Some Legacy Requirements

```
function f(x,x) {console.log(x)}
f(1,2);   //logs: 2

function g(x) {
  var x;
  console.log(x);
}
g(1);    //logs: 1

function h(x) {
  function x() {return 2}
  console.log(x());
}
g(funtion() {return 1});    //logs: 2
```

Proposal Part 1

- Simple (ES≤5.1) parameter lists introduce “var bindings” in the top-level scope of the function
- All ES≤5.1 rules apply
  - Duplicated parameter names
  - Parameter names may be same as var or function declaration
  - Magic arguments object

Proposal Part 2

- If a parameter list uses *any* parameter syntax introduced in ES6, new rules apply
  - Destructuring parameters
  - Default value initializers
  - Rest parameter
- New rules:
  1. Parameter lists introduce “let bindings” in the top-level scope of the function
     - A. No duplicate parameter names
     - B. Parameters names may not be the same as any other function top-level declaration
  2. Temporal dead zone rules apply to parameter default value initializers
     - A. Hoisted top-level function/var declarations are initialized after parameter initialization
  3. “Strict” arguments object (copy of actual args, no parameter joining)

New Rules Examples

```
function f(x, x, ...rest) {}
```

Syntax Error: duplicate parameter name (Rule 1.A)

```
function f(x, {a:x, b:y}) {}
```

Syntax Error: duplicate parameter name (Rule 1.A)

```
function f([x]) {var x;}
```

Syntax Error: Redeclaration of parameter X (Rule 1.B)

```
function f([x]) {let x;}
```

Syntax Error: Redeclaration of parameter X (Rule 1.B)

```
function f([x]) { {var x;} }
```

Syntax Error: Redeclaration of parameter X using hoisted var (Rule 1.B)

```
function f([x]) { {let x;} }
```

Valid, redeclaration is in inner block

```
function f([x]) {function x() {}}
```

Syntax Error: Redeclaration of parameter X (Rule 1.B)

```
function f([x]) {class x {}}
```

Syntax Error: Redeclaration of parameter X (Rule 1.B)

New Rules Examples

```
function f(x, y=x) {}
```

Valid, x has been initialized when y's default value expression is evaluated

```
function f(x=y, y}) {}
```

Runtime : ReferenceError exception: y not yet initialized (Rule 2)

```
const a="A";
function f(x=a) {}
```

Valid, a is visibly declared and initialized in the surrounding scope

```
function f(x=a) {var a;}
```

Runtime : ReferenceError exception: "a" not yet initialized (Rule 2.A)

```
function f(x=a) {const a="A";}
```

Runtime : ReferenceError exception: "a" not yet initialized X (Rule 2.A)

```
function a() {return "A"}
function f(x=a()) {}
```

Valid, a is visibly declared and initialized in the surrounding scope

```
function f(x=a()) {function a() {return "A"}}
```

Runtime : ReferenceError exception: "a" not yet initialized X (Rule 2.A)

Issue
Do new function forms get poison pill properties, even when they are not strict?

- Alternatives:
  - Only if strict, just like current Function definitions
  - Always: New forms, do compat. reason to support bogus legacy properties
  - Always: Because we decide that new forms are always strict.