Bard 0.4

Kernel

Atoms

```
;;; Numbers
    0
    2.3
    2/3
    3e23
    #x10ff
    #b1011

;;; names
'Barney
"A name with embedded whitespace"

;;; named constants
nothing
true
false
undefined
```

Lists

A **list** is a value that consists of an ordered sequence of values.

```
;;; list construction
[]
[0 1 [2 3]]
['Barney 'Fred "Wilma and Betty"]

;;; constant literal lists
'(1 2 3 4)
'(Fred Barney)

;;; function calls as lists
(* 2 3 4)
(print 'Barney)
(apply + [2 3 4])
```

Objects

An **object** is a value that consists of a set of key/value pairs.

```
;;; object construction
{}
{'name: 'Fred 'age: 45}
```

Functions

```
;;; anonymous functions
(function (x) x)
(function args args)
(function (x y)
  (+ x y))
```

Variables

```
;;; global variables
bard> (define x 5)
x
bard> x
5

;;; lexical variables
(let ((x 2)
      (y 3))
  (* x y))
```

Closures

A **closure** is a function that contains a lexical environment in which some variables are defined.

```
;;; return an anonymous counter function
;;; that closes over the variable count
(let ((count 0))
  (method ()
    (begin
      (set! count (+ count 1))
      count)))
```

Generic functions

Bard functions are **generic**; that means that a function can have more than one definition. When a function is called, the definition chosen is the one that matches the arguments passed.

```
;;; a function with two methods:

(define (add x:Number y:Number)
  (+ x y))

(define (add x:Name y:Name)
  (append x y))

  bard> (add 2 3)
  5
  bard> (add 'Foo 'bar)
  Foobar
```

Protocols

A **protocol** is an object whose keys are variable and function names, and whose values are variable values and functions. Bard protocols serve as namespaces. All of Bard's built-in features are accessible through variables defined in built-in protocols.

A single standard protocol is bound to the special variable `bard`. Its variables are bound to the built-in protocols that make up Bard's standard library. Each built-in protocol has a name of the form:

```
bard.*protocol-name*.
```

protocol-name is the name of the specific protocol. Examples are:

```
bard.kernel
bard.system
```

Some Bard implementations may support protocols with longer names, like

```
bard.system.network
```

Despite how it might look, Bard's namespaces are not hierarchical. `bard.system.network` is distinct from `bard.system`, not contained within it. This style of naming is just a convention used to organize namespaces so that related ones have related names.

Built-in protocols

```
bard.language
bard.math
bard.system
```