Question 1

The following `if` blocks are equivalent to the one given in the problem statement:

I.

```
if (e())
  do_something();
if (f())
  if (g())
    do_something();
if (h())
  do_something();
```

II.

```
if (e())
  if (f())
    do_something();
if (g())
  if(h())
    do_something();
```

III.

```
if (e())
  if (f())
    do_something();
  if (g())
    do_something();
```

Question 2

(a)
```
class Node:
  def __init__(self, val):
    self.value = val
    self.next = None

class HashTable:
  def __init__(self, buckets):
    self.array = [None] * buckets

  def insert(self, val):
    bucket = hash(val) % len(self.array)
    tmp_head = Node(val)
    tmp_head.next = self.array[bucket]
    self.array[bucket] = tmp_head

def gen(arr):
  for i in range(arr):
    curr = arr[i]
    while arr[i] is not None:
      yield curr.value
      curr = curr.next
```

(b)
```
def HTIterator:
  def __init__(self, arr):
    self.arr = arr
    self.bucket = -1
    self.curr = None

  def __next__(self):
    while self.curr is None:
      self.bucket += 1

      if len(self.arr) <= self.bucket:
        raise StopIteration

      self.curr = self.arr[self.bucket]
    value = self.curr.value
    self.curr = self.curr.next
    return value
```

(c)
```
hash_table = HashTable(10)
...
for item in hash_table:
  print(item)
```

(d)
```
hash_table = HashTable(10)
...
it = hash_table.__iter__()
while True:
  try:
    print(it.__next__())
  except StopIteration:
    pass
```

(e)
```
class HashTable:
...
```

```
def forEach(self, f):
  for i in range(0, len(self.array)):
    curr = arr[i]
    while curr is not None:
      f(curr)
      curr = curr.next
```

Question 3

(a) `X = green`

(b) `false`

(c) `Q = tomato`,
`Q = beet`

(d) `Q = celery, R = green`,
`Q = tomato, R = red`,
`Q = persimmon, R = orange`,
`Q = beet, R = red`,
`Q = lettuce, R = green`

Question 4

(a)
```
likes_red(X) :- food(Y), likes(X, Y), color(Y, red).
```

(b)
```
likes_food_color(X, Z) :-
        food(Y),
        likes(X, Y),
        color(Y, Z).
    likes_foods_of_colors_menachen_likes(X) :-
        likes_food_color(X, Z),
        likes_food_color(menachen, Z).
```

Question 5

```
reachable(X, Y) :-
    road_between(X, Y);
    road_between(Y, X);
    road_between(X, Z),
    road_between(Z, Y).
```

Question 6

(a) {`X = bar`}

(b) Does not unify since the arities do not match.

(c) {`Z = X`}

(d) {`X = barf, Y = bletch`}

(e) Does not unify since `bletch` $\neq$ `barf`

(f) {`X = bar, Y = barf`}

(g) {`Y = bar(a, Z)`}

(h) Does not unify since `Z` cannot bind to both `barf` and `bletch`.

(i) {`Q = [A|B|C]`}

(j) Does not unify since `X` cannot bind to `[a]`.

Question 7

```
insert_lex(X, [], [X]).
insert_lex(X, [Y|T], [X,Y|T]) :- X =< Y.
insert_lex(X, [Y|T], [Y|NT]) :-
    X > Y, insert_lex(X, T, NT).
```

Question 8

```
count_elem([], Total, Total).
count_elem([Hd|Tail], Sum, Total) :-
    Sum1 is Sum + 1,
    count_elem(Tail, Sum1, Total).
```

Question 9

```
gen_list(_, 0, []).
gen_list(Value, N, [Value|Tail]) :-
    N > 0,
    N1 is N - 1,
    gen_list(Value, N1, Tail).
```

Question 10

```
append_item([], Item, [Item]).
append_item([Head|TailIn], Item, [Head|TailOut]) :-
    append_item(TailIn, Item, TailOut).
```