cs1120: Exam 2

Due: Thursday Nov 29 at 3:30pm (in class)

UVA ID (e.g., wrw6y) : ANSWER KEY

Directions

Work alone. You may not discuss these problems or anything related to the material covered by this exam with anyone except for the course staff between receiving this exam and turning it in.

Open resources. You may use any books you want, lecture notes, slides, your notes, and problem sets. You may ***not*** use PyCharm or Eclipse (or Java or Python), but it is not necessary to do so. You may also use external non-human sources including books and web sites (but your answers should ***not*** reference any procedures not found in the course notes or book). If you use anything other than the course books, slides, and notes, cite what you used. You may not obtain any help from other humans other than the course staff.

Answer well. Answer all questions 1-8 (question 0 is your UVA ID in two places, which hopefully everyone will receive full credit for), and optionally answer question 9.

You may either: (1) print out this exam and write your answers on it or (2) write your answers directly into the provided Word template and print the result out. Whichever one you choose, you must turn in your answers printed **on paper** and they must be clear enough for us to read and understand. You should not need more space than is provided to write good answers, but if you want more space you may attach extra sheets. If you do, make sure they are clearly marked.

The questions are not necessarily in order of increasing difficulty, so if you get stuck on one question you should continue on to the next question. There is no time limit on this exam, but it should not take a well-prepared student more than a few hours to complete. It may take you longer, though, so please do not delay starting the exam. There is no valid excuse (other than a medical or personal emergency) for running out of time on this exam.

No "snow jobs". If you leave a question **blank**, you will receive ~**30%** of the points for it. If you have no idea and waste our time with long-winded guessing, we will be less sanguine and the grading will be more sanguine. :-)

Use any procedure from class. In your answers, you may use any Python or Java procedure that appears in the lecture notes or in the book without redefining it (e.g., length, filter, sort, map, etc.). If there are multiple similar names (e.g., map vs. list-map, len vs. length), use whichever you like. Do ***not*** pull random procedures from the Internet.

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler. Your programs will be judged for correctness, clarity and elegance, but you will not lose points for trivial errors (such as missing a closing parenthesis).

UVA ID *again* (e.g., wrw6y) : ANSWER KEY

Your Scores

0	1	2	3	4	5	6	7	8	EC	Total
10	10	10	10	10	6	6	10	10	0	82

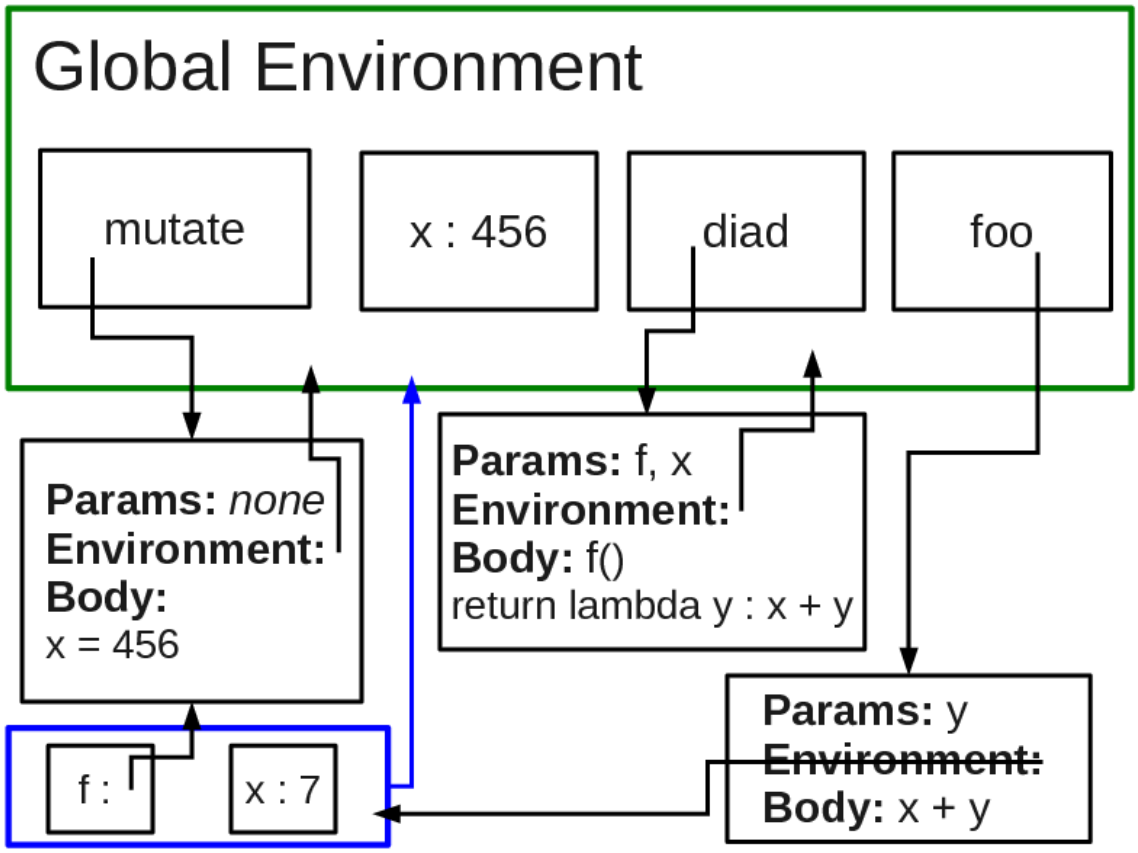
(Your scores are recorded on the second page so that they are not visible to other students when tests are distributed or passed back. *We always use cover sheets on our TPS reports. Didn't you get that memo?*)

Question 0. Write your UVA ID in the two required places. Staple your exam together with an actual staple. We're serious: if you turn it in without a staple, or if some pages are loose or missing (etc.), you will lose points on Question 0. Incorrectly collated exams take longer to grade.

Question 1: Imperative Programming and the Environment Model.
This question asks you two questions related to Python and the Environment Model. Consider the following sequence of imperative Python commands:

```
x = 123
def mutate():
    x = 456
def diad(f, x):
    f()
    return lambda y : x + y
foo = diad(mutate, 7)
```

Consider the Environment diagram below which results after executing those commands. Complete the diagram by filling in all of the blanks. Note that at least one of the blanks requires you to draw an arrow.



What single argument should be passed to **`foo`** so that it returns **`457`** as its output?

450

The change to "x" performed by "f" inside "diad" changes the "x" in the global environment. Mistaking this makes "1" a tempting, but still incorrect, answer.

Question 2: Computability.

In Double Adams' *The Hitchhiker's Guide to the Galaxy*, a large computer called Deep Thought was constructed to learn the Answer to the Ultimate Question of Life, the Universe and Everything. It turns out that the answer is **42**.

Consider the following decision problem definition:

The **python-program-to-output-ultimate-answer-exists?** problem takes no input. It returns true if at least one Python program exists that prints out exactly and only the Answer to the Ultimate Question of Life, the Universe and Everything (i.e., 42). It returns false if no Python programs exist that prints out exactly the required Answer.

Is the **python-program-to-output-ultimate-answer-exists?** problem decideable or undecideable?

It is **decideable**.

Why or why not?

We can use the **Proof By Construction** technique from Slide #13 of Class Notes #09 to prove that there is a Python program that returns 42 by showing one:

```
def python_program_that_outputs_ultimate_answer():
        return 42
```

Since such a program exists, we can make:

```
def python_program_to_output_ultimate_answer_exists():
        return True     # we know at least one exists, from above!
```

Since we have a computer program that implements python-program-to-output-ultimate-answer-exists?, it is **computable** (i.e., **decideable**).

This problem asked you to apply the knowledge we covered in class and distinguish between one particular implementation of the a problem and whether or not that problem is decideable. It's tempting to think "oh, to solve this my implementation would have to iterate over all possible Python programs, and that would take infinitely long, so it can't be done!". However, decideability asks about **any** implementation, not just the one you are initially thinking of – so even my "cheap" implementation (which is hard-coded to return the right answer) qualifies.

Question 3: Computability.

In Double Adams' *The Hitchhiker's Guide to the Galaxy*, a large computer called Deep Thought was constructed to learn the Answer to the Ultimate Question of Life, the Universe and Everything. It turns out that the answer is **42**.

Consider the following decision problem definition:

The **outputs-ultimate-answer?** problem takes a Python program source code *S* as input. It returns true if evaluating *S()* (i.e., running *S* on no arguments) produces exactly the Answer to the Ultimate Question of Life, the Universe and Everything (i.e., 42). It returns false otherwise.

Is the **outputs-ultimate-answer?** problem decideable or undecideable?

This is **un**decideable.

Why or why not?

The short answer is that it is exactly the same as the **evaluates-to-3** problem from Slide #06 of Class Notes #18, but with 3 replaced by 42. Since that problem was undecideable, this one is as well.

The long answer is that if we could solve this problem we could solve the halting problem. So the proof proceeds by **Contradiction** (as per Slide #07 or Class Notes #18). We know that a halting problem solver, halts?, is non-sensical. Assume we could solve outputs-ultimate-answer?. Now we can solve halts:

```
def halts(P):
    def widget():
        P()
        return 42
    return outputs-ultimate-answer(widget)
```

The only way outputs-ultimate-answer(widget) can return True is if P() halted. So we've solved the halting problem. But that's impossible, so outputs-ultimate-answer must not be computable (i.e., must **not be decideable**).

Question 4: Object-Oriented Programming and Inheritance.
Consider the following Python definition representing an old landline telephone:

```
public class Telephone {
      public static void call(Number number) {
            usePhoneNetworkToCall(number);
      }
      public static void billing(boolean isIncomingCall) {
            if (!isIncomingCall) {
                  billCustomerForCall(); // only bill for outgoing calls
            }
      }
      public static void playRingtone(Number callerNumber) {
            phonePlayRingtone("generic ringtone") ; // always same
      }
}
```

Write a Java definition for a **Cellphone** class. The **Cellphone** class:

1. is a subclass of **Telephone**
2. overrides the **billing** method to bill the customer for all calls, both incoming and outgoing
3. adds a **myRingtones** state variable (i.e., non-static field) that is a HashMap mapping Numbers to Strings
4. adds a constructor to explicitly initialize the **myRingtones** state variable to be empty for each newly constructed **Cellphone**
5. adds an **addRingtone** method that takes two arguments: a telephone number and a string (the ringtone) and updates the **myRingtones** HashMap with that information
6. extends the **playRingtone** method to check the **myRingtones** dictionary: if the **callerNumber** has an associated ringtone, it plays that. Otherwise, it calls the **playRingtone** method of its superclass.

```
public class Cellphone extends Telephone {
  public static void billing(boolean isIncomingCall) {
    billCustomerForCall();
  }
  public HashMap<Number,String> myRingtones;
  public Cellphone() { /* Constructor! */
    myRingtones = new HashMap<Number,String>();
  }
  public static void addRingtone(Number n, String rt) {
    myRingtones.put(n, rt);
  }
  public static void playRingTone(Number callerN) {
   if (this.myRingtones.containsKey(callerN)) {
      phonePlayRingtone(myRingtones.get(callerN));
    } else
      super.playRingtone(callerN);
  }
}
```

Question 5: Interpreters and Lazy Evaluation.

For the next two questions, you are given a procedure definition. Your answer should describe its asymptotic running time when evaluated using (a) Mini Python (the language defined in PS7), and (b) a hypothetical Mini Python with Lazy Evaluation (as defined in the class notes and lectures: a value will not be computed until it is needed). You should include:

1. the answer
2. a clear supporting argument
3. a definition of all variables you use in your answer.

Remember to omit constant factors (e.g., 2n+7 is in O(n), so just say O(n), not O(2n+7)). Assume that Mini Python handles all of the Python syntax used here.

```
def ulric(x):
    if x > 5:
        return ulric(x-1) - ulric(x-1)
    else:
        return 77
```

(a) Running time in Mini Python:

Theta(2^x)

Consider:
ulric(8)
ulric(7) ulric(7)
ulric(6) ulric(6) ulric(6) ulric(6)
ulric(5) ulric(5) ulric(5) ulric(5) ulric(5) ulric(5) ulric(5) ulric(5)

It has **twice as many** recursive calls for each x above 5. So ulric(9) will do twice as much work as ulric (8). So the running time is Theta($2^{(X-5)}$), which is the same as Theta(2^X).

(b) Running time in Lazy Mini Python:

Theta(2^x)

Even though the ultimate answer is either 77 (if x <= 5) or 0 (for all other x), Lazy Mini Python will still compute all of the recursive calls because we need the result of ulric(x-1) for the subtraction. We humans know it's pointless to do the computation, but Lazy Mini Python, as we defined it, is not that smart. Slide #31 of Class Notes #20 shows that we deThunk objects for arithmetic.

Question 6: Interpreters and Lazy Evaluation.

```
def neisser(a,b):
      if (b > 0):
            return neisser( neisser(a, b+1) , b-1 )
      else:
            return 0
```

(a) Running time in Mini Python:

This procedure does not terminate in Mini Python!

Consider neisser(5,6). Since 6>0, it returns neisser(neisser(5,7),5). The inner call to neiesser(5,7) calls neisser(5,8) ... which never ends! As written, neisser is **not an algorithm.**

(b) Running time in Lazy Mini Python:

Theta(b)

In Lazy Mini Python, we defer assignments until their values are needed. This is shown on Slide #06 of Class Notes #20. Calling a function involves assigning the actual arguments to the formal parameters. So neisser(5,6) will return neisser(thunk,5), which will return neisser(thunk, 4) ... down to neisser(thunk, 0) which returns 0. Since we never force the evaluation of parameter a (i.e., we never deThunk a), we never have to make all of those unending recursive calls, so neisser just counts down b and **runs in linear time**.

Question 7: Typechecking and Mini Python.

Define a **typeIf(expr, env)** Java procedure that checks the type of a Mini Python **if-then-else** statement. In the class notes we neglected to provide a type checking procedure for "if". Recall that according to the Mini Python grammar, the "else:" branch must always be provided. (For more general information on "if", see Chapter 11.3.2 of the Course Book. Note that the Course Book is building an interpreter for Scheme, not an interpreter for Python.) Every "if" expression will thus always have both an then-branch and also an else-branch (i.e., both a consequent and an alternate). Your code should statically check that the predicate expression has primitive type boolean. In addition, in order for an "if" expression to be type correct, the then-branch and the else-branch (i.e., the consequent and the alternate) must produce values of the *same type*. The type of an "if" expression is the type that is shared by the then-branch and else-branch.

```
// Want something like typeDef, Slide #12, Class #22
public Type typeIf(ArrayList<Object> expr,
                   Environment env) {
      // expr = ["IF", COND, THEN-BRANCH, ELSE-BRANCH]
      Type tCond = typeCheck(expr.get(1), env);
      if (tCond.isPrimitive() &&
            tCond.getName().equals("boolean")) {
            Type tThen = typeCheck(expr.get(2), env);
            Type tElse = typeCheck(expr.get(3), env);
            // Cannot use equals(), must use matches()
            // from Slide #29 of Class #21 to see if
            // tThen and tElse are the same.
            if (tThen.matches(tElse)) {
                  return tThen;  // or tElse ...
            } else {
                  return new ErrorType("if has then-branch
and else-branch with different types");
            }
      } else {
            return new ErrorType("if conditional must have
type boolean");
      }
}
```

Question 8. Networking, Latency, Bandwidth.

Suppose you send three one-bit packets over a network. The first is sent at time t=0, arrives at time t=4. The second is sent at time t=3 and arrives at time t=9. The third is sent at time t=6 and arrives at time t=11. We might represent this example in Java as:

```
int [] sent      = {0, 3, 6};      // bit departure times
int [] arrived   = {4, 9, 11};     // bit arrival times
int n            = 3;              // number of bits
```

The average latency for those three packets is 5.0 seconds per bit. The total bandwidth for that three packet sequence is 3/11bits per second. You can assume that **n>0** and **sent[i] < arrived[i]** for all **0 <= i < n**. Define a Java procedure **averageLatency** that computes the average latency for ***any*** such set of one-bit packets. Define a procedure **totalBandwidth** that computes the total bandwidth for ***any*** such sequence of one-bit packets (hint: they may arrive out of order!). Do not worry about rounding or floating point numbers: assume int division works perfectly. Your procedures must work for all inputs, not just the example above.

```
public int averageLatency(int [] sent, int [] arrived, int n) {
        // latency is transit time: "arrived – sent"
        int totalLatency = 0;
        for (int i=0; i<n; i=i+1) {
                int thisLatency = arrived[i] – sent[i];
                totalLatency = totalLatency + thisLatency;
        }
        return totalLatency / n;
}

public int totalBandwidth(int [] sent, int [] arrived, int n) {
        // bandwidth is "bits per second", so we'll calculate the
        // total number of bits sent (which is just n) divided by the
        // total transit time. Since the bits may be sent out of
        //  order, we'll have to find the earliest starting time and
        // the latest arrival time.
        int tStart = sent[0];
        int tEnd = arrived[0];
        for (int i = 0; i < n; i = i + 1) {
                if (sent[i] < tStart) tStart = sent[i];
                if (arrived[i] > tEnd) tEnd = arrived[i];
        }
        int totalTime = tEnd – tStart;
        return n / totalTime;
}
```

Question 9. 1 point if not blank. Do you think your performance on this Exam will fairly reflect your understanding of the material in the second half of the course? If not, explain why. Is there anything else you'd like to say about the course or the material thus far?

Since the exam did not ask any questions related to mock turtlenecks, liberal arts trivia, or the best way to dodge candy, I do not feel it accurately reflects the majority of CS 1120.