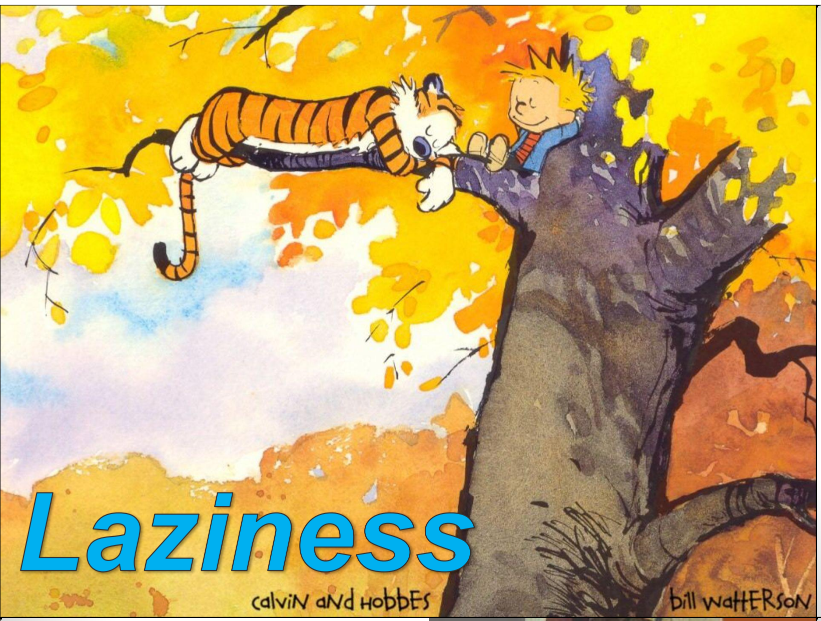


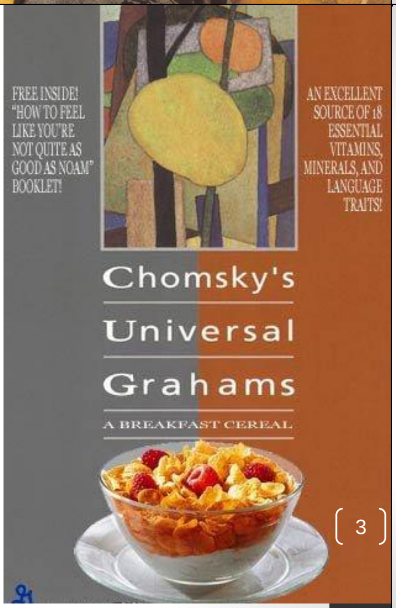
# Laziness

## One-Slide Summary

- Building an **interpreter** is a fundamental idea in computing. Eval and Apply are **mutually recursive**.
- The most complicated parts of **meval** are the handling of functions (**def**) with **evalCall**.
- The most complicated part of **evalCall** is handling the new environment. You must add variables to that environment corresponding to the arguments, and then **apply** the procedure body in that new environment.
- In **lazy evaluation**, a value is not computed until it is needed. A **thunk** is a piece of code that performs a delayed computation.

## Outline

- Eval
- Apply
- Lazy
- Thunk

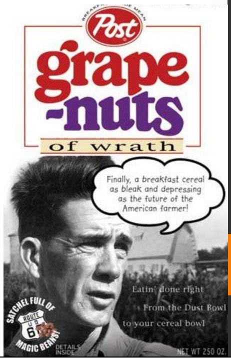

FREE INSIDE! "HOW TO FEEL LIKE YOU'RE NOT QUITE AS GOOD AS NOAM" BOOKLET!
AN EXCELLENT SOURCE OF 18 ESSENTIAL VITAMINS, MINERALS, AND LANGUAGE TRAITS!
Chomsky's Universal Grahams
A BREAKFAST CEREAL

## Problem Set 7

- You will be writing an **interpreter** for **MiniPython**, a simple version of Python
- Your interpreter will be written in **Java**
- This demonstrates that Java is at least as powerful as MiniPython
  - ***Why?***
- It turns out that MiniPython is also at least as powerful as Java (!)
  - ***Why?***

## meval review

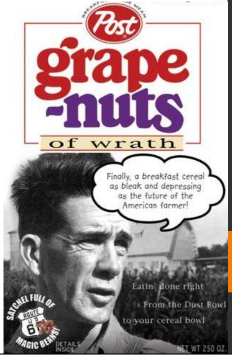
```
public static Object meval(Object expr, Environment env){
  if(expr instanceof ArrayList<?>) {
    ArrayList<Object> exp = (ArrayList<Object>) expr;
    Object operation = exp.get(0);
    if (operation.equals(“if”)) {
      return evalIf(exp,env);
    } else if (operation.equals(“callE”) {
      return evalCall(exp,env);
    }
    // ... Apply other evals here
  } else {
    return evalPrimative(expr,env);
  }
```


Post
grape-nuts
of wrath
Finally, a breakfast cereal as bleak and depressing as the future of the American farmer!
Eatin' done right
From the Dust Bowl to your cereal bowl

## meval Review

```
public static Object meval(Object expr, Environment env){
  if(expr instanceo
    ArrayList<Obje                   ayList<Object>) expr;
    Object operation          p.get(0);
    if (operation   quals(“if”)) {
      return evalIf(exp,env);
    } else if (operation.equals(“callE”) {
      return evalCall(exp,env);
    }
    // ... More evals here
  } else {
    return evalPrimative(expr,env);
  }
```

How do we apply an `if` statement ?


Post
grape-nuts
of wrath
Finally, a breakfast cereal as bleak and depressing as the future of the American farmer!
Eatin' done right
From the Dust Bowl to your cereal bowl

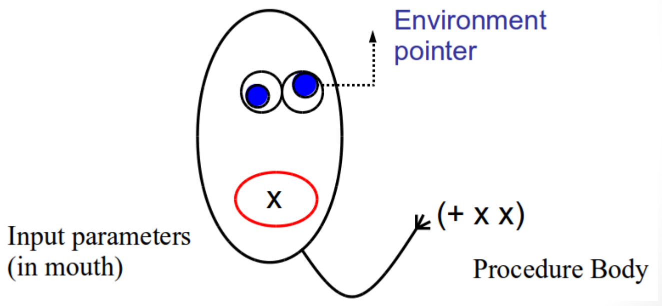
```
public static Object evalIf(ArrayList<Object> exp,
                            Environment env) {
    // Evaluate the condition
    Object cond = meval(exp.get(1),env);

    // Check that the condition is a Boolean.
    if (!(cond instanceof Boolean)) {
      throw new EvalError("Expected a boolean");
    }
    // If the condition was true, evaluate the true clause
    if ((Boolean) cond) {
      return evalSequence(exp.get(2),env);
    }
    // If the condition was false, evaluate the false clause
    else {
      return evalSequence(exp.get(3),env);
    }
  }
```

## Applying Primitives

- Primitive operations like +, -, >, <, if, while, assign, etc. follow a similar pattern:
  - Evaluate subexpressions
  - Perform operation on the evaluated subexpressions
  - Do something with the result (return, update env, etc.)
- Calling procedures is *more complex* because we need to create a new environment for the procedure and fill in the *formal parameters* with the given values

## Implementing Procedures

- What do we need to record?



## Evaluating Procedures

```
public static Object evalCall(ArrayList<Object> exp, Environment env) {
  // Look up the defined procedure in the environment
  Procedure proc = ((Procedure)env.lookupVar((String)exp.get(1));

  // Get the formal argument values passed to the function
  ArrayList<Object> passedArgs = (ArrayList<Object>) exp.get(2);

  // Get the expected argument names from the procedure
  ArrayList<String> expectedArgNames = proc.args;

  // Create a new environment with the procedure's env as the parent
  Environment newEnv = new Environment(proc.env);

  // Evaluate then bind each argument value to the argument name
  for (int i = 0; i < passedArgs.size(); i++)
  newEnv.addVar(expectedArgNames.get(i),meval(passedArgs.get(i),env));

  // Evaluate the function body with the new environment
  return meval(proc.body, new_env);
}
```

## Procedure Class

```
public class Procedure {
  public ArrayList<String> args; //Formal parameters
  public ArrayList<Object> body; //Body code

  //Env to use when executing this procedure
  public Environment env;
  // Constructor
  public Procedure(ArrayList<String> args_,
                   ArrayList<Object> body_,
                   Environment       env_) {
    args = args_;
    body = body_;
    env = env_;
  }
}
```

## Liberal Arts Trivia: Geography

- This island nation in southeast Asia is located about 20 miles off the southern coast of India. It is home to 20 million people (mostly Sinhalese and Tamils), is a center of the Buddhist religion, and possesses rich tropical forests. during WWII it was used as a base for Allied forces against the Japanese Empire. It was known as Ceylon before 1972.

## Liberal Arts Trivia: Italian Literature

- This Florentine poet of the Middle Ages is called *il Sommo Poeta*. His central work, the *Divinia Commedia*, is often considered the greatest literary work composed in the Italian language and is a masterpiece of world literature. He is often called the Father of the Italian Language: he wrote the *Commedia* in the early 14th century in a new language he called "Italian" based on the regional dialect of Tuscany with some Latin and other bits thrown in.
- Bonus: Who guides him in Hell and Purgatory?

## Implemented Interpreter!

What's missing?

Special forms:
  lambda functions
  for loops
  negative numbers
  and, or, not
Built-in types:
  floating point numbers,
  strings, lists, etc.

But it is still just as powerful as Python or Java!

## Lazy Evaluation

- Procrastination finally pays off

## Lazy Evaluation

- **Lazy Evaluation**: don't evaluate expressions until their value is really needed
  - We might save work this way, since sometimes we don't need the value of an expression
  - We might change the meaning of some expressions, since the order of evaluation matters
- Not a wise policy for problem sets (all answer values will always be needed!)

## Lazy Examples

**Python**

```
def ohnoes(x):
  return x / 0

def doWork(x):
  y = ohnoes(5)
  return x * 2

doWork(10)
ZeroDivisionError:
```

**Lazy Python**

```
def ohnoes(x):
  return x / 0

def doWork(x):
  y = ohnoes(5)
  return x * 2

doWork(10)
20
```

- Ordinary men and women, having the opportunity of a happy life, will become more kindly and less persecuting and less inclined to view others with suspicion. The taste for war will die out, partly for this reason, and partly because it will involve long and severe work for all. Good nature is, of all moral qualities, the one that the world needs most, and good nature is the result of ease and security, not of a life of arduous struggle. Modern methods of production have given us the possibility of ease and security for all; we have chosen, instead, to have overwork for some and starvation for others. Hitherto we have continued to be as energetic as we were before there were machines; in this we have been foolish, but there is no reason to go on being foolish forever.

Bertrand Russell, *In Praise of Idleness,* 1932

(co-author of *Principia Mathematica*, proved wrong by Gödel's proof)

## How do we make our evaluation rules *lazier?*

Example: variable assignment

```
    x = 9 * 9 * nine(9) * y * 9999999999999…9
```

To ***eagerly*** evaluate an assignment:

1. evaluate the expression to the right of the '='
   (let's say the result is 9)
2. update the environment with the result for the lval
   (add x → 9 to the environment)

**How can we make this lazier?**

## Answer: *do it later*

Example: variable assignment

```
    x = 9 * 9 * nine(9) * y * 9999999999999…9
```

To ***lazily*** evaluate an assignment:

~~1. evaluate the expression to the right of the '='~~

1. **Delay** evaluating the expression until x is needed

~~2. update the environment with the result for the lval~~

2. Remember the environment during the assignment so we can use it again during evaluation

…but how do we implement this?

## Liberal Arts Trivia: Canadian Literature

- In this 1908 book, the title character is a talkative red-haired orphan. She moves to the village of Avonlea to live with farmers Matthew and Marilla Cuthbert. She becomes bosom friends with Diana Barry and has a complex relationship with Gilbert Blythe. Her vivid imagination and cheerful outlook often land her in trouble.
- Bonus: Name the setting's Canadian Province.

## Liberal Arts Trivia: Biology

- This generic term is used for many plants in the genus *Allium*. The plant is edible, grown underground as a vertical shoot that is used for food storage. It is one of the oldest vegetables, and is available fresh, frozen, canned, carmelized, pickled, powdered, chopped, and dehydrated. They are rarely eaten alone, and can be sharp, spicy, tangy, pungent, mild or sweet. Tissue from this plant is often used in science education to demonstrate microscope usage because it has large cells. In Bronze age settlements, traces have been found near the fig and date going back to 5000 BCE. The ancient Egyptians worshiped it, believing that its spherical shape and concentric rings symbolized eternal life; it was used in Egyptian burial rituals (e.g., placed in the eye sockets of Ramesses IV).

## Liberal Arts Trivia: Neuroscience

- This medical visualization technique is most commonly used to visualize the internal structure and function of the body. Notably, it uses no ionizing radiation, but instead uses powerful fields to align the hydrogen atoms in water in the body. Radiofrequency fields are used to alter the alignment of the hydrogen atoms, which can then be detected by a scanner. The process was first used on humans in 1977.

## Evaluation of Expressions

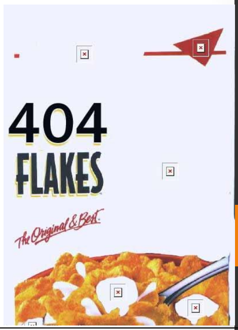
- Applicative Order ("eager evaluation")
  - Evaluate all subexpressions before apply
  - Python, MiniPython, Java
- Normal Order ("lazy evaluation")
  - Evaluate arguments when the value is needed
  - Algol60 (sort of), Haskell, Miranda, **LazyPython**
- "Normal" Python order is **not** "Normal Order"

## Delaying Evaluation

- We need to record everything we will need to evaluate the expression later
- After evaluating the expression, record the result for reuse
- A thunk is a piece of code that performs a delayed computation
- When building an interpreter you have some choice as to what computation will be delayed and for how long. (You can lazily evaluate many types of expressions)

## I Thunk I Can

```
public class Thunk {
  private ArrayList<Object> expr;
  private Environment env;
  private boolean evaluated;
  private Object value;

  public Thunk(ArrayList<Object> expr_, Environment env_) {
    this.expr = expr_; this.env = env_;
    this.evaluated = false;
  }
  public Object value() {
    if (!this.evaluated) {
      this.value = forceEval(expr, env);
      this.evaluated = true;
    }
    return this.value;
  }
}
```



## Forcing Evaluation

```
public class Thunk {
  private ArrayList<
  private Environmen
  private boolean ev
  private Object val

  public Thunk(Array
              Environment env_) {
    this.expr = expr_; this.env = env_;
    this.evaluated = false;
  }
  public Object value() {
    if (!this.evaluated) {
      this.value = forceEval(expr, env);
      this.evaluated = true;
    }
    return this.value;
  }
}
```

```
public Object forceEval(expr, env) {
  value = meval(expr, env);
  if (value instanceof Thunk)
    return value.value();
  else
    return value;
}
```

## Lazy Assignment

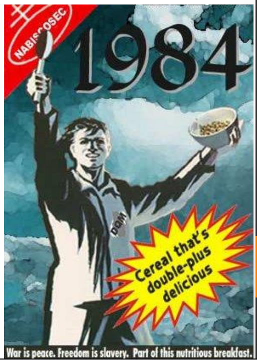
```
public static Object evalAssign(ArrayList<Object> exp,
                                Environment env) {
  String id = (String) exp.get(1); //Name of Variable
  Object res = (Object) meval(exp.get(2),env); //Evaluate
  env.addVariable(id, res); //Add to environment
  return null; //No return value for assignments
}
```

```
public static Object lazyEvalAssign(ArrayList<Object> exp,
                                Environment env) {
  String id = (String) exp.get(1); //Name of Variable
  Object res = (Object) new Thunk(exp.get(2),env); //Skip Eval
  env.addVariable(id, res); //Add to environment
  return null; //No return value for assignments
}
```

## What else needs to change?

- At some point we need to call value() on a thunk
  - But when?

Hint: where do we need *real* values, instead of Thunks?


NABISCOSEC
1984
Cereal that's double-plus delicious
War is peace. Freedom is slavery. Part of this nutritious breakfast.

## Primitive Operations!

- Option 1: redefine primitives to work on thunks
- Option 2: assume primitives need values of all their arguments

## Unthunk Youself

```
public Object deThunk(ArrayList<Object> expr) {
  //How is this different than forceEval()?
  if (expr instanceof Thunk)
    return expr.value();
  else
    return expr;
}

public static Object evalAdd(… exp, … env) {
  Object v1 = meval(deThunk(exp.get(1)), env);
  Object v2 = meval(deThunk(exp.get(2)), env);
  //(Omitting error handling for non-ints)
  return (Integer)v1 + (Integer)v2;
}
```

## Short-circuit evaluation

- Java and Python are **eager evaluation** languages
  - But they can still skip unnecessary computation

```
int x = -9;

if (x > 0 & computePItoNDigits(99999999999999999) < x) {
  System.out.println(“X is between 0 and PI”);
} else {
  System.out.println(“X isn’t between 0 and PI!”);
}

>>> Terminates in 2353 seconds
```

- What part of the if condition could we skip? Why?

## Short-circuit evaluation

- Substituting ‘&&’ for ‘&’ results in ***short-circuit evaluation*** of the condition

```
int x = -9;

if (x > 0 && computePItoNDigits(9999999999999999) < x) {
  System.out.println(“X is between 0 and PI”);
} else {
  System.out.println(“X isn’t between 0 and PI!”);
}

>>> Terminates in 0.0001 seconds
```

- If the first half of an and statement is false, the entire boolean statement must be false. What about ‘or’ ?

## Homework

- Read Chapter 13
- PS 7 due next Tuesday