

# CHATBOT Z UŻYCIEM PYTHONA I PROLOGA

Michał Bąk  
Jakub Burek  
Adrian Chołody

Cracow University of Technology,  
Computer Science

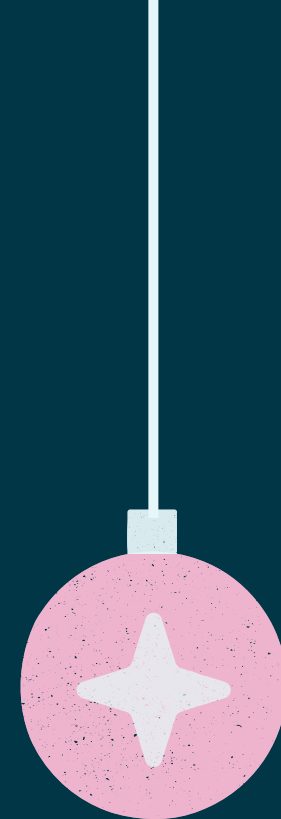
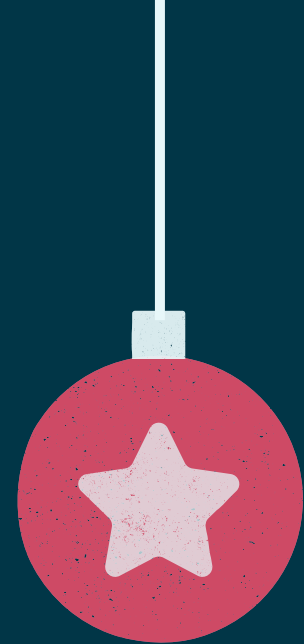


# AGENDA



- 01 Wprowadzenie
- 02 Analiza problemu i plan działania
- 03 Baza relacji w Prologu
- 04 Integracja Prologa z Pythonem
- 05 Tworzenie wzorców pytań w Pythonie
- 06 Testowanie i wyniki
- 07 Podsumowanie
- 08 Bibliografia





# WPROWADZENIE





# ANALIZA ZADANIA

## CEL:

Zbudowanie bazy relacji w Prologu i wykorzystanie jej w systemie pytania-odpowiedzi (*chatbot*).



# ETAPY REALIZACJI

**ANALIZA  
PROBLEMU**



**BAZA  
RELACJI  
W  
PROLOGU**

**INTEGRACJA  
PYTHONA  
I  
PROLOGA**

**KONSTRUKCJA  
WZORCÓW  
PYTAŃ**

**TESTOWANIE  
I POPRAWA**







# BAZA RALACJI W PROLOGU





# BAZA RELACJI W PROLOGU

## Przykładowa relacja

```
sister(X, Y) - X is a sister of Y
```

```
sister(X, Y) :- female(X), parents(X, Z, W), parents(Y, Z, W), X \= Y.
```



# BAZA RELACJI W PROLOGU

```
% =====
%           male(X) - X is a man
% =====

male(adam).
male(stefan).
male(staszek).
male(marek).
male(pawel).
male(witek).
male(kamil).
male(krzysztof).
male(jakub).
male(michal).
male(adrian).

% =====
%           female(Y) - Y is a woman
% =====

female(ala).
female(alina).
female(maria).
female(ania).
female(magda).
female(justyna).
female(wioletta).
female(martyna).
female(barbara).
female(katarzyna).
female(asia).
female(karolina).

% =====
%           parents(X, Y, Z) - X is a child of Y (father) and Z (mother)
% =====

parents(stefan, staszek, maria).
parents(ala, staszek, maria).
parents(marek, stefan, magda).
parents(witek, krzysztof, ala).
parents(ania, marek, alina).
parents(adrian, pawel, wioletta).
parents(michal, pawel, wioletta).
parents(jakub, pawel, wioletta).
parents(kamil, adrian, karolina).
```



# BAZA RELACJI W PROLOGU

```
% =====  
%           sister(X, Y) - X is a sister of Y  
% =====  
sister(X, Y) :- female(X), parents(X, Z, W), parents(Y, Z, W), X \= Y.  
  
% =====  
%           brother(X, Y) - X is a brother of Y  
% =====  
brother(X, Y) :- male(X), parents(X, Z, W), parents(Y, Z, W), X \= Y.  
|  
% =====  
%           parent(X, Y) - Y is a parent of X  
% =====  
parent(X, Y) :- parents(X, Y, _) ; parents(X, _, Y).  
  
% =====  
%           father(X, Y) - Y is the father of X  
% =====  
father(X, Y) :- parent(X, Y), male(Y).
```

# BAZA RELACJI W PROLOGU

```
% =====  
%           mother(X, Y) - Y is the mother of X  
% =====  
mother(X, Y) :- parent(X, Y), female(Y).  
  
% =====  
%           grandparent(X, Y) - Y is a grandparent of X  
% =====  
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).  
  
% =====  
%           grandmother(X, Y) - Y is the grandmother of X  
% =====  
grandmother(X, Y) :- parent(X, Z), parent(Z, Y), female(Y).  
  
% =====  
%           grandfather(X, Y) - Y is the grandfather of X  
% =====  
grandfather(X, Y) :- parent(X, Z), parent(Z, Y), male(Y).  
  
% =====  
%           uncle(X, Y) - Y is an uncle of X  
% =====  
uncle(X, Y) :- parent(X, Z), siblings(Y, Z), male(Y), Y \= Z.
```

# BAZA RELACJI W PROLOGU

```
% =====  
%          aunt(X, Y) - Y is an aunt of X  
% =====  
aunt(X, Y) :- parent(X, Z), siblings(Y, Z), female(Y), Y \= Z.  
  
% =====  
%          cousins(X, Y) - X and Y are cousins  
% =====  
cousins(X, Y) :- parent(X, Z), parent(Y, W), siblings(Z, W).
```



INTEGRACJA  
PROLOGA Z  
PYTHONEM



# DLACZEGO PYSWIP?

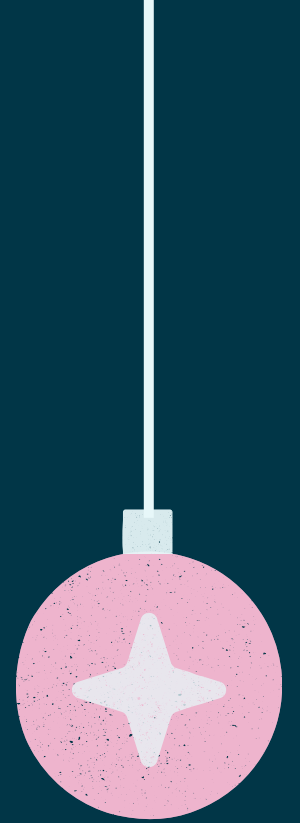
```
from pyswip import Prolog  
  
prolog = Prolog()  
prolog.consult("parents.pl")
```

Zdecydowaliśmy się na użycie **pyswip** do integracji, ponieważ umożliwia proste załadowanie bazy relacji i wykonywanie zapytań z poziomu Pythona.

Biblioteka pozwala dynamicznie mapować pytania w języku naturalnym na zapytania Prolog.



# WZORCE PYTAŃ W PYTHONIE



## Przykładowy wzorzec

```
r"who are the cousins of (\w+)": lambda name: f"cousins({name.lower()}, X)."
```

## Generowanie zapytania do prologa

```
prolog_query = query_func(*match.groups())
```

Wzorce w Pythonie, które identyfikują kluczowe elementy pytania (*relację*).

Na tej podstawie generowane jest zapytanie Prolog w odpowiednim formacie, które można wykonać na bazie relacji.



## Wzorce zapytań

```
query_patterns = {  
    r"who is the father of (\w+)": lambda name: f"father({name.lower()}, X).",  
    r"who is the mother of (\w+)": lambda name: f"mother({name.lower()}, X).",  
    r"who are the siblings of (\w+)": lambda name: f"siblings(X, {name.lower()}).",  
    r"does (\w+) have a sister": lambda name: f"sister(_, {name.lower()}).",  
    r"does (\w+) have a brother": lambda name: f"brother(_, {name.lower()}).",  
    r"who are the cousins of (\w+)": lambda name: f"cousins({name.lower()}, X).",  
    r"is (\w+) the uncle of (\w+)": lambda person1, person2: f"uncle({person2.lower()}, {person1.lower()}).",  
    r"who is the aunt of (\w+)": lambda name: f"aunt({name.lower()}, X).",  
    r"who is the grandfather of (\w+)": lambda name: f"grandfather({name.lower()}, X).",  
    r"who are the parents of (\w+)": lambda name: f"parent({name.lower()}, X)."  
}
```



# CHATBOT



## Czyszczenie danych wejściowych

```
def sanitize_input(input_text):  
    return input_text.strip().lower().replace("?", "")
```

## Główny kod Chatbota

```
print("Chatbox is ready! Ask me about relationships.")  
while True:  
    user_input = input("You: ")  
    if user_input.lower() in ["exit", "quit"]:  
        print("Goodbye!")  
        break  
    response = parse_and_query(user_input)  
    print(f"Chatbox: {response}")
```

# Mapowanie pytań i wykonywanie zapytań

```
def parse_and_query(user_input):
    user_input = sanitize_input(user_input)

    for pattern, query_func in query_patterns.items():
        match = re.match(pattern, user_input)
        if match:
            prolog_query = query_func(*match.groups())

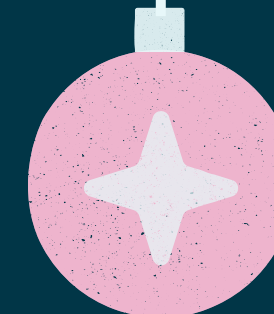
            try:
                results = list(prolog.query(prolog_query))
            except Exception as e:
                return f"Error: {e}"

            if pattern.startswith(r"does"):
                return "Yes" if results else "No"

            if pattern.startswith(r"is"):
                return "Yes" if results else "No"

            if results:
                return ", ".join(result['X'] for result in results if 'X' in result)
            else:
                return "No results found."

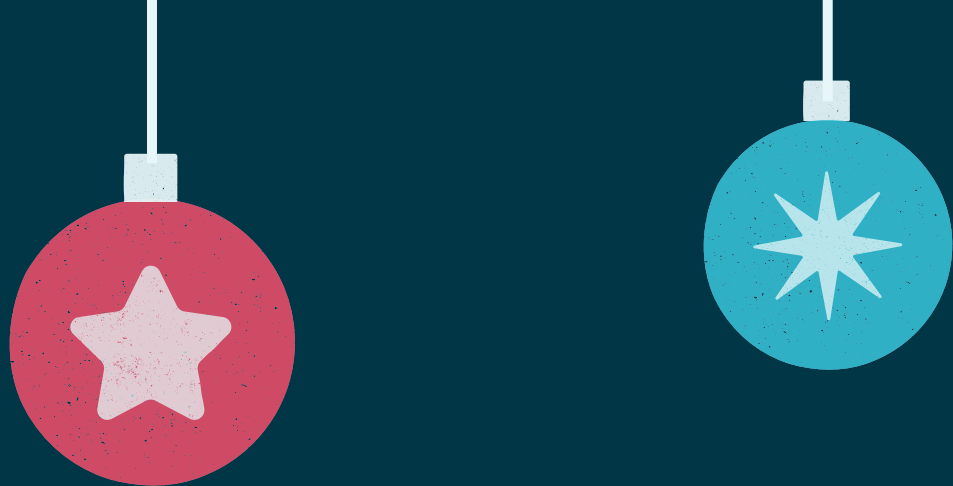
    return "I don't understand the question."
```



# TESTOWANIE







# TESTY

- sprawdzenie poprawności działania systemu
- testowane były różne typy pytań
- błędna odpowiedź -> poprawa zapytań

```
Chatbox is ready! Ask me about relationships.  
You: Who is the father of Stefan?  
Chatbox: staszek  
You: Who is the mother of Ala?  
Chatbox: maria  
You: Who are the siblings of Adrian?  
Chatbox: michał, jakub  
You: Does Adrian have a sister?  
Chatbox: No  
You: Does Marek have a brother?  
Chatbox: No  
You: Is Marek the uncle of Ania?  
Chatbox: Yes  
You: exit  
Goodbye!
```



# PODSUMOWANIE



# KLUCZOWE WNIOSKI

- System poprawnie odpowiada na większość pytań dzięki spójności bazy relacji i wzorców w Pythonie.
- Integracja Prologu z Pythonem umożliwiła efektywne połączenie logiki z przetwarzaniem języka naturalnego.
- Testowanie ujawniło, że precyzyjne wzorce w Pythonie są kluczowe dla prawidłowego mapowania pytań na zapytania Prolog.
- Projekt może być rozwijany o bardziej złożone pytania i większą bazę relacji, aby obsługiwać bardziej zróżnicowane scenariusze.

# BIBLIOGRAFIA

## **Prolog**

<https://www.openbookproject.net/py4fun/prolog/prolog1.html>

## **Biblioteka pyswip**

<https://github.com/yuce/pyswip>

## **Film instruktażowy: Łączenie Pythona z Prologiem**

<https://www.youtube.com/watch?v=1jwAHlz8WXc>





**MASZ PYTANIE?  
PYTAJ!**





# DZIEKUJEMY ZA UWAGĘ!

