The way a system is designed is not always how it is used in practice.

By mining open source software, it is possible to measure how code is being used in practice. We present Codex, a database constructed from over 3 million lines of Ruby code that can be queried across a variety of parameters (e.g., identifier names, method calls, code complexity, and AST nodes) to enable new data-driven applications.

We describe three applications that demonstrate how Codex can help programmers: statistical linting, pattern annotation, and an open source Ruby Gem.

**Statistical Linting**

For example, this line will cause unexpected behavior: `foo = bar.downcase!` We have never seen the function `downcase!` chained after an assignment. The reason is `bar.downcase!` modifies `bar` and returns `nil`, while the non-bang version `bar.downcase` will work as intended.

**Function signatures**

The arguments in this line should be swapped: `arr = Array.new([1,2,3], 2)` Once swapped, `arr` will equal `[[1,2,3], [1,2,3]]` Not all blocks use their return values. For example, this line does nothing: `arr = arr.each{|word| word.downcase}` Instead, changing the method calling the block (replacing each with map) or changing the block's return statement (replacing `downcase` with `downcase!`) would work to modify the array.

**Pattern Annotation**

Codex identifies common idioms and annotates them with a crowd of human workers. Consider the idiom: `sum every element in a list`. It cannot be found in any source of official documentation. However, Codex identifies `inject { |x,y| x + y }` as a common snippet and passes it off to human workers who provide the annotation.

**Ruby Idioms**

Many valuable Ruby idioms are not collated in documentation. For instance, consider the idiom: `concat every element in a list`. It cannot be found in any source of official documentation. However, Codex identifies `inject { |x,y| x + y }` as a common snippet and passes it off to human workers who provide the annotation.

**Ruby Gem**

CodexLib is an open source Ruby Gem generated to factor out particularly common idioms into a new standard library. We found 13 instances of this pattern, which extracts a Class from a given name and also handles namespaces:

```ruby
if var0.const_defined?(var1)
    var0.const_get(var1)
else
    var0.const_missing(var1)
end
```

This can be combined into a single method, `const_full_get(var1)`, and would help those trying to use `const_get` by itself (a problem that has been discussed on StackOverflow).

**Detecting Unlikely Code**

Sometimes programmers program badly; they write incorrect code or violate language conventions. CodexLint points out unlikely components of code, indicating that they may be incorrect or just unconventional. We show users a statistic that represents how many times we saw their code relative to a more common piece of code, which is often interesting even when the unlikely code is not actually incorrect.

We highlight unlikely components of code from these four categories:

1. **Function chaining**
   - For example, this line will cause unexpected behavior: `foo = bar.downcase!`
   - We have never seen the function `downcase!` chained after an assignment.
   - The reason is `bar.downcase!` modifies `bar` and returns `nil`, while the non-bang version `bar.downcase` will work as intended.

2. **Function signatures**
   - The arguments in this line should be swapped: `arr = Array.new([1,2,3], 2)`
   - Once swapped, `arr` will equal `[[1,2,3], [1,2,3]]`

3. **Block return values**
   - Not all blocks use their return values. For example, this line does nothing: `arr = arr.each{|word| word.downcase}`
   - Instead, changing the method calling the block (replacing each with map) or changing the block's return statement (replacing `downcase` with `downcase!`) would work to modify the array.

4. **Identifier types**
   - "array" is an unusual name for a hash
   - "ptr", "ind", and "count" are unusual names for a string

We thank the Stanford University Computer Science Department and the CURIS Program for their support.