation
- Knowledge transfer
- Coordination among individuals

Core Components:

- **Lexicon** – vocabulary (words, symbols, signals)
- **Syntax** – rules of structure
- **Semantics** – meaning
- **Pragmatics** – use in context

Language is a general concept — not limited to humans.

Types of Languages

- **Programming Languages** – A formal language designed to give instructions to computers.
  Example: *Python, Java, C++*

- **Formal Languages** – mathematics / logic
  Propositional logic, regular expressions, Mathematical formulas:

- **Natural Language**

What Is Natural Language?

A language that **evolved naturally among humans** for communication.

Examples:

- English
- Portuguese
- Mandarin
- Arabic
- Sign languages

Characteristics:

- Ambiguous
- Context-dependent
- Flexible
- Evolving over time
- Culturally shaped

Ambiguity in Natural Language

Natural language is inherently ambiguous.

Example:

> “I saw the man with the telescope.”

Possible interpretations:

- I used a telescope.
- The man had a telescope.

Ambiguity is a feature — not a flaw.

Types of ambiguity:

- **Lexical** (word meaning)
  - “bank” → riverbank / financial institution
- **Syntactic** (structure)
  - modifies the VP vs NP
- **Semantic** (interpretation)
  - “Every student read a book”: same vs different book
- **Pragmatic** (context-driven)
  - “Wow, right on time” → sarcasm

Why Language Matters

Modern intelligence is not just individual intelligence.

- Human intelligence is networked intelligence.
- Language connects human brains.
- Language enabled cultural evolution.
- Writing enabled knowledge accumulation across time.

Without language:

- No large-scale cooperation
- No complex societies
- No modern technology

Language is foundational to human societal intelligence.

Evolutionary Perspective

- Primates diverged ~65 million years ago.
- Humans and chimpanzees split ~6 million years ago.
- Human language likely emerged only a few hundred thousand years ago.
- Writing appeared ~5,000 years ago.

Language is evolutionarily recent, but civilization-scale transformative

Human Language Understanding & Reasoning | Daedalus | MIT Press

The Beginning of NLP

Early Beginnings

- **1950** - Alan Turing proposes the **Turing Test** in *“Computing Machinery and Intelligence”*.
- **1954** - **Georgetown-IBM experiment**: first demonstration of **machine translation** (Russian → English).
  - **Goal:** Use computers to process and understand human language
  - **Limitations:** Very little knowledge of grammar or semantics; mostly **word-for-word translation**.
  -
- **1966–67** - **ELIZA**: Early chatbot using pattern-matching; simulated conversation with a psychotherapist
- **1970s -** LUNAR: NLP application designed to allow scientists to query data about moon rocks returned by the Apollo missions

The Four Eras of NLP

1. **1950–1969** → Early Machine Translation
   a. It was imagined that translation could quickly build on the great successes of computers in code breaking during World War II.

2. **1970–1992** → Rule-Based Systems
   a. they started to model and use some of the complexity of human language understanding, such as syntax structure.

3. **1993–2012** → Statistical NLP
   a. digital text became abundantly available, and the compelling direction was to develop algorithms that could achieve some level of language understanding

4. **2013–Present** → Neural & Foundation Models
   a. introduction of deep learning. Words and sentences are represented by a position in a (several hundred- or thousand-dimensional) real- valued vector space, and similarities of meaning or syntax are represented by proximity in this space.

Each era reflects:

- Available data
- Available computation
- Available theory

Human Language Understanding & Reasoning | Daedalus | MIT Press

Slide 6 — Era 1 (1950–1969): Early MT

Characteristics:

- Very limited computation
- Very little data
- Almost no linguistic knowledge

Systems:

- Word-to-word translation lookup
- Simple morphological rules

Result:

- Overpromised
- Underperformed

Plano

- Introdução ao Python
- Unix Filters
- Expressões Regulares
- Corpora
- Gramática do Português
- Terminologia Dicionários e enciclopédias
- Word Embeddings
- Web Scraping
- Domain Specific Language
- NER / PoS / Q&A …

Ferramentas

- Lark Parser
- spaCy
- Stanza
- NLTK
- Flask
- BS4
- Selenium
- Gensim
- FastText
- HuggingFace
- Terminal

Ficheiros

- html
- xml
- json
- yaml
- latex
- tmx
- pdf
- texto

Aulas

Ambiente de trabalho:

- Unix/Windows
- Python > 3.10
- Editor de texto
  - VS CODE
    - Ctrl + B
    - Theme
    - Ctrl + (+/-)
    - Alt + Z

GitHub:

- https://github.com/lfcc1/plneb-2425
- Bibliografia
- Aulas
- Data
- Slides
- Tpcs

Avaliação

- Testes
- Trabalhos práticos
- TPCs

- Teste - 40%
- Trabalhos práticos - 40%
- Trabalhos de casa - 20%
- Nota mínima 8

Trabalho obrigatório!

Datas: ...

TPC

- Criação de repositório Github: plneb-2526
- Criação de diretoria para cada TPC
  - TPC1
  - TPC2
  - ...
- Descrição para cada TPC em Markdown (Readme.md)
  - Basic Syntax | Markdown Guide

Github

Installation:

ubuntu:

```
sudo apt update
sudo apt upgrade
sudo apt install git
```

windows:
https://gitforwindows.org/

Setup:

```
echo "# Repositório PLNEB-2526" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/lfcc1/plneb-2526.git
git push -u origin master
```

Introdução ao Python

```
#!/usr/bin/env python3

print("Hello World")

editor = input("What is your favorite text editor? ")

a = [3, 6, 7, 2, 5, 9, 4, 0, 10]

for i in a:
  if i < 5:
    print(str(i) + " is lower than 5")
  elif i > 5:
    print(str(i) + " is higher than 5")
  else:
    print("FIVE!")

# comment
```

- Listas
- Dicionários
- Ficheiros
- Exercícios

Strings

Creating a String: "..." ou '...';
multiline String: """...""" ou '''...''';

```
string1 = "hello!"
string2 = 'hello'
multilineString1 = """Hello everyone!
I'm a multiline string.
"""

multilineString2 = '''Hello guys!
I'm also a multiline string!'''

string2 = 'I'm also a multiline string!'
```

Strings

* Comprimento duma string:
`...len(frase)...`;
* Verificar se a frase contem a palavra “dia” uma parte:
`if "dia" in frase...`;
* Verificar se não contém a palavra “dia”:
`if "dia" not in frase...`;
* Conversão para maiúsculas:
`frase.upper()`;
* Conversão para minúsculas:
`frase.lower()`;
* Remover espaço branco do início e do fim:
`frase.strip()`;
frase.lstrip() frase.rstrip()
* Substituir a palavra “Hoje” por “Today”:
`frase.replace("Hoje", "Today")`;
* Partir uma string em bocados usando um separador:
`frase.split("#")`;
* Concatenação de 2 strings:
`s1 + s2`.

String Slicing

* Slicing a string: `frase[start:stop:step]`; # start through not past stop, by step
  - from the start: `frase[:stop]`;
  - until the end: `frase[start:]`;
  - negative indexes.
  - amount by which the index increases `frase[::step]`

```
text = "hello world"
slice = text[1:5]
slice = text[:5]
slice = text[5:]
slice = text[5:-1]
slice = text[-5:]
slice = text[:-2]
slice = text[::2]
slice = text[::-1]
```

Listas

```
listaA = [1,2,3]
listaB = [1,2,3,"Joana","Bruno","Filipe"]
lista = ["Joana","Bruno","Filipe"]

lista[2]
# "Filipe"
"Susana" in lista
# False
lista.append("Maria")
# ['Joana', 'Bruno', 'Filipe', 'Maria']
lista.insert(2,"Pedro")
# ['Joana', 'Bruno', 'Pedro', 'Filipe',
'Maria']
lista[3] = "João"
# ['Joana', 'Bruno', 'Pedro', 'João', 'Maria']
concat = [1,2,3] + [4,5,6]
# [1, 2, 3, 4, 5, 6]
```

Listas

```
lista = list(range(0,10))
#[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
lista.pop(3)
# [0, 1, 2, 4, 5, 6, 7, 8, 9]
lista.pop(-3)
# [0, 1, 2, 4, 5, 6, 8, 9]
lista.remove(5)
# [0, 1, 2, 4, 6, 8, 9]
```

```
dict = {}
colors = {"fcp": "blue", "slb": "red", "scp": "green"}
```

Dicionários

```
len(colors)
# 3
"slb" in colors
# True
"fcb" in colors
# False
colors["fcp"]
# blue
colors["aca"]
#KeyError: 'aca'
colors.get("aca")
# None
```

```
colors.keys()
# ['fcp', 'slb', 'scp']
colors.values()
# ['blue', 'red', 'green']
colors.items()
#[('fcp', 'blue'), ('slb', 'red'), ('scp', 'green')]

colors | {"aca": "black","fcp": "blue and white"}
```

Listas por compreensão

```
newlist = [expression for item in iterable if condition == True]

lista = list(range(1,10))
# [1, 2, 3, 4, 5, 6, 7, 8, 9]
quadrados = [x * x for x in lista]
# [1, 4, 9, 16, 25, 36, 49, 64, 81]
Gera a lista [2, 4, 6, 8] usando listas em compreensão:
pares = [x for x in lista if x % 2 == 0 ]


fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
Gera a lista das palavras com a letra “a” usando listas em compreensão:
frutas_a = [fruta for fruta in fruits if “a” in fruta ]
```

Ordenação

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
Ordena a lista alfabeticamente: #['banana', 'kiwi', 'mango', 'orange', 'pineapple']
thislist.sort()
res = sorted(thislist)
Ordena a lista alfabeticamente por ordem inversa: #['pineapple', 'orange', 'mango', 'kiwi', 'banana']
thislist.sort(reverse = True)
res = sorted(thislist, reverse = True)

frutas = {"orange":12, "mango":10, "kiwi":43, "pineapple":11, "banana":20}
Qual o resultado de: res = sorted(frutas)
#['banana', 'kiwi', 'mango', 'orange', 'pineapple']
Ordena os elementos do dicionário de frutas por quantidade
#[('mango', 10), ('pineapple', 11), ('orange', 12), ('banana', 20), ('kiwi', 43)]
res = sorted(frutas.items(), key=ordena)
```

```
def ordena(e):
    return e[1]
```

Ficheiros

```
file = open('data/exemplo', "r")
lines = file.readlines()

file1 = open('data/exemplo', "r")
string = file1.read()

file2 = open('data/exemplo', "r")
line = file2.readline(10)
```

```
['Ola! Bem vindo a Scripting no Processamento de Linguagem Natural \n', 'Este
ficheiro é apenas um exemplo.\n', 'Boa sorte!']
---
Ola! Bem vindo a Scripting no Processamento de Linguagem Natural
Este ficheiro é apenas um exemplo.
Boa sorte!
---
Ola! Bem v
```

Exercícios

1. Programa que pergunta ao utilizador o nome e imprime em maiúsculas.
2. Função que recebe array de números e imprime números pares.
3. Função que recebe nome de ficheiro e imprime linhas do ficheiro em ordem inversa.
4. Função que recebe nome de ficheiro e imprime número de ocorrências das 10 palavras mais frequentes no ficheiro.
5. Função que recebe um texto como argumento e o ”limpa”: separa palavras e pontuação com espaços, converte para minúsculas, remove acentuação de caracteres, etc.

Exercícios - TPC

Create a function that:

1. given a string “s”, reverse it.
2. given a string “s”, returns how many “a” and “A” characters are present in it.
3. given a string “s”, returns the number of vowels there are present in it.
4. given a string “s”, convert it into lowercase.
5. given a string “s”, convert it into uppercase.
6. Verifica se uma string é capicua
7. Verifica se duas strings estão balanceadas (Duas strings, s1 e s2, estão balanceadas se todos os caracteres de s1 estão presentes em s2.)
8. Calcula o número de ocorrências de s1 em s2
9. Verifica se s1 é anagrama de s2.
    - "listen" e "silent": Deve imprimir True
    - "hello", "world": Deve imprimir False

ripgrep

“A line-oriented search tool that recursively searches the current directory for a regex pattern”

“ripgrep is a command line tool that searches your files for patterns that you give it. It behaves as if reading each file line by line. If a line matches the pattern provided to ripgrep, then that line will be printed.”

Install ripgrep (windows)

```
> $PSVersionTable
```

```
> winget install pwsh
```

```
> winget search ripgrep
```

```
Name          Id                          Version Match        Source
-------------------------------------------------------------------
ugrep         Genivia.ugrep               7.2.2   Tag: ripgrep winget
RipGrep GNU   BurntSushi.ripgrep.GNU      14.1.1               winget
RipGrep MSVC  BurntSushi.ripgrep.MSVC     14.1.1               winget
```

```
> winget install BurntSushi.ripgrep.GNU  14.1.1
```

Install poppler (windows)

https://scoop.sh

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
```

```
> scoop search poppler
```

```
Results from local buckets...

Name    Version   Source Binaries
----    -------   ------ --------
poppler 24.08.0-0 main
```

```
> scoop install poppler
```

Exercises

Tendo como base o livro “Harry Potter e a Pedra Filosofal”, use o comando **ripgrep** para resolver os seguintes desafios:

1. Procure ocorrências da palavra “magia”;
2. Conte o número de ocorrências da palavra “varinha”;
3. Conte o número de ocorrências das palavras “Magia” e “magia”;
4. Imprima duas linhas de texto à volta de cada ocorrência da expressão “Pedra Filosofal”;
5. Procure ocorrências de "Voldemort" e os seus pseudónimos comuns ("Você-Sabe-Quem", "Lorde das Trevas", etc);
6. Procure momentos principais do enredo utilizando palavras-chave. Por exemplo: “batalha”, “morte”, “vitória”. Deve considerar um contexto de 2 linhas de texto antes e depois de cada palavra-chave;
7. Liste os capítulos do livro (número e designação);
8. Encontre as linhas onde o “Harry” e a “Hermione” são ambos mencionados.

Processamento de Linguagem Natural Engenharia Biomédica

Luís Filipe Cunha
lfc@di.uminho.pt