Self Extended Module as Toplevel Object

(Pollution Free Object) Feature #6609

Defining methods on toplevel pollutes Object class and thus *all* objects. This is bad practice, so why allow it? In addition, toplevel object does not provide usual methods expected of a *namespace*.

CURRENTLY

```
self.class  #=> Object

def foo; end
Object.private_instance_methods(false)  #=> [:foo]

define_method(:foo){}
NoMethodError: undefined method ‘define_method’ ...

Foo = 10
const_defined?(:Foo)
NoMethodError: undefined method `const_defined?' ...
```

Many more examples, almost all Module methods can’t be used.

PROPOSAL

Solve both issues in one go by making toplevel object a self extended module instead of current instance of Object which delegates (only a little) to Object class.

```
module Main
  extend self
  # toplevel evaluates as if here
end

self.class  #=> Module
```

Module is *real* namespace.

ISSUES

- **Toplevel methods pollute all objects, which is useless and can potentially cause bugs with meta-programming. e.g.** `private_methods.include?()`**.**
- **Does not act like other namespaces. Can’t define dynamic methods, lookup constants, use callbacks, etc.**

BENEFITS

- **Toplevel freedom! Create DSLs which can be evaluated at toplevel without concern over use of `` `def` ``.**
- **Access to Main from anywhere is easy. `` `Main.binding` `` instead of `` `TOPLEVEL_BINDING` ``.**
- **No one can use bad practice any more.**