Prolog Programming Assignment #2: A Favorite Pokemon KB plus Simple List Processing Exercises

What's It All About?

Programming exercises that (1) focus on querying and extending a Prolog knowledge base about Pokemon, and (2) afford you an opportunity to engage in som Prolog list processing.

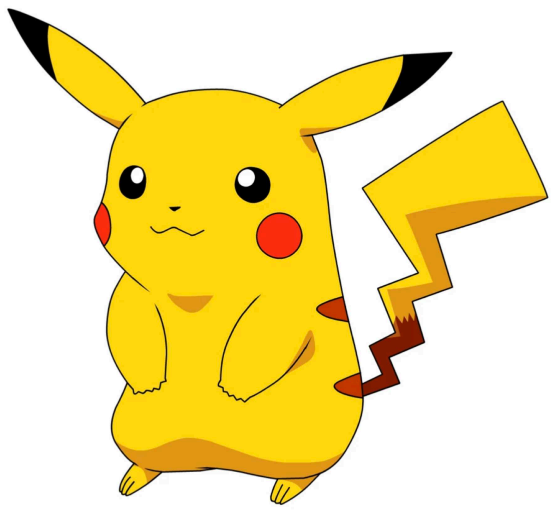
Overall Charge

Generate a solution template document that is consistent with the accompanying solution template. Then, please do each of the Prolog tasks, adding source code and demos to your template in the appropriate manner.

Due Date

Please complete your work on this assignment, and post your work to your web work site, no later than Sunday, April 16th, 2023.

Task 1: Pokemon KB Interaction and Programming



Part 1: The Initial Pokemon KB

For this task, please incorporate, into your computational world, the knowledge base on pokemon trading cards that I am providing as a sibling document to the one that you are now reading. The code works for me, so if it doesn't work for you, that is probably because of an "error in transmission" that you will need to sort out.

Once the KB loads, simply copy it into the appropriate part of your solution document.

Part 2: Queries

Please engage in a Prolog interaction to duplicate what I did in rendering the accompanying demo – before I replaced my queries with "query stubs". In other words, you are to recreate the demo by providing the 20 queries that I redacted. The following "20 questions" indicate what you need to focus on for each query:

1. Query 1: Is picachu a "creatio ex nihilo" (created out of nothing) pokemon?
2. Query 2: Is raichu a "creatio ex nihilo" pokemon?
3. Query 3: By means of hand intervention, list all of the "creatio ex nihilo" pokemon.
4. Query 4: By means of the standard idiom of repetition, list all of the "creatio ex nihilo" pokemon.
5. Query 5: Does squirtle evolve into wartortle?
6. Query 6: Does wartortle evolve into squirtle?
7. Query 7: Does squirtle evolve into blastoise?
8. Query 8: By means of hand intervention, list all triples of pokemon such that the first evolves into the second and the second evolves into the third.
9. Query 9: By means of the standard idiom of repetition, list all pairs of pokemon such that the first evolves through an intermediary to the second - placing an arrow between each pair.
10. Query 10: By means of the standard idiom of repetition, list the names of all of the pokemon.
11. Query 11: By means of the standard idiom of repetition, list the names of all of the fire pokemon.
12. Query 12: By means of the standard idiom of repetition, provide a summary of each pokemon and its kind, representing each pairing of name and kind in the manner suggested by the redacted demo.
13. Query 13: What is the name of the pokemon with the waterfall attack?
14. Query 14: What is the name of the pokemon with the poison-powder attack?
15. Query 15: By means of the standard idiom of repetition, list the names of the attacks of all of the water pokemon.
16. Query 16: How much damage (hp count) can poliwhirl absorb?
17. Query 17: How much damage (hp count) can butterfree absorb?
18. Query 18: By means of the standard idiom of repetition, list the names of all of the pokemon that can absorb more than 85 units of damage.
19. Query 19: By means of the standard idiom of repetition, list the names of all of the pokemon that can dish out more than 60 units of damage with one instance of their attack.
20. Query 20: By means of the standard idiom of repetition, list the names and the hit point value for each of the "creation ex nihilo" pokemon, with the results formatted as the redacted demo suggests.

```
bash-3.2$ swipl
<<redacted>>

?- consult('pokemon_plus.pro').
% pokemon.pro compiled 0.00 sec, 54 clauses
true.

?- <<Query 1>>
true.

?- <<Query 2>>
false.

?- <<Query 3>>
Name = pikachu ;
Name = bulbasaur ;
Name = caterpie ;
Name = charmander ;
Name = vulpix ;
Name = poliwag ;
Name = squirtle ;
Name = staryu.

?- <<Query 4>>
pikachu
bulbasaur
caterpie
charmander
vulpix
poliwag
squirtle
staryu
false.

?- <<Query 5>>
true.

?- <<Query 6>>
false.

?- <<Query 7>>
false.

?- <<Query 8>>
X = bulbasaur,
Y = ivysaur,
Z = venusaur ;
X = caterpie,
Y = metapod,
Z = butterfree ;
```

```
X = charmander,
Y = charmeleon,
Z = charizard ;
X = poliwag,
Y = poliwhirl,
Z = poliwrath ;
X = squirtle,
Y = wartortle,
Z = blastoise ;
false.

?- <<Query 9>>
bulbasaur --> venusaur
caterpie --> butterfree
charmander --> charizard
poliwag --> poliwrath
squirtle --> blastoise
false.

?- <<Query 10>>
pikachu
raichu
bulbasaur
ivysaur
venusaur
caterpie
metapod
butterfree
charmander
charmeleon
charizard
vulpix
ninetails
poliwag
poliwhirl
polywrath
squirtle
wartortle
blastoise
staryu
starmie
false.

?- <<Query 11>>
charmander
charmeleon
charizard
vulpix
ninetails
false.

?- <<Query 12>>
nks(name(pikachu),kind(electric))
nks(name(raichu),kind(electric))
nks(name(bulbasaur),kind(grass))
nks(name(ivysaur),kind(grass))
nks(name(venusaur),kind(grass))
nks(name(caterpie),kind(grass))
nks(name(metapod),kind(grass))
nks(name(butterfree),kind(grass))
nks(name(charmander),kind(fire))
nks(name(charmeleon),kind(fire))
nks(name(charizard),kind(fire))
nks(name(vulpix),kind(fire))
nks(name(ninetails),kind(fire))
nks(name(poliwag),kind(water))
nks(name(poliwhirl),kind(water))
nks(name(polywrath),kind(water))
nks(name(squirtle),kind(water))
nks(name(wartortle),kind(water))
nks(name(blastoise),kind(water))
nks(name(staryu),kind(water))
nks(name(starmie),kind(water))
false.

?- <<Query 13>>
N = wartortle

?- <<Query 14>>
N = venusaur

?- <<Query 15>>
water-gun
amnesia
dashing-punch
bubble
waterfall
hydro-pump
slap
star-freeze
false.

?- <<Query 16>>
HP = 80

?- <<Query 17>>
HP = 130

?- <<Query 18>>
raichu
venusaur
butterfree
charizard
ninetails
polywrath
blastoise
false.
```

```
?- <<Query 19>>
raichu
venusaur
butterfree
charizard
ninetails
false.

?- <<Query 20>>
pikachu:  60
bulbasaur:  40
caterpie:  50
charmander:  50
vulpix:  60
poliwag:  60
squirtle:  40
staryu:  40
false.
```

Part 3: Programs

Please extend the pokemon knowledge base in your `pokemon.pro` file by adding the programs (rules) specified below, consulting the accompanying demo for clarification, as needed. **Note: You should probably proceed by adding one predicate at a time, being sure to test/demo that predicate before moving on to do the next one.**

1. Define a parameterless predicate called `display_cen` to display the names of all of the "creatio ex nihilo" pokemon.
2. Define a parameterless predicate called `display_not_cen` to display the names of all of the pokemon who are not "creatio ex nihilo" pokemon.
3. Define a predicate called `generator` taking two parameters, the name of a pokemon and the type of a pokemon, which returns true if the pokemon is of the given type.
4. Define the parameterless predicate called `display_names` to list the names of all of the pokemon represented in the KB.
5. Define the parameterless predicate called `display_attacks` to list the names of all of the attacks that the pokemon represented in the KB can unleash.
6. Define the parameterless predicate called `display_cen_attacks` to list the names of all of the attacks that just the creation ex nihilo pokemon represented in the KB can unleash.
7. Define a predicate called `indicate_attack` taking one parameter, the name of a pokemon, which displays, for the named pokemon, a short text of the form: NAME –> ATTACK.
8. Define a parameterless predicate called `indicate_attacks` which displays a short text of the form NAME –> ATTACK for each pokemon in the KB, one short text per line.
9. Define a predicate called `powerful` taking one parameter, the name of a pokemon, which succeeds only if the attack associated with the named pokemon yields with more than 55 units of damage.
10. Define a predicate called `tough` taking one parameter, the name of a pokemon, which succeeds only if the the named pokemon can absorb at least 100 units of damage (that is, has an hp count that is more than 100).
11. Define a predicate called `awesome` taking one parameter, the name of a pokemon, which succeeds only if the the named pokemon is both powerful and tough.

12. Define a predicate called `powerful_but_not_vulnerable` taking one parameter, the name of a pokemon, which succeeds only if the the named pokemon is powerful and not tough.
13. Define a predicate called `type` taking two parameters, the name of a pokemon, and the type of a pokemon, which succeeds only if the the named pokemon is of the specified type.
14. Define a predicate called `dump_kind` taking one parameter, the kind of a pokemon, which displays complete information for all of the pokemon in the KB of the specified kind, doing so in a manner that is consistent with the representation of the pokemon in the KB.
15. Define a predicate called `family` taking one parameter, presumed to be a "creatio ex nihilo" pokemon, which displays the "evolutionary family" of the specified pokemon, all on a given line, as illustrated in the demo.
16. Define a parameterless predicate called `families` to display all of the evolutionary pokemon families, representing the families in the manner illustrated in the demo.
17. Define a predicate called `lineage` taking one parameter, the name of a pokemon, which displays all of the information for the pokemon and for each subsesquent pokemon in the evolutionary lineage of the pokemon. (Please see the demo to assure that you are properly understanding what this program is supposed to do.)

```
bash-3.2$ swipl
<<redacted>>

?- consult('pokemon_plus.pro').
% pokemon_plus.pro compiled 0.00 sec, 69 clauses
true.

?- display_cen_names.
pikachu
bulbasaur
caterpie
charmander
vulpix
poliwag
squirtle
staryu
true.

?- display_not_cen_names.
raichu
ivysaur
venusaur
metapod
butterfree
charmeleon
charizard
ninetails
poliwhirl
poliwrath
wartortle
blastoise
starmie
true.

?- generator(Name,fire).
```

```
Name = charmander ;
Name = vulpix ;
false.

?- generator(Name,water).
Name = poliwag ;
Name = squirtle ;
Name = staryu ;
false.

?- generator(Name,electric).
Name = pikachu ;
false.

?- generator(Name,grass).
Name = bulbasaur ;
Name = caterpie ;
false.

?- display_names.
pikachu
raichu
bulbasaur
ivysaur
venusaur
caterpie
metapod
butterfree
charmander
charmeleon
charizard
vulpix
ninetails
poliwag
poliwhirl
poliwrath
squirtle
wartortle
blastoise
staryu
starmie
true.

?- display_attacks.
gnaw
thunder-shock
leech-seed
vine-whip
poison-powder
gnaw
stun-spore
whirlwind
scratch
slash
```

```
royal-blaze
confuse-ray
fire-blast
water-gun
amnesia
dashing-punch
bubble
waterfall
hydro-pump
slap
star-freeze
true.

?- display_cen_attacks.
gnaw
leech-seed
gnaw
scratch
confuse-ray
water-gun
bubble
slap
true.

?- indicate_attack(charmander).
charmander --> scratch
true

?- indicate_attack(bulbasaur).
bulbasaur --> leech-seed
true

?- indicate_attacks.
pikachu --> gnaw
raichu --> thunder-shock
bulbasaur --> leech-seed
ivysaur --> vine-whip
venusaur --> poison-powder
caterpie --> gnaw
metapod --> stun-spore
butterfree --> whirlwind
charmander --> scratch
charmeleon --> slash
charizard --> royal-blaze
vulpix --> confuse-ray
ninetails --> fire-blast
poliwag --> water-gun
poliwhirl --> amnesia
poliwrath --> dashing-punch
squirtle --> bubble
wartortle --> waterfall
blastoise --> hydro-pump
staryu --> slap
starmie --> star-freeze
```

```
true.

?- powerful(Name).
Name = raichu ;
Name = venusaur ;
Name = butterfree ;
Name = charizard ;
Name = ninetails ;
Name = wartortle ;
Name = blastoise ;
false.

?- tough(Name).
Name = venusaur ;
Name = butterfree ;
Name = charizard ;
Name = poliwrath ;
Name = blastoise ;
false.

?- awesome(Name).
Name = venusaur ;
Name = butterfree ;
Name = charizard ;
Name = blastoise ;
false.

?- powerful_but_vulnerable(Name).
Name = raichu ;
Name = ninetails ;
Name = wartortle ;
false.

?- type(squirtle,Type).
Type = water

?- type(caterpie,Type).
Type = grass

?- type(Name,fire),write(Name),nl,fail.
charmander
charmeleon
charizard
vulpix
ninetails
false.

?- dump_kind(water).
pokemon(name(poliwag),water,hp(60),attack(water-gun,30))
pokemon(name(poliwhirl),water,hp(80),attack(amnesia,30))
pokemon(name(poliwrath),water,hp(140),attack(dashing-punch,50))
pokemon(name(squirtle),water,hp(40),attack(bubble,10))
pokemon(name(wartortle),water,hp(80),attack(waterfall,60))
pokemon(name(blastoise),water,hp(140),attack(hydro-pump,60))
pokemon(name(staryu),water,hp(40),attack(slap,20))
pokemon(name(starmie),water,hp(60),attack(star-freeze,20))
true.

?- dump_kind(grass).
pokemon(name(bulbasaur),grass,hp(40),attack(leech-seed,20))
pokemon(name(ivysaur),grass,hp(60),attack(vine-whip,30))
pokemon(name(venusaur),grass,hp(140),attack(poison-powder,70))
pokemon(name(caterpie),grass,hp(50),attack(gnaw,20))
pokemon(name(metapod),grass,hp(70),attack(stun-spore,20))
pokemon(name(butterfree),grass,hp(130),attack(whirlwind,80))
true.

?- family(pikachu).
pikachu raichu
true

?- family(bulbasaur).
bulbasaur ivysaur venusaur
true.

?- family(caterpie).
caterpie metapod butterfree
true.

?- families.
pikachu raichu
bulbasaur ivysaur venusaur
caterpie metapod butterfree
charmander charmeleon charizard
vulpix ninetails
poliwag poliwhirl poliwrath
squirtle wartortle blastoise
staryu starmie
true.

?- lineage(pikachu).
pokemon(name(pikachu),electric,hp(60),attack(gnaw,10))
pokemon(name(raichu),electric,hp(90),attack(thunder-shock,90))
true

?- lineage(squirtle).
pokemon(name(squirtle),water,hp(40),attack(bubble,10))
pokemon(name(wartortle),water,hp(80),attack(waterfall,60))
pokemon(name(blastoise),water,hp(140),attack(hydro-pump,60))
true

?- lineage(wartortle).
pokemon(name(wartortle),water,hp(80),attack(waterfall,60))
pokemon(name(blastoise),water,hp(140),attack(hydro-pump,60))
true

?- lineage(blastoise).
```

```
pokemon(name(blastoise),water,hp(140),attack(hydro-pump,60))
true.

?- lineage(charmander).
pokemon(name(charmander),fire,hp(50),attack(scratch,10))
pokemon(name(charmeleon),fire,hp(80),attack(slash,50))
pokemon(name(charizard),fire,hp(170),attack(royal-blaze,100))
true

?-
```

Part 4: Demo

Create a demo which mimics that provided in the previous part of this task, the demo which was provided to help inform you about what the various Prolog predicates do.

Part 5: Add 12 More Pokemon to the KB

Add 12 additional pokemon to the KB, all from the four types present in the KB, and at least two from each of the four types.

Part 6: Demo

Create a demo which, once again, mimics that provided in the thrid task, after loading the KB augmented with your 12 pokemon.

Presentational Notes for Task 1

Place the following items within the "Task 1: Pokemon KB Interactions and Programming" section of your presentation document:

1. The given Pokemon KB (the `pokemon.pro` file) for Part 1 of this task.
2. The demo for Part 2 of this task.
3. The Extended knowledge base (the `pokemon.pro` file) for Part 3 of this task.
4. Your very own Part 4 demo, which mimics the demo that I presented in the Part 3 task, thus assuring that your predicate definions are soundly written.
5. The Extended knowledge base (the `pokemon.pro` file) for Part 5 of this task.
6. Your very own Part 6 demo, which mimics the demo that I presented in the Part 3 task, but which operates upon the KB with your 12 pokemon added.

Task 2: List Processing in Prolog

1. Do what is asked in the “Head/Tail Referencing Exercises” section that is presented in Lesson 5 on list processing in Prolog.
2. Establish a file called `list_processors.pro` in which to place some list processing functions. Then, add to it definitions of the functions appearing in the “Example List Processors” section of Lesson 5. Then, mimic the demo associated with the example list processor functions presented in Lesson 5, being sure to save the demo for inclusion in your presentation document.
3. Please do what is asked in the “List Processing Exercises” section of Lesson 5, which involves defining some functions and performing a demo.

Presentational Notes for Task 2

Place the following items within the “Task 2: List Processing in Prolog” section of your presentation document:

1. Your demo corresponding to the “Head/Tail Referencing Exercises”.
2. The Prolog file containing all of the list processing function definitions that you were asked to write, those associated with the “Example List Processors” section from Lesson 5, and those associated with the “List Processing Exercises” from Lesson 5.
3. The demo associated with the “Example List Processors” that is provided in Lesson 5.
4. The demo that you are asked to create in the “List Processing Exercises” section of Lesson 5.