
This is a 120 minute, CLOSED notes, books, etc. exam.
ASK if anything is not clear.
WORK INDIVIDUALLY.
Strategy: Scan the entire exam first. Work on the easier ones before the harder ones. Don't waste too much time on any one problem. Show all work on the space provided. Write your name on each page. Check to make sure you have 9 pages.

Question	Points	Score
1	5	
2	5	
3	5	
4	5	
5	5	
6	10	
7	10	
8	15	
9	15	
10	5	
11	10	
12	10	
13	15	
Total:	115	

1. (5 points) Select **all** of the following true statements about JavaScript.
   A. JavaScript lacks classes, and hence cannot be considered an object-oriented programming language.
   B. Inside of a function that is not part of an object, `this` will refer to the global object, unless it was invoked with the keyword `new`, in which case it refers to a new object being created.
   C. Functions in JavaScript are *closures*; that is, they remember variables from the scope where they were created.
   D. JavaScript is a purely functional programming language.
   E. Every object has a `prototype` field, which is another object; if you add properties to this object, then they are available to all objects created with the same constructor.

2. (5 points) Select **all** of the following true statements about Inform 7 and *domain specific languages* (DSLs).
   A. DSLs use "natural language" to make programming easier for less computer-savvy users.
   B. Inform 7 is a declarative, logical, and functional programming language.
   C. "Natural language" is used to write programs in Inform 7, meaning that programs look like a human language such as English.
   D. Inform 7 is an example of a DSL for creating web-based videogames.
   E. DSLs are languages specialized for certain tasks; for instance SQL is a DSL for interacting with databases.

3. (5 points) Select **all** of the following true statements about Ruby's `method_missing`, JavaScript's object proxies, and metaprogramming.
   A. Object proxies in JavaScript are a proposed *Reflection API* allowing for advanced metaprogramming in JavaScript.
   B. *Reflection APIs* allow for introspection and self-modification; *Intercession APIs* allow special *traps* or *hooks* to alter the behavior of objects.
   C. Metaobjects are functions that dictate the behavior of other objects.
   D. JavaScript's object proxies are an example of a metaobject protocol, with handler objects that serve as the metaobjects for special "proxy objects".
   E. Ruby's `method_missing` is a trap that is invoked whenever an unknown method is called.

4. (5 points) Select **all** of the following true statements about Prolog.
   A. Prolog uses a *depth-first* search strategy.
   B. Prolog is a logical and declarative programming language, often used in artificial intelligence.
   C. The cut operator `!` prevents the Prolog engine from backtracking past the cut.
   D. Depending on its use, the cut operator may be an optimization that does not affect the results (a so-called "red-cut"), or it may be a "green-cut" that changes the results of the program.
   E. Relationships in Prolog are implicitly mutual, so that if `friend(joe,bob)` is true, `friend(bob,joe)` is also true.

5. (5 points) Select **all** of the following true statements about type systems.
   A. Type systems may be useful to catch errors before executing the program.
   B. Tools such as Eclipse (or other IDEs) can leverage types to learn more details about a program and provide better tips to developers.
   C. Compilers leverage typing information to provide more efficient code.
   D. Almost all type systems sometimes prevent valid programs from running.
   E. JavaScript is a statically typed language, meaning that it catches errors only at runtime.

6. (10 points) Consider the following JavaScript program:

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.honk = function() { console.log("honk!"); }
}
Car.prototype.honk = function() { console.log("Meep!"); }

var car1 = new Car("Chevy", "Nova");
var car2 = new Car("Tesla", "Model S", 2014);
var car3 = Car("Ford", "Explorer", 2001); // Forgot to call "new"

car1.honk();
delete car2.honk;
car1.honk();
car2.honk();
car3.honk();
console.log(model);
```

(a) Select **all** of the following true statements.

A. This code throws an error when creating `car1`, since an argument is missing.

B. This code throws an error when trying to create `car3`, since the programmer forgot to call `new`

C. This code throws an error when calling `car3.honk()`, since `honk` is not defined for the object `car3`.

D. This code throws an error when calling `car3.honk()`, since `car3` is not defined.

E. This code throws an error when calling `console.log(model)`, because `model` is not defined.

(b) Comment out any lines of the program that cause an error. Now what is the output of this program?

7. (10 points) Consider the following program:

```
function mystery(n) {
  var curPos = 2, lst = [], i;
  for (i=curPos; i<n; i++) lst[i]=i;
  return function() {
    var x;
    while (lst[curPos] === undefined) {
      if (curPos > n) throw new Error("Nothing left");
      curPos++;
    }
    x = curPos;
    while (x < n) {
      delete lst[x];
      x += curPos;
    }
    return curPos;
  }
}

var next = mystery(25);
for (i=0; i<5; i++) {
  console.log(next());
}
```

What is the output of this program? **NOTE:** After the while loop, the value of `lst` will be

```
[ undefined, undefined, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
  15, 16, 17, 18, 19, 20, 21, 22, 23, 24 ]
```

8. (15 points) Consider the following Prolog knowledge base:

```
male(james1).
male(charles1).
male(charles2).
male(james2).
male(george1).

female(catherine).
female(elizabeth).
female(sophia).

parent(charles1, james1).
parent(elizabeth, james1).
parent(charles2, charles1).
parent(catherine, charles1).
parent(james2, charles1).
parent(sophia, elizabeth).
parent(george1, sophia).
```

(a) Define `grandparent(X,GP)`.

(b) Define `sister(X,Sis)`. (Be sure that a woman is not her own sister).

(c) Define `cousin(X,Cuz)`. (Be sure that no one is their own cousin).

9. (15 points) Write an ANTLR grammar for parsing (simplied) JSON objects. You need to handle objects, numbers, and strings only. For simplicity, you may assume that all keys are alphabetic only, that all numbers are non-negative integers, and that strings only use double quotes.

   Your grammar should handle the following cases (one per line):

```
{}
{name: "walter white", accountBalance: 80000000}
{name: "sherlock", partner:{name: "watson", jobTitle:"MD"}}
```

   Starter code is given below:

```
grammar Json;

WS      : [ \t\r\n]+ -> skip ; // ignore whitespace

STRING  : '"' (~["])* '"' ;
```

10. (5 points) Consider the following JavaScript program:

```
function Reindeer(name) {
  this.name = name;
}
var dasher = new Reindeer("Dasher");
var dancer = new Reindeer("Dancer");
var rudolph = new Reindeer("Rudolph");

rudolph.nose = "red";
Reindeer.prototype.nose = "black";

console.log(dancer.nose);
console.log(rudolph.nose);

var olive = Reindeer("The Other Reindeer"); // Forgot to call new
console.log(olive);
console.log(olive.nose);
console.log(name);
```

Comment out any lines of the program that cause an error. What is the output of this program now?

11. (10 points) Consider the following JavaScript program:

```
var globalVar = 0;
function makeAdder(x) {
  return function (y) {
    return x + y + globalVar;
  }
}
var addTwo = makeAdder(2);
console.log(addTwo(9));
```

What is the output? Also, describe **in step-by-step detail** how the interpreter resolves the value of `globalVar` and how that relates to scoping.

12. (10 points) Show the output of the following Ruby program:

```
class LinkedList
  attr_accessor :val, :next_node
  def initialize (val, next_node)
    @val = val
    @next_node = next_node
  end
  def get_node(n)
    if n == 0 then @val
    elsif @next_node then @next_node.get_node(n-1)
    else nil
    end
  end
  def each_node &blk
    puts "Reading #{@val}..."
    yield @val
    puts "Finished with #{@val}."
    @next_node.each_node &blk if @next_node
  end
  def method_missing(method_name)
    # The regular expression matches any string that starts with 'it'
    # followed exclusively by decimal digits.
    if method_name.to_s.match(/^it\d+$/) then
      s = method_name.to_s
      s.slice!("it") # Removes 'it' from s
      return get_node(s.to_i)
    end
    puts "#{method_name} is not a method for this object"
  end
end

def make_list lst
  lst.reverse!
  linked_list = prev_node = nil
  lst.each do |x|
    linked_list = LinkedList.new(x, prev_node)
    prev_node = linked_list
  end
  linked_list
end

ll = make_list [9,4,6,12]

sum = 0
ll.each_node do |v|
  sum += v
end
puts sum

puts ll.it3
puts ll.it2
```

13. (15 points) Write Prolog rules to determine all of the ways to make change for a given amount. For example, if your call `change(7,Q,D,N,P)`, the result should produce the following solutions:

    - `Q=0`, `D=0`, `N=1`, `P=2`.
    - `Q=0`, `D=0`, `N=0`, `P=7`.

    Do not forget to use the `is` operator where appropriate. It is OK if your solution produces duplicate results.

```
change(0, 0, 0, 0, 0).
change(Amount, Quarters, Dimes, Nickels, Pennies) :-
```