One-Slide Summary

- **Object-Oriented Programming** encapsulates state and methods together into objects. This hides implementation details (cf. inheritance) while allowing methods to operate on many types of input.
- **Python** and **Java** are both universal, imperative, object-oriented languages but differ several ways including treatment of whitespace and typing.
- Building an interpreter is a fundamental idea in computing. **Eval** and **Apply** are **mutually recursive**.
- We can write a Java program that acts as an interpreter for programs written in **MiniPython**, a simplified version of the **Python** language.

Outline

- Object Lessons
- Ali G
- Interpreters
- Eval
- Apply

Who was the first object-oriented programmer?

> “So, by a vote of 8 to 2, we have decided to ship the hundredth revision completely, and we should have it finished this afternoon.”

Implementing Interpreters

By the word operation, we mean any process which alters the mutual relation of two or more things, be this relation of what kind it may. This is the **most general definition**, and would include all subjects in the universe. Again, it might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine... Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.

**Ada, Countess of Lovelace**, around 1843
Learn Languages: Expand Minds

Languages change the way we think.
The more languages you know, the more different ways you have of thinking about (and solving) problems.

Music with Movable Type?
- *Odhecaton*: secular songs published in 1501 Venice
  - First book of music ever printed using movable type,
  - Hugely influential both in publishing in general, and in dissemination of the Franco-Flemish musical style.
  - Seeing business potential, in 1498 Petrucci had obtained an exclusive 20-year license for all printing activities related to music anywhere in the Venetian Republic.
  - He printed two parts on the right-hand side of a page, and two parts on the left: four singers or instrumentalists could read from the same sheet.
  - Petrucci's publication not only revolutionized music distribution, it contributed to making the Franco-Flemish style the international musical language of Europe for the next century.

Computability in Theory and Practice
(Intellectual Computability Discussion on TV Video)
(I hope this works!)

Ali G Problem
- **Input**: a list of 2 numbers with up to \(d\) digits each
- **Output**: the product of the 2 numbers

Is it computable?
Yes – a straightforward algorithm solves it. Using elementary multiplication techniques it is \(O(d^2)\)

Can real computers solve it?
Ali G was Right!

- Theory assumes ideal computers:
  - Unlimited, perfect memory
  - Unlimited (finite) time
- Real computers have:
  - Limited memory, time, power outages, flaky programming languages, etc.
  - There are many computable problems we cannot solve with real computer: the actual inputs do matter (in practice, but not in theory!)

Liberal Arts Trivia: Biology

- This family of non-venomous serpents contains the longest snake in the world. They have teeth, heat-sensing organs, and ambush prey. They kill by a process of constriction: sufficient pressure is applied to the prey to prevent it from inhaling, and the prey succumbs to asphyxiation and is swallowed whole.

Liberal Arts Trivia: Chemistry

- This element is a ductile metal with very high thermal and electrical conductivity. When pure and fresh it has a pinkish or peachy color, but it turns green with age (oxidation). It has played a significant role in the history of humanity. In the Roman era it was usually mined on Cyprus; hence the provenance of its modern name (Cyprium to Cuprum).

Implementing Interpreters

Inventing a Language

- Design the grammar
  - What strings are in the language?
  - Use BNF to describe all the strings in the language
- Make up the evaluation rules
  - Describe what everything the grammar can produce means
- Build an evaluator
  - A procedure that evaluates expressions in the language
Is this an exaggeration? (SICP, p. 360)

It is no exaggeration to regard this as the most fundamental idea in programming:

The evaluator, which determines the meaning of expressions in the programming language, is just another program.

To appreciate this point is to change our images of ourselves as programmers. We come to see ourselves as designers of languages, rather than only users of languages designed by others.

Environmental Model of Evaluation

- To evaluate an expression, evaluate all the subexpressions and apply the value of the first subexpression to the values of the other subexpressions.
- To apply a procedure to a set of arguments (evalCall in PS7), evaluate the body of the function in a new environment.
  - To construct this environment, make a new frame with an environment pointer that is the environment of the procedure that contains places with the formal parameters bound to the arguments.

The Plan – Front End

- We are given a string like "(( 5 +  6) - 10)"
- First, we remove whitespace and parse it according to our MiniPython grammar.
- We will represent MiniPython expressions internally as either
  - Primitive elements, like "5" or "+
  - String ArrayLists representing functions and arguments, like {"-", {"+", "5", "6"}, "10"}
  - (Plus some other details in PS7.)

The Plan – Back End

- So now we have Java representations of MiniPython expressions, such as:
  - "123"  # 123
  - ["-", ["+", "5", "6"], "10"]  # (5 + 6) - 10
- We want to evaluate such expressions. We will write a method, meval(), that does this.
  - meval("123") = 123
- We'll also write evalCall() to apply functions created by def.
  - evalCall("pow", "2", "4") = 16

Evaluation and Environments

- Recall the Environment Model for Python
  - Go back to Class 13 or Book Chapter 9 if not!
- Let's think about some meval inputs and outputs together:
  - meval("12") = 12
  - meval("x") = ??
- So meval will need an environment:
  - meval("12",env) = 12
  - meval("x",env) = value-of-x-in-env
So we might try to write `meval` by cases:

```java
public static Object meval(Object expr, Environment env){
  if (expr instanceof ArrayList<?>)
    return evalCall(expr, env);
  Object operation = exp.get(0);
  if (operation.equals("if") {
    return evalIf(expr, env);
  } else if (operation.equals("callE") {
    return evalCall(expr, env);
  } else {
    return evalPrimative(expr, env); // "3", "x", etc.
  }
}
```

What does this look like in real Java?

```java
public static Object meval(Object expr, Environment env){
  if (expr instanceof ArrayList<?>)
    return evalCall(expr, env);
  Object operation = exp.get(0);
  if (operation.equals("if") {
    return evalIf(expr, env);
  } else if (operation.equals("callE") {
    return evalCall(expr, env);
  } else {
    return evalPrimative(expr, env); // "3", "x", etc.
  }
}
```

Tracing Simple Evaluations

```java
System.out.println(Parser.parse("55"));
55 // or very close
```

Evaluation Trace:
```
meval("55", env)
```
```
evalPrimative("55")
55
```

What about something more complicated like (5 + 6)?

Parses to {"+", "5", "6"}
meval Basics

- So we might try to write meval by cases:

```
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")) { // [if, [<, x, 2],...]
            return evalIf(exp, env);
        } else if (operation.equals("callE")) { // [pow, 2, 4]
            return evalCall(exp, env);
        } else {
            return evalPrimitive(expr, env); // "3", "x", etc.
        }
    }
}
```

meval Basics

- So we might try to write meval by cases:

```
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")) { // [if, [<, x, 2],...]
            return evalIf(exp, env);
        } else if (operation.equals("callE")) { // [pow, 2, 4]
            return evalCall(exp, env);
        } else {
            return evalPrimitive(expr, env); // "3", "x", etc.
        }
    }
}
```

meval Basics

- So we might try to write meval by cases:

```
else if (operation.equals("def")) {
    return evalDef(expr, env);
} else if (operation.equals("if")) {
    return evalIf(expr, env);
} else if (operation.equals("return")) {
    return evalReturn(expr, env);
} else if (operation.equals("assign")) {
    return evalAssign(expr, env);
} else if (operation.equals("print")) {
    return evalPrint(expr, env);
} else if (operation.equals("call")) {
    evalcall(expr, env); return null;
} else if (operation.equals("while")) {
    return evalWhile(expr, env);
} else if (operation.equals("callE")) {
    return evalCall(expr, env);
} else if (operation.equals("get")) {
    return evalGetListElt(expr, env);
} else if (operation.equals("list")) {
    return evalList(expr, env);
} else if (operation.equals("sizeof")) {
    return evalSize(expr, env);
} else if (operation.equals("*")) {
    return evalPrimitive(expr, env);
} // also "+", "/", ">", "<", and <=
```

Liberal Arts Trivia: Philosophy

- In the philosophy of mind, this is used to describe views in which the mind and matter are two ontologically separate categories. In this, neither mind nor matter can be reduced to each other in any way. This is typically opposed to reductive materialism. A well-known example of this is attributed to Descartes, holding that the mind is a nonphysical substance.
Liberal Arts Trivia: Chemistry

- This exothermic chemical process involves the rapid oxidation of a fuel material, releasing light, heat and various reaction products. Fuels of interest often include organic compounds and hydrocarbons. Slower oxidation processes, like rusting, are not part of this process.

Liberal Arts Trivia: Statistics

- A t-test is a statistical hypothesis test in which the test statistic has a *This* distribution if the null hypothesis is true. The *This* distribution arises when estimating the mean of a normally distributed population when the sample size is small. It was first published by William Gosset in 1908 while he worked at a Guinness Brewery in Dublin. The brewery forbade the publication of research by its staff members (!), so he published the paper under a pseudonym.

```python
else if (operation.equals("def")) {
    return evalDef(expr, env);
} else if (operation.equals("if")) {
    return evalIf(expr, env);
} else if (operation.equals("return")) {
    return evalReturn(expr, env);
} else if (operation.equals("assign")) {
    return evalAssign(expr, env);
} else if (operation.equals("print")) {
    return evalPrint(expr, env);
} else if (operation.equals("callf")) {
    evalCall(expr, env); return null;
} else if (operation.equals("while")) {
    return evalWhile(expr, env);
} else if (operation.equals("callE")) {
    return evalCall(expr, env);
} else if (operation.equals("get")) {
    return evalGetListE(expr, env);
} else if (operation.equals("list")) {
    return evalList(expr, env);
} else if (operation.equals("sizeof")) {
    return evalSize(expr, env);
} else if (operation.equals("==")) {
    return evalEquals(expr, env);
} else {
    return null;
}
```

Applying Primitives

How about: (5 + 6) ?

The expression parses like this: ["+", "5", "6"]
Applying Primitives

```java
public static Object evalAdd(ArrayList<Object> exp, Environment env) {
    Object v1 = Evaluator.meval(exp.get(1), env);
    Object v2 = Evaluator.meval(exp.get(2), env);
    if (v1 instanceof Integer && v2 instanceof Integer) {
        return (Integer)v1+(Integer)v2;
    } else if(v1 instanceof ArrayList<?> && v2 instanceof ArrayList<?>) {
        ArrayList<Object> list1 = (ArrayList<Object>) v1;
        ArrayList<Object> list2 = (ArrayList<Object>) v2;
        ArrayList<Object> new_list = (ArrayList<Object>) list1.clone();
        new_list.addAll(list2);
        return new_list;
    } else throw new EvalError("Cannot add " + v1 + " and " + v2);
}
```

Applying Primitives

```java
public static Object evalAdd(ArrayList<Object> exp, Environment env) {
    Object v1 = Evaluator.meval(exp.get(1), env);
    Object v2 = Evaluator.meval(exp.get(2), env);
    if (v1 instanceof Integer && v2 instanceof Integer) {
        return (Integer)v1+(Integer)v2;
    } else throw new EvalError("Cannot add " + v1 + " and " + v2);
}
```

Applying Primitives

```java
public static Object evalAdd(ArrayList<Object> exp, Environment env) {
    Object v1 = Evaluator.meval(exp.get(1), env);
    Object v2 = Evaluator.meval(exp.get(2), env);
    if (v1 instanceof Integer && v2 instanceof Integer) {
        return (Integer)v1+(Integer)v2;
    } else throw new EvalError("Cannot add " + v1 + " and " + v2);
}
```

Applying Primitives

```java
public static Object evalAdd(ArrayList<Object> exp, Environment env) {
    Object v1 = Evaluator.meval(exp.get(1), env);
    Object v2 = Evaluator.meval(exp.get(2), env);
    if (v1 instanceof Integer && v2 instanceof Integer) {
        return (Integer)v1+(Integer)v2;
    } else throw new EvalError("Cannot add " + v1 + " and " + v2);
}
```
### Applying Primitives

```java
public static Object evalAdd(ArrayList<Object> exp, Environment env) {
    Object v1 = Evaluator.meval(exp.get(1), env);
    Object v2 = Evaluator.meval(exp.get(2), env);
    if (v1 instanceof Integer && v2 instanceof Integer) {
        return (Integer)v1 + (Integer)v2;
    } else if (v1 instanceof ArrayList && v2 instanceof ArrayList) {
        ArrayList<Object> list1 = (ArrayList<Object>) v1;
        ArrayList<Object> list2 = (ArrayList<Object>) v2;
        ArrayList<Object> new_list = (ArrayList<Object>) list1.clone();
        new_list.addAll(list2);
        return new_list;
    }
    throw new EvalError("Cannot add "+ v1 + " and "+ v2);
}
```

### Tracing

How about: \((5 + x)\) ?

The expression parses like this: \([-, [+, 5, x], 10]\)

```java
meval([-,[+, 5, x],10], env)
```

### More Tracing

The expression parses as: \([-, [+], 5, 6, 10]\)

```java
meval([-,[+, 5, 6, 10], env]
```

### Implementing evalCall

```java
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);
```
Implementing evalCall

```java
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(env.get1(), proc.exp);
  // Evaluate then bind each expected argument name to the passed argument value
  for (int i = 0; i < passedArgs.size(); i++)
    newEnv.addVar(expectedArgNames.get(i), meval(passedArgs.get(i), newEnv));
  // Evaluate the function body with the new environment
  return meval(proc.body, newEnv);
}
```

Get the definition of "add2" from the environment.

```java
meval("add2")
```
Implementing evalCall

```java
public static Object evalCall(ArrayList<Object> exp, Environment env) {
    // Look up the defined procedure in the environment
    Procedure proc = ((Procedure)env.lookupVar(new 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 them 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, newEnv);
}
```

Homework

- Work on Problem Set #7
  - If we give you a long time before a problem set is due, what does that imply?