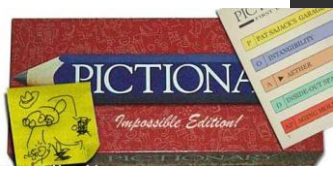
Types of Types



One-Slide Summary

- In **lazy evaluation**, expressions are not evaluated until their values are needed.
- A **type** is a (possibly infinite) set of values.
- Each type supports a set of valid operations.
- Types can be latent or manifest, static or dynamic, strong or weak.
- We can change the MiniPython interpreter to support manifest (program visible) types.

Outline

- Administration
- Lazy Evaluation Recap
- Types
- MiniPython with Manifest Types

Problem Set 8

- Understand and modify a dynamic web application
- Already posted

Problem Set 9

- Team requests are due **Thursday, November 15th**
  - (email Wes before midnight)
- Descriptions are due **Tuesday, November 20th**

Lazy Evaluation Recap

- 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
- Change the Evaluation rule for Application
- Use **thunks** to delay evaluations

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
}
```

Liberal Arts Trivia: Cognitive Science

- This philosophy of mind dominated for the first half of the 20th century. It developed as a reaction to the inadequacies of introspectionism. In it, all things which organisms do – including acting, thinking and feeling – should be regarded as actions or reactions, usually to the environment. It holds that there are no philosophical differences between publicly observable processes (actions) and privately observable processes (thinking and feeling).
- Bonus: B.F. Who?

Liberal Arts Trivia: Civil Rights

- The landmark 1967 Supreme Court case *Loving v. Virginia* declared Virginia's anti-miscegenation statue, the "Racial Integrity Act of 1924", unconstitutional. This effectively ended laws preventing what?

Types

Type Review

- A Type is a (possibly infinite) set of values
- You can do some things with some types, but not others
- Each Type has associated valid operations

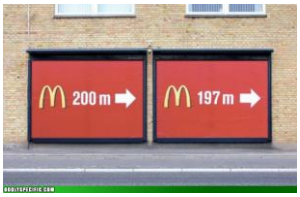
```
5 * 2       //Evaluates to an int
5 * "hi"    //Evaluates to what? hihihihihi? 0?
```

- Why are types useful? They help identify errors and define correct behavior.

Video Break

- Java contains statically checked, manifest types.
- Python contains dynamically checked, latent types.
- However, both are **strongly typed**. Permissible operations on types are well-defined and strongly enforced.
- What can happen if we our language fails to follow well-defined or intuitive type rules?

What do we need to do to change our MiniPython interpreter to provide manifest types?



Liberal Arts Trivia: Media Studies

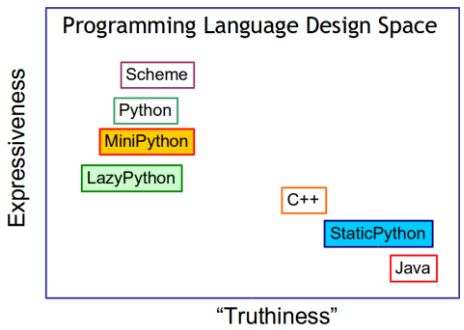
- This technique in film editing combines a series of short shots into a sequence of condensed narrative. It is usually used to advance the story as a whole and often to suggest the passage of time.

Liberal Arts Trivia: European History

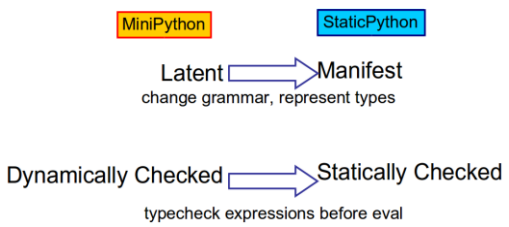
- This relatively slender, sharply pointed sword was popular in Europe in the 16th and 17th centuries. It is mainly used for thrusting attacks. It is characterized by a complex hilt, which protects the hand wielding it. The etymology may derive the word from the Greek ραπίζειν "to strike."

Liberal Arts Trivia: United Kingdom History

- This United Kingdom Tory prime minister was most famous for his military work during the Peninsular Campaign and the Napoleonic Wars. He was nicknamed the "Iron Duke" because of the iron shutters he had fixed to his windows to stop pro-reform mobs from breaking them – as an MP he was opposed to reform. It is unclear whether the well-known beef tenderloin, pate and puff pastry dish is named after him.



Types of Types



Manifest Types

Need to change the grammar rules to include types in definitions and parameter lists:

Old Grammar:

```
DefStmt     := def Name (ArgList) : Sequence
ArgList     := ε | Arg Arglist
Arg         := Name
```

```
def compare(ascending, x, y):
  if(ascending):
    return x < y
  else:
    return y < x
```


Manifest Types

Need to change the grammar rules to include types in definitions and parameter lists:

New Grammar:

```
DefStmt     := def Name (ArgList) : Type :: Sequence
ArgList     := ε | Arg Arglist
Arg         := Name : Type
```

```
def compare(ascending : Type, x : Type, y : Type) : Type ::
  if(ascending):
    return x < y
  else:
    return y < x
```

But what is "Type?"

Types in MiniPython

```
Type          ::= PrimitiveType
                | ListType
                | ProcedureType
PrimitiveType ::= int | boolean
ListType      ::= Type ListType | []
ProcedureType ::= (ListType -> Type)
```

But what does all this mean?

Examples

```
Type          ::= PrimitiveType
                | ProcedureType
                | ListType
PrimitiveType ::= int | boolean
ListType      ::= Type ListType | []
ProcedureType ::= (ListType -> Type)
```

9

>>> int

9 + 9

>>>([int, int] -> int)

compare(true, 10, 5)

>>>([boolean, int, int] -> boolean)

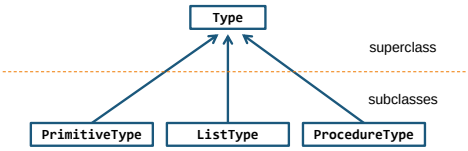
pow(2,3) - 4

>>>([([int, int] -> int) , int] -> int)

(Apologies to monochromatic slider printers)

Representing Types

```
Type          ::= PrimitiveType | ProcedureType | ListType
PrimitiveType ::= int | boolean
ListType      ::= Type ListType | []
ProcedureType ::= (ListType -> Type)
```




```
public class Type {
  //Default return values for subclasses
  private isPrimitiveType()     {return false;}
  private isListType()          {return false;}
  private isProcedureType()     {return false;}

  public static Type fromString(String s) {
    ArrayList<Object> ast = Parser.parse(s);
    return Type.fromParsed(ast.get(0));
  }

  public static Type fromParsed(Object operation) {
    //Inspect operator and create type
  }
}
```

PrimitiveType

```
public class PrimitiveType extends Type {
  public String name;
  public PrimitiveType(String name_) {
    this.name = name_;
  }
  @Override      //Redefined from class Type
  public isPrimitiveType() {return true;}

  public boolean matches(Type other) {
    return other.isPrimitiveType() &&
           other.name.equals(this.name);
  }
}
```


ListType

```
public class ListType extends Type {
  public ArrayList<Type> contents;
  public ProcedureType(ArrayList<Type> contents_) {
    this.contents = contents_;
  }
  @Override      //Redefined from class Type
  public isListType() {return true;}

  public boolean matches(Type other) {
    // How does this work?
  }
}
```


ListType Continued

```
public boolean matches(Type other) {
  if (other.isListType() &&
      this.contents.size() == other.contents.size())
  {
    for (int i = 0; i < this.contents.size(); i++) {
      Type me = this.contents.get(i);
      Type other = other.contents.get(i);
      if (!me.matches(other))
        return false; //Type mismatch!
    }
    return true;
  }
  else return false;
}
```


ProcedureType

```
public class ProcedureType extends Type {
  public ArrayList<Type> args;
  public Type returnType;
  public ProcedureType(String name_,
                       ArrayList<Type> args_,
                       Type returnType_) {
    this.name = name_;
    this.args = args_;
    this.returnType = returnType_;
  }
  @Override      //Redefined from class Type
  public isPrimitiveType() {return true;}

  public boolean matches(Type other) {
    // How can we implement this comparison?
  }
}
```


ProcedureType

```
public class ProcedureType extends Type {
  ...
  public boolean matches(Type other) {
    return other.isProcedureType() &&
      this.args.matches(other.args) &&
      this.returnType.matches(other.returnType);
  }
}
```


Homework

- Problem Set 7 due
- Thursday: Problem Set 9 team requests due