

Introduction to the Lox Language

Online interpreter

lox.nanmu.me/en/

```
1 // This is a comment
2 print "Hello, world!";
```

C style syntax for familiarity.

A high-level language

Lox, while very small, is a high-level language.

Most similar to *JavaScript* and *Lua*

And shares some **features** with them:

- dynamic typing
- automatic memory management

Dynamic Typing

Variables can store values of any type, and can change type at runtime.

Static typing requires more work to learn and implement

Automatic Memory Management

Memory management tends to be error-prone and tedious.

Lox uses a garbage collector to automatically reclaim memory that is no longer in use.

Using such as **reference counting** or **tracing garbage collection**.

Data types

- Booleans

```
1 true;  
2 false;
```

- numbers (all numbers are double-precision floating point)

```
1 1234;  
2 12.34;
```

- strings

```
1 "This is a string.;"
```

- nil

```
1 nil;
```

Expressions

arithmetic

```
1 add + me;  
2 subtract - me;  
3 multiply * me;  
4 divide / me;
```

- **operands**: the things on the left and right
- **binary**: because there are two operands
- **infix**: because the operator is in between
- one arithmetic operator can be both **infix** and **prefix**

comparison and equality

```
1 less < than;
2 lessThan ≤ orEqual;
3 greater > than;
4 greaterThan ≥ orEqual;
```

logical

```
1 !true;  
2 !false;
```

Works like control flow

```
1 true and false  
2 true or false
```

- if both true > return right side
- if not both true > return left side

Vice versa for `or`

precedence

Same precedence as C

```
1 var average = (min + max) / 2;
```

Statements

Statements

An expression produces a **value**, a statement produces an **effect**.

```
1 print "hello, world!";  
  
1 "An expression";  
  
1 {  
2     print "This is a block.";  
3     print "It contains multiple statements.";  
4 }
```

Variables

Variables

```
1 var imAVariable = "value";  
2 var iAmNil;
```

```
1 var breakfast = "bagels";  
2 print breakfast;
```

Control Flow

Control Flow

```
1 if (condition) {  
2     print "yes";  
3 } else {  
4     print "no";  
5 }
```

```
1 var i = 1;  
2 while (i < 10) {  
3     print i;  
4     a = a + 1;  
5 }
```

```
1 for (var i = 0; i < 10; i = i + 1) {  
2     print i;  
3 }
```

Functions

Functions

```
1 makeBreakfast(rice, egg, soySauce);  
2 makeRice();
```

```
1 fun printSum(a, b) {  
2     print a + b;  
3 }
```

- argument: an actual value
- parameter: the variable in the function definition

Closures

functions are *first-class* values

```
1 func addPair(a, b) {  
2     return a + b;  
3 }  
4  
5 fun identity(a) {  
6     return a;  
7 }  
8  
9 print identity(addPair)(1, 2);
```

Closures

Scoping also works as expected

```
1 fun outerFunction(a) {  
2     fun innerFunction(b) {  
3         return "local";  
4     }  
5     return innerFunction;  
6 }
```

Closures

```
1  fun returnFunction() {  
2      var outside = "outside";  
3  
4      fun inner() {  
5          print outside;  
6      }  
7      return inner;  
8  }  
9  
10 var fn = returnFunction();  
11 fn();
```

Classes

Why object oriented

1. People still use object-oriented programming for many tasks.
2. It **works***

Why is lox object oriented

Lox already has scope and closures, so it's basically a functional language.

You'll end up using OOP at some point

Knowing how to make one is useful

Classes or prototypes

There are two main approaches to making objects

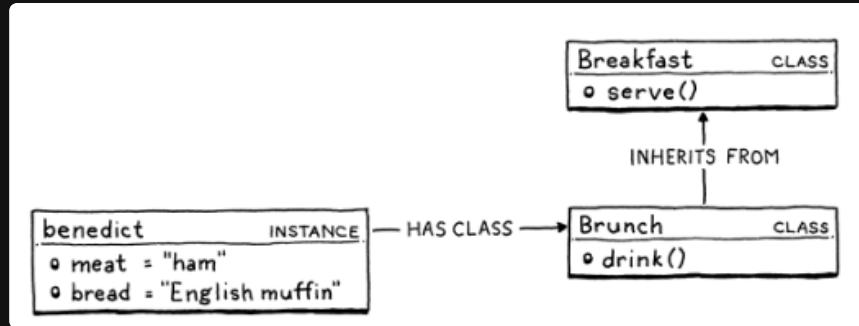
1. Classes
2. Prototypes

Classes

In class based languages, the main concepts are

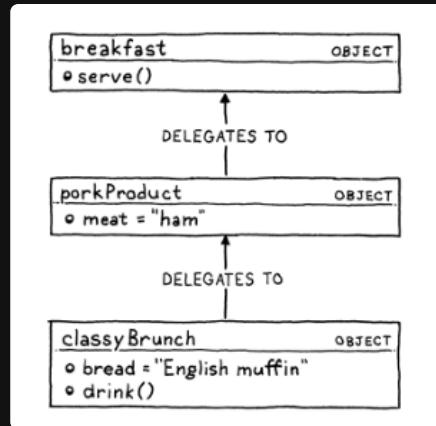
1. instances, which store the **state** of an object
2. classes, which contains the **methods and inheritance** chain

To call a method on an instance, you go inside the instance, find the class, then find the method.



Prototypes

In prototype based languages, there are only objects and no classes



And objects can directly "inherit" from other objects (*delegate*)

Simpler to implement, but people are more familiar with classes

Inlox

```
1 class Breakfast {  
2     cook() {  
3         print "Cooking breakfast!";  
4     }  
5  
6     serve(who) {  
7         print "Here is your breakfast, " + who + "!";  
8     }  
9 }  
10  
11 var someVariable = Breakfast; // creates a class object  
12 someFunction(Breakfast);  
13  
14 var breakfast = Breakfast(); // creates an instance  
15 breakfast.cook();
```

Initialization

The idea of object oriented programming is to **bundle** data and behavior together

Like some other dynamically typed languages, Lox let's you *freely* add properties to objects

```
1 breakfast.meat = "ham";
2 breakfast.bread = "white bread"
3
4 class Breakfast {
5     serve(who) {
6         print "Here is your " + meat + " on " + bread + ", " + who;
7     }
}
```

Initialization

`init()` is called when an instance is created

```
1  class Breakfast {  
2      init(meat, bread) {  
3          this.meat = meat;  
4          this.bread = bread;  
5      }  
6  
7      serve(who) {  
8          print "Here is your " + this.meat + " on " + this.bread + ", " + who;  
9      }  
10 }
```

Inheritance

Inheritance is how object oriented languages work

```
1 class Brunch < Breakfast {  
2     drink() {  
3         print "Here is your beer";  
4     }  
5 }
```

Where `Brunch` is a derived class or *subclass*, and `Breakfast` is a base class or *superclass*.

Inheritance

Every method defined in `Breakfast` is also available in `Brunch`.

```
1 var benedict = Brunch("bacon", "english muffin");
2 benedict.serve("student");
```

Even the `init()` method is inherited, but you can access the superclass's version with `super`

```
1 class Brunch < Breakfast {
2     init(meat, bread, drink) {
3         super.init(meat, bread);
4         this.drink = drink;
5     }
6 }
```