(T) To avoid the dangling-else problem, Java requires every if statement to have both a then clause and an else clause.
(T) In functional languages, the function is always a first-class citizen.
(T) Lazy evaluation is a technique that can make it easy to avoid unnecessary computation.
(T) Not every Scheme procedure returns a value.
(T) The Scheme interpreter optimizes tail-recursion as iteration.
(T) In Scheme, any value that is not considered as False is interpreted as True.
(T) Scheme's syntax eliminates the need for precedence rules.
(F) Swing was the original Java GUI toolkit and the AWT was added in Java 2.
(F) In the model-view-controller design pattern, the model implements the user interface.
(T) Like Lisp, Java uses garbage collection to perform memory management.
(F) Prolog is a strongly typed language.
(F) All local variables must be declared before they are used in Prolog.
(F) Unlike the Reference Counting algorithm, Mark-Sweep makes only a single pass through the heap.
(T) When a thread is waiting for an I/O operation, it is said to be blocked.
(F) Executing a program with more than one Java thread requires a computer with multiple processors.
You need to design a program to schedule the times and room assignments for all final exams at a large university. Which programming language/paradigm is the best choice? : **Logical, Prolog**
You need to design a program that crawls web pages and extracts any contact information. Which programming language/paradigm is the best choice? : **Imperative, Perl**
You need to design a simulator for an autonomous Mars rover. Which programming language/paradigm is the best choice? : **Functional, Scheme**
You need to design an event-driven game with a graphical user interface and many related characters, actions, and items. Which programming language/paradigm is the best choice? : **O-O, Python**
program designed to have two or more execution contexts : **concurrent program**
encapsulates a shared variable with operations "signal" and "wait" : **monitor**
block of heap memory that cannot be accessed by the program : **garbage**
an integer variable and an associated thread queue : **semaphore**
concept particularly useful in event-driven GUI programs : **model-view-controller**
program designed so that different pieces are run on autonomous computers connected by a network : **distributed program**
block of information associated with each function activation : **activation record**
symbol that names the relationship : **functor**
occurs when the resulting value of a variable depends on the execution order of two or more threads : **race condition**
program type that typically has no perceived stopping point : **event-driven program**
delays argument evaluation in a function call until the argument is needed : **lazy evaluation**
occurs when a thread is waiting for an event that will never happen : **deadlock**
contains a head and a list of predicates : **horn clause**
expression that appears in a function call : **argument**
allows inferred propositions to be computed from given propositions : **resolution**

```
begin
    integer a;
    procedure foo (b: integer);
        begin
            a := a+1;
            b := b+4;
            print (a);
        end;
    a := 0;
    foo(a);
    print(a);
end;
```

How many function parameters are there in the code? **(1)**

```
begin
    integer a;
    procedure foo (b: integer);
        begin
            a := a+1;
            b := b+4;
            print (a);
        end;
    a := 0;
    foo(a);
    print(a);
end;
```

How many function arguments appear in the code? Give the total number, summing over function calls. **(3)**

```
begin
    integer a;
    procedure foo (b: integer);
        begin
            a := a+1;
            b := b+4;
            print (a);
        end;
    a := 0;
    foo(a);
    print(a);
end;
```

What values are printed if the language is pass by value? **(1, 1)**

```
begin
    integer a;
    procedure foo (b: integer);
        begin
            a := a+1;
            b := b+4;
            print (a);
        end;
    a := 0;
    foo(a);
    print(a);
end;
```

What two values are printed if the language is pass by value-result? **(1, 4)**

```
begin
    integer a;
    procedure foo (b: integer);
        begin
            a := a+1;
            b := b+4;
            print (a);
        end;
    a := 0;
    foo(a);
    print(a);
end;
```

What two values are printed if the language is pass by reference? **(5, 5)**
Complete a Scheme function that returns the cube x^3 of a given input x.
Example use:
> (cube 3)
27

(define (cube x)
(* x (* x (* x)))
)

(define (cube x)
(* x x x)
)

(define (cube x)
(expt x 3)
)

(define (f x) (cube x))
(define (g y) (plusOne y))
What is the output of this line of Scheme code? **(9)**
(g (f 2))

(define (f x) (cube x))
(define (g y) (plusOne y))
What is the output of this line of Scheme code? **(2 9 28)**
(map g (map f '(1 2 3)))

(define (f x) (cube x))
(define (g y) (plusOne y))
What is the output of this line of Scheme code? **(8 27 64)**
(map f (map g '(1 2 3)))

```
A(m,n) = { n+1                 if m = 0,
           A(m-1,1)            if m > 0 and n = 0,
           A(m-1, A(m,n-1)) if m > 0 and n > 0}
```

Complete the Scheme implementation of the Ackermann function.

```
(define (A m n)
  (cond
     ((equal? m 0)
        (+ n 1)
     )
     ((and (> m 0) (equal? n 0))
        (A (- m 1) 1)
     )
     ((and (> m 0) (> n 0))
        (A (- m 1) (A m (- n 1)))
     )
  )
)
```

Example query:
?- mother-in-law(mona, X).
X = marge; **(motherinlaw(M,C) :- parent(M,S), married(S,C), female(M).)**
Example query:
?- sister-in-law(X, homer).
X = patty;
X = selma;
sibling(X,Y) :- parent(P,X), parent(P,Y), X\=Y.
sisterinlaw(S,I) :- motherinlaw(M,I), mother(M,S), not(married(I,S)), female(S).
sisterinlaw(S,I) :- fatherinlaw(M,I), father(M,S), not(married(I,S)), female(S).
sisterinlaw(S,I) :- sibling(X,I), married(X,S), female(S).
Example query:
?- brother-in-law(herb, X).
X = marge;
sibling(X,Y) :- parent(P,X), parent(P,Y), X\=Y.
brotherinlaw(S,I) :- motherinlaw(M,I), mother(M,S), not(married(I,S)), male(S).
brotherinlaw(S,I) :- fatherinlaw(M,I), father(M,S), not(married(I,S)), male(S).
brotherinlaw(S,I) :- sibling(X,I), married(X,S), male(S).

```
class ExceptionalQuestion {
    public static void main(String[] args)
    {
        try{
            System.out.println("Start");
            ExceptionalQuestion x = new ExceptionalQuestion();
            x.aMethod();
            System.out.println("After method.");
        }
        catch (SomeException error)
        {
            System.out.println("main's catch");
         }
         finally
         {
             System.out.println("main's finally");
         }
         System.out.println("End");
    }

    public void aMethod() throws SomeException {
        try{
            System.out.println("In aMethod");
            throw new SomeException();
        }
        catch (SomeException error)
        {
            System.out.println("aMethod's catch");
        }
        finally
        {
            System.out.println("aMethod's finally");
        }
    }
}
```
Start
In aMethod
aMethod's catch
aMethod's finally
After method.
main's finally
End