## Запуск мінімальної екпертної системи[¶](#Запуск-мінімальної-екпертної-системи)

В консолі CLIPS уже можна добавляти правила, факти та виконувати запуск. При використані experta потрібно створити об'єкт який буде симулювати в собі CLIPS

In [1]:

from experta import KnowledgeEngine

class BasicExample(KnowledgeEngine):

pass

engine = BasicExample()

В середині об'єкта створюються початкові факти, правила і функції, а запуск еспертної системи відбувається як операція над екземпляром об'єкта

Запуск ЕС у середовищі CLIPS:

CLIPS

(reset)

(run)

Запуск ЕС у Python:

In [2]:

engine.reset()

engine.run()

## Створення сутностей, шаблонів та фактів[¶](#Створення-сутностей,-шаблонів-та-фактів)

В середовищі CLIPS існують deftemplate та defclass для опису шаблонів фактів та сутностей предметної області. При використані experta використовуються лише класи.

Cтворення шаблону animal та person в середовищі CLIPS:

CLIPS

(deftemplate animal

(slot name)

(slot approx\_weight)

(slot approx\_height)

(slot tail)

(slot character)

(multislot related\_animals))

(deftemplate person

(slot name)

(slot animal\_kind)

(slot age)

(multislot friends))

Cтворення класу animal та person мовою Python:

In [3]:

from experta import Fact, Field

# Сутність повина наслідуватися від класу experta.Fact

class Animal(Fact):

# атрибути класу повині бути типу experta.Filed та мати в описаний тип даних

# тип даних може бути як стандартний для Python, так і бути описаним вручну

# mandatory вказує на обов'язковість атрибуту в класі

name = Field(str, mandatory=True)

approx\_weight = Field(int, mandatory=True)

approx\_height = Field(int, mandatory=True)

tail = Field(int, mandatory=True)

character = Field(str, mandatory=True)

related\_animals = Field(list, mandatory=True)

class Person(Fact):

name = Field(str, mandatory=False)

animal\_kind = Field(str, mandatory=True)

age = Field(int, mandatory=True)

friends = Field(list, mandatory=False)

Створимо декілька початкових фактів для ЕС. Розглянем створення через deffacts та assert

CLIPS deffacts:

CLIPS

(deffacts initfacts "Initial Facts"

(dogs are our friends)

(animal (name dog) (approx\_weight 20) (approx\_height 30) (tail 1)

(related\_animals cat wolf) (character strong)

)

(animal (name cat) (approx\_weight 4) (approx\_height 10) (tail 1)

(character sweet) (related\_animals dog wolf)

)

(animal (name wolf) (approx\_weight 4) (approx\_height 40) (character bad)

(tail 1)

(related\_animals dog wolf)

)

)

Python deffacts:

In [4]:

from experta import DefFacts

class Example1(KnowledgeEngine):

@DefFacts()

def init\_pets(self):

yield Animal(name='dog', approx\_weight=20, approx\_height=30, tail=1, character='strong', related\_animals=['cat', 'wolf'])

yield Animal(name='cat', approx\_weight=4, approx\_height=10, tail=1, character='sweet', related\_animals=['wolf', 'dog'])

yield Animal(name='wolf', approx\_weight=4, approx\_height=40, tail=1, character='bad', related\_animals=['wolf', 'cat'])

CLIPS assert:

CLIPS

(assert (person (name Rick) (animal\_kind dog) (age 3) (friends Tim Kitty)))

(assert (person (name Tim) (animal\_kind dog) (age 4) (friends Rick)))

(assert (person (name Kitty) (animal\_kind cat) (age 4) (friends Rick)))

(assert (person (animal\_kind mollusc) (age 1) (friends)))

В Python точно так само зробити не можна, потрібно або створити функцію в середині об'єкта ЕС, або добавити факти в екзепляр ЕС.

Реалізація через функцію:

In [5]:

class Example2(KnowledgeEngine):

def init\_persons(self):

self.declare(Person(name="Rick", animal\_kind='dog', age=3, friends=['Tim', 'Kitty']))

self.declare(Person(name="Tim", animal\_kind='dog', age=4, friends=['Rick']))

engine = Example2()

engine.reset()

engine.init\_persons()

Додавання фактів через екземпляр ЕС:

In [6]:

engine.declare(Person(name="Kitty", animal\_kind='dog', age=4, friends=['Rick']))

engine.declare(Person(animal\_kind='mollusc', age=1))

Out[6]:

Person(animal\_kind='mollusc', age=1)

Переглянем список фактів. CLIPS:

CLIPS

(facts)

Python:

In [7]:

engine.facts

Out[7]:

FactList([(0, InitialFact()),

(1,

Person(name='Rick', animal\_kind='dog', age=3, friends=frozenlist(['Tim', 'Kitty']))),

(2,

Person(name='Tim', animal\_kind='dog', age=4, friends=frozenlist(['Rick',]))),

(3,

Person(name='Kitty', animal\_kind='dog', age=4, friends=frozenlist(['Rick',]))),

(4, Person(animal\_kind='mollusc', age=1))])

Видалення фактів за номером в CLIPS:

CLIPS

(retract 1 3)

В Python функція приймає лише одне значення тому прийдеться її викликати двічі для досягнення того ж ефекту:

In [8]:

engine.retract(1)

engine.retract(3) # якщо виконати команду ще раз, то це призведе до помилки бо факти за такими індексам відсутні

Модифікуєм факт у середовищі CLIPS:

CLIPS

(modify 2 (age 5))

Python:

In [9]:

engine.declare(engine.modify(engine.facts[2], age=5))

engine.facts

Out[9]:

FactList([(0, InitialFact()),

(4, Person(animal\_kind='mollusc', age=1)),

(5,

Person(name='Tim', animal\_kind='dog', age=5, friends=frozenlist(['Rick',])))])

Як бачимо, напряму редактувати факт не можна, його треба витягнути і заново вставити змінений, це призводить до зміщення індексації(факт 2 став 5), що може збити з толку в деяких випаках

Видалимо усі факти. CLIPS:

CLIPS

(clear)

Python:

In [10]:

engine.facts.clear() # факти зберігаються в звичайному списку, і дана функція стандартна для списку Python

engine.facts

Out[10]:

## Створення правил[¶](#Створення-правил)

Я створюю клас ЕС, від якого буду наслідувати наступні приклади. В класі я добавлю початкові факти, це дозволить уникнути дублювання коду в наступних прикладах, а також дещо змінню класи сутностей

In [11]:

class Animal(Fact):

name = Field(str, mandatory=True)

approx\_weight = Field(int, mandatory=False)

approx\_height = Field(int, mandatory=False)

tail = Field(int, mandatory=True)

water = Field(bool, mandatory=True)

related\_animals = Field(list, mandatory=True)

class Person(Fact):

name = Field(str, mandatory=False)

animal\_kind = Field(str, mandatory=True)

age = Field(int, mandatory=True)

friends = Field(list, mandatory=False)

class ExampleRules(KnowledgeEngine):

@DefFacts()

def init\_facts(self):

yield (Animal(name='dog', tail=1, water=False, land=True, approx\_weight=20, approx\_height=30, related\_animals=['wolf']))

yield (Animal(name='wolf', tail=1, water=False, land=True, approx\_weight=4, approx\_height=10, related\_animals=['dog', 'koyote']))

yield (Animal(name='shark', tail=1, water=True, land=False, approx\_weight=4, approx\_height=40, related\_animals=['fish']))

yield (Person(name="Rick", animal\_kind='dog', age=3, friends=['Tim', 'Kitty']))

yield (Person(name="Tim", animal\_kind='cat', age=4, friends=['Rick']))

yield (Person(name="Kitty", animal\_kind='dog', age=4, friends=['Rick']))

##### Задайте правила для яких умовним елементом є зразок з певними символьними обмеженнями для знаходження впорядкованих або невпорядкованих фактів:[¶](#Задайте-правила-для-яких-умовним-елемен)

CLIPS:

CLIPS

(defrule FindTailWaterAnimal

(animal (tail 1) (life\_environment water))

=>

(printout t crlf “Found tail water animal!” crlf)

)

Python:

In [12]:

from experta import Rule

class Example(ExampleRules):

# Правило задається за допомогою декоратора Rule

@Rule(Animal(water=True))

def findTailWaterAnimal(self):

print('Found tail water animal!')

engine = Example()

engine.reset()

engine.run()

*Found tail water animal!*

##### Правило, зразку якого відповідають факти, що визначать тварин, які є родичами собаки:[¶](#Правило,-зразку-якого-відповідають-факт)

CLIPS:

CLIPS

(defrule FindDogRelative

(animal (related\_animals dog))

=>

(printout t crlf “Found dog relative!” crlf)

)

У Python немає можливості зробити в точності як в CLIPS, для того щоб зробити перевірку 'dog' in [...] потрібно виокремити зміну та працювати з нею, або в декораторі, або в тілі функції. Працювати із змінною в тілі функції простіше, так і поступим:

In [13]:

from experta import MATCH

class Example(ExampleRules):

# Match.name дозволяє передати name в функцію

# Ім'я не обов'язково має співпадати з назвою атрибута сутності

# Наприклад: name=MATCH.n передасть в функцію значення n

# Якщо MATCH застосовується до зміни яку не треба передавати в середину функції, то її можна просто не приймати

@Rule(Animal(related\_animals=MATCH.related\_animals))

def findDogRelative(self, related\_animals):

if 'dog' in related\_animals:

print('Found dog relative!')

engine = Example()

engine.reset()

engine.run()

*Found dog relative!*

Команда ppdefrule у Python відсутня, як і збереження правил і фактів і вигляді файлівю Якщо потрібно подібний функціонал, то рекоменується розглянути питання серіалізації у мові Python, в часності документи **Yaml**

##### Задайте правила для яких умовним елементом є зразок в якому використовуються символьні обмеження та групові символи всіх видів для знаходження впорядкованих або невпорядкованих фактів.[¶](#Задайте-правила-для-яких-умовним-елемен)

Правило, зразку якого відповідають факти, що визначають тварину з будь-яким ім’я у віці 3х років і яка товаришує з Kitty та, можливо, з іншими тваринами

CLIPS:

CLIPS

(defrule FindKittyFriends

(person (name ?) (age 3) (friends $? Kitty $?))

=>

(printout t crlf “Found Kitty friend!” crlf)

)

Python:

In [14]:

class Example(ExampleRules):

@Rule(Person(age=3, friends=MATCH.friends))

def findKittyFriends(self, friends):

if 'Kitty' in friends:

print('Find Kitty Friends')

engine = Example()

engine.reset()

engine.run()

Find Kitty Friends

##### Задайте правила у яких використовуються зв’язуючі обмеження різних типів:[¶](#Задайте-правила-у-яких-використовуються)

Дане правило знайде всі пари тварин та їх друзів, які не є котами (кішками):

CLIPS:

CLIPS

(defrule FindAnimals

(person (name ?xname) (friends $? ?x $?))

(person (name ?x) (animal\_kind ~cat))

=>

(printout t crlf “Animal ” ?xname “ and his friend (not cat) ” ?x crlf)

)

Python:

In [15]:

from experta import L

# В Python не можна проводити тіж операції над строками, що у CLIPS, наприклад операція "не"

# Якщо нам потрібний такий функціонал, потрібно використати Literal - клас L з бібліотеки experta

class Example(ExampleRules):

@Rule(Person(animal\_kind=~L('cat'), name=MATCH.name, friends=MATCH.friends))

def findAnimals(self, name, friends):

print(f'{name} and his friend (not cat) {", ".join(list(friends))}')

engine = Example()

engine.reset()

engine.run()

Kitty and his friend (not cat) Rick

Rick and his friend (not cat) Tim, Kitty

##### Задайте правила у яких використовуються предикатні обмеження:[¶](#Задайте-правила-у-яких-використовуються)

Дане правило знайде всіх тварин, що мають ім’я довжиною до 5 літер, та їх друзів, у яких ім’я починається літерою “T”:

CLIPS:

CLIPS

(defrule PredicateFindAnimals

(person (name ?xname&:(< (str-length ?xname) 5))

(friends $? ?x&:(= 0 (str-compare (sub-string 1 1 (upcase ?x)) “T”)) $?))

=>

(printout t crlf “Animal ” ?xname “ and his friend (‘T…’) ” ?x crlf)

)

Python:

In [16]:

from experta import P, AS

# Предикат створюється за допомогою класу P який приймає булеву лямбда-функцию

# Оператор AS дозволяє передавати в середину методу цілу сутність, а не її атрибути як MATCH

class Example(ExampleRules):

@Rule(

AS.person << Person(

name=P(lambda x: len(x) < 5),

friends=P(lambda x: [True for name in x if name[0] == "T"])

)

)

def PredicateFindAnimals(self, person):

for fr in person.as\_dict()["friends"]:

print(f'{person.as\_dict()["name"]} and his friend (T...) {fr}' )

engine = Example()

engine.reset()

engine.run()

*Rick and his friend (T...) Tim*

*Rick and his friend (T...) Kitty*

Задайте правила у яких використовуються обмеження, що повертають значення:

Дане правило знайде всіх тварин у яких середня вага в 5 разів перевищує вік:

CLIPS:

CLIPS

(defrule ReturnFindAnimals

(person (name ?xname) (animal\_kind ?ac) (age ?age))

(animal (name ?ac) (approx\_weight ?aw&=(\* 5 ?age)))

=>

(printout t crlf “Animal ” ?xname “, kind = ” ?ac “, weight = ” ?aw “, age = ” ?age crlf)

)

Python немає прямих аналогів таких обмежень, найкращим вибором буде використовувати клас TEST:

In [17]:

from experta import TEST

# TEST дозволяє створювати більш складні предикати

# MATCH передає зміні не тільки в функцію, але і у TEST

class Example(ExampleRules):

@Rule(Person(animal\_kind=MATCH.animal\_kind, age=MATCH.age),

Animal(name=MATCH.name, approx\_weight=MATCH.approx\_weight),

TEST(lambda name, animal\_kind: animal\_kind == name),

TEST(lambda age, approx\_weight: age < approx\_weight / 5)

)

def f(self, name, age, animal\_kind, approx\_weight):

print(name, animal\_kind, age, approx\_weight)

engine = Example()

engine.reset()

engine.run()

*dog dog 3 20*

##### Задайте правила з використанням умовного елементу test.[¶](#Задайте-правила-з-використанням-умовног)

Це правило знаходить усі можливі пари тварин у яких перша тварина має більший зріст ніж друга і на екран виводяться назви цих тварин:

CLIPS:

CLIPS

(defrule TestRule

(animal (name ?n1) (approx\_height ?h1))

(animal (name ?n2) (approx\_height ?h2))

(test (> ?h1 ?h2))

=>

(printout t crlf ?n1 " height > " ?n2 " height" crlf)

)

Python:

In [18]:

class Example(ExampleRules):

@Rule(

Animal(approx\_height=MATCH.approx\_height1, name=MATCH.n1),

Animal(approx\_height=MATCH.approx\_height2, name=MATCH.n2),

TEST(lambda approx\_height1, approx\_height2: approx\_height1 > approx\_height2)

)

def testRule(self, n1, n2):

print(n1, 'higest than', n2)

engine = Example()

engine.reset()

engine.run()

*shark higest than wolf*

*shark higest than dog*

*dog higest than wolf*

Це правило знаходить усі можливі пари тварин, які мають хвіст та у яких перша тварина має не меншу вагу ніж друга і на екран виводяться назви цих тварин:

CLIPS

CLIPS

(defrule TestRule1

(animal (name ?n1) (approx\_weight ?h1) (tail 1))

(animal (name ?n2) (approx\_weight ?h2) (tail 1))

(test (>= ?h1 ?h2))

=>

(printout t crlf ?n1 " weight >= " ?n2 " weight" crlf)

)

Python

In [19]:

class Example(ExampleRules):

@Rule(

Animal(approx\_weight=MATCH.approx1, name=MATCH.n1, tail=1),

Animal(approx\_weight=MATCH.approx2, name=MATCH.n2, tail=1),

TEST(lambda approx1, approx2: approx1 >= approx2)

)

def testRule1(self, n1, n2):

print(n1, 'more heavily than', n2)

engine = Example()

engine.reset()

engine.run()

*shark more heavily than wolf*

*wolf more heavily than shark*

*dog more heavily than shark*

*shark more heavily than shark*

*dog more heavily than wolf*

*wolf more heavily than wolf*

*dog more heavily than dog*

##### Задайте правила з використанням комбінацій умовних елементів or, and та not.[¶](#Задайте-правила-з-використанням-комбіна)

Це правило знаходить всіх тварин, які мають хвіст або живуть у воді:

CLIPS:

CLIPS

(defrule OrAndNotRule

(or (animal (name ?n1) (tail 1))

(animal (name ?n1) (life\_environment water)))

=>

(printout t crlf ?n1 crlf)

)

Python:

In [20]:

from experta import OR

# Логічні функції аналогічні синтаксису CLIPS

class Example(ExampleRules):

@Rule(OR(Animal(tail=1, name=MATCH.name),

Animal(water=True, name=MATCH.name))

)

def orAndNotRule(self, name):

print(name)

engine = Example()

engine.reset()

engine.run()

*shark*

*shark*

*wolf*

*dog*

Це правило знаходить всіх кішок або тих тварин, які товаришують з Kitty:

CLIPS:

CLIPS

(defrule OrAndNotRule1

(or (person (name ?n1) (animal\_kind cat))

(person (name ?n1) (friends $? Kitty $?)))

=>

(printout t crlf ?n1 crlf)

)

Python:

In [21]:

class Example(ExampleRules):

@Rule(OR(

Person(name=MATCH.name, animal\_kind='cat'),

Person(name=MATCH.name, friends=P(lambda x: "Kitty" in x))

))

def orAndNotRule1(self, name):

print(name)

engine = Example()

engine.reset()

engine.run()

*Tim*

*Rick*

Це правило знаходить тих тварин, які не живуть у воді та мають поганий характер або тих тварин, які споріднені з кішками:

CLIPS:

CLIPS

(defrule OrAndNotRule2

(or (and (not (life\_environment water))

(animal (name ?n1) (character bad))

)

(animal (name ?n1) (related\_animals $? cat $?))

)

=>

(printout t crlf “Found animal: ” ?n1 crlf)

)

Python:

In [22]:

from experta import AND

class Example(ExampleRules):

@Rule(OR(

AND(Animal(name=MATCH.name, character='bad'),

Animal(name=MATCH.name, water=~L(True))

),

Animal(name=MATCH.name, related\_animals=P(lambda x: 'cat' in x))

))

def orAndNotRule2(self, name):

print(name)

engine = Example()

engine.reset()

engine.run()

##### Задайте правила з використанням умовного елементу exists та наведіть їх внутрішнє представлення з використанням умовного елементу not.[¶](#Задайте-правила-з-використанням-умовног)

Це правило спочатку перевіряє чи вірне твердження, що собаки є нашими друзями, а потім первіряє чи є серед набору твари ті, які споріднені з собаками. Дане правило активізується лише раз, не дивлячись на те, що у нас є кілька тварин, які споріднені з собаками:

CLIPS:

CLIPS

(defrule ExistsRule

(dogs are our friends)

(exists (animal (related\_animals $? dog $?)))

=>

(printout t crlf “Found tailed dog’s relatives” crlf)

)

(defrule ExistsRule1

(dogs are our friends)

(not (not (animal (related\_animals $? dog $?))))

=>

(printout t crlf “Found tailed dog’s relatives” crlf)

)

Python:

In [23]:

from experta import EXISTS, NOT

class Example(ExampleRules):

@Rule(

EXISTS(Animal(related\_animals=P(lambda x: 'dog' in x)))

)

def existsRule(self):

print("existsRule: Found tailed dog’s relatives")

@Rule(

NOT(NOT((Animal(related\_animals=P(lambda x: 'dog' in x)))))

)

def existsRule1(self):

print("existsRule1: Found tailed dog’s relatives")

engine = Example()

engine.reset()

engine.run()

existsRule1: Found tailed dog’s relatives

existsRule: Found tailed dog’s relatives

##### Задайте правила з використанням умовного елементу forall та наведіть їх внутрішнє представлення з використанням умовних елементів and та not.[¶](#Задайте-правила-з-використанням-умовног)

Це правило перевіряє чи усі тварини, що живуть на суші мають хвіст:

CLIPS:

CLIPS

(defrule ForAllRule

(forall

(animal (tail 1))

(animal (life\_environment land))

)

=>

(printout t crlf “All land animals are tailed” crlf)

)

(defrule ForAllRule1

(not (and (animal (tail 1))

(not (animal (life\_environment land)))

)

)

=>

(printout t crlf “All land animals are tailed” crlf)

)

In [24]:

from experta import FORALL

class Example(ExampleRules):

@Rule(

FORALL(

Animal(tail=1),

Animal(land=True)

)

)

def forAllRule(self):

print('ForAllRule: All land animals are tailed')

@Rule(

NOT(

AND(

Animal(tail=1),

NOT(Animal(land=True))

)

)

)

def ForAllRule1(self):

print('ForAllRule1: All land animals are tailed')

engine = Example()

engine.reset()

engine.run()

*ForAllRule1: All land animals are tailed*

*ForAllRule1: All land animals are tailed*

*ForAllRule: All land animals are tailed*

## Сворення родових функцій та методів[¶](#Сворення-родових-функцій-та-методів)

Бібліотекою experta не передбачено особливих засобів для відтворення родових функцій та методів CLIPS. Реалізація таких методів покладена на стандартні засоби мови Python

##### Створимо функцію для визначення середньої ваги тварини певного виду на основі результатів вимірів двох тварин цього виду:[¶](#Створимо-функцію-для-визначення-середнь)

CLIPS

(deffunction average (?x ?y) (/ (+ ?x ?y) 2))

##### Створимо функцію для визначення середньої ваги тварини певного виду, яка може приймати 2 або 3 параметри:[¶](#Створимо-функцію-для-визначення-середнь)

CLIPS

(deffunction average3 (?x ?y $?z)

(if (= 0 (length ?z))

then

(average ?x ?y)

else

(/ (+ ?x ?y (nth 1 ?z)) 3)

))

На Python код буде виглядати наступним чином:

In [25]:

def avarage(x, y):

print((x + y) / 2)

def avarage3(x, y, z=None):

if z: print((x + y + z) / 3)

else: avarage(x, y)

avarage(1, 2)

avarage3(1, 2, 4)

1.5

2.3333333333333335

##### Створимо методи родової функції для всіх можливих наборів і послідовностей параметрів:[¶](#Створимо-методи-родової-функції-для-всі)

CLIPS

(defmethod showWeight ( (?name STRING) (?lo NUMBER) (?hi NUMBER (> ?hi ?lo)))

(printout t "Animal of type " ?name " has weight in range (" ?lo " to " ?hi ")" crlf ))

(defmethod showWeight ( (?name STRING) (?hi NUMBER) (?lo NUMBER (> ?hi ?lo)))

(printout t "Animal of type " ?name " has weight in range (" ?lo " to " ?hi ")" crlf ))

(defmethod showWeight ( (?lo NUMBER) (?hi NUMBER (> ?hi ?lo)))

(printout t "Animal of type <UNKNOWN> has weight in range (" ?lo " to " ?hi ")" crlf ))

(defmethod showWeight ( (?hi NUMBER) (?lo NUMBER (> ?hi ?lo)))

(printout t "Animal of type <UNKNOWN> has weight in range (" ?lo " to " ?hi ")" crlf ))

Попробуєм повтрити функціонал CLIPS:

In [26]:

def showWeight(x, y, name='<UNKNOWN>'):

print(f'Animal of type {name} has weight in range ({min(x, y)} to {max(x, y)})')

showWeight(0.5, 1.2)

showWeight(0.6, 0.3)

showWeight(5, 10, 'Sobaka')

showWeight(25, 10, 'Velika Sobaka')

*Animal of type <UNKNOWN> has weight in range (0.5 to 1.2)*

*Animal of type <UNKNOWN> has weight in range (0.3 to 0.6)*

*Animal of type Sobaka has weight in range (5 to 10)*

*Animal of type Velika Sobaka has weight in range (10 to 25)*

Або створимо функцію яка може викликати різні інструкції в залежності від параметрів:

In [27]:

def showWeight(x, y, name=''):

def f1():

print(f'Animal of type <UNKNOWN> has weight in range ({x} to {y})')

def f2():

print(f'Animal of type <UNKNOWN> has weight in range ({y} to {x})')

def f3():

print(f'Animal of type {name} has weight in range ({x} to {y})')

def f4():

print(f'Animal of type {name} has weight in range ({y} to {x})')

if x < y and not name: f1()

if x > y and not name: f2()

if x < y and name: f3()

if x > y and name: f4()

showWeight(0.5, 1.2)

showWeight(0.6, 0.3)

showWeight(5, 10, 'Sobaka')

showWeight(25, 10, 'Velika Sobaka')

*Animal of type <UNKNOWN> has weight in range (0.5 to 1.2)*

*Animal of type <UNKNOWN> has weight in range (0.3 to 0.6)*

*Animal of type Sobaka has weight in range (5 to 10)*

*Animal of type Velika Sobaka has weight in range (10 to 25)*

In [ ]: