contentful

Andreas Tiefenthaler / pxlpnk

Pry

Swiss army knife of a modern Rubyist


WENGER

What is a REPL

R Read

E Eval

P Print

L Loop

```
(loop (print (eval (read))))
```

```
(loop {(puts (eval gets.chomp))})
```

```
loop { puts(eval(gets.chomp)) }
```

```
❯ ruby -e "(loop { (puts (eval gets.chomp ))})"
puts 'Hello'
Hello

puts Hello
-e:1:in `eval': uninitialized constant Hello
(NameError)
  from -e:1:in `eval'
  from -e:1:in `block in <main>'
  from -e:1:in `loop'
  from -e:1:in `<main>'
❯
```

Existing Ruby REPLs

IRB

interactive Ruby

.irbrc file is just Ruby

add your own helpers & customize the shell

Helpful http://irb.tools/

Pry

http://pryrepl.org/ & on GitHub

has very good documentation

Lots of extensions

Ruby 

Recap

Read Eval Print Loop

Powerful tool based on a "simple idea”

Ruby has multiple options: IRB, Pry

An introduction

```
› gem install pry

› echo 'gem "pry"' >> Gemfile
› bundle install

› pry
[1] pry(main)>
```

```
[1] pry(main)> help
Help
  help               Show a list of commands or information about a specific command.

Context
  cd                 Move into a new context (object or scope).
  find-method        Recursively search for a method within a class/module or the current namespace.
  ls                 Show the list of vars and methods in the current scope.
  pry-backtrace      Show the backtrace for the pry session.
  raise-up           Raise an exception out of the current pry instance.
  reset              Reset the repl to a clean state.
  watch              Watch the value of an expression and print a notification whenever it changes.
  whereami           Show code surrounding the current context.
  wtf?               Show the backtrace of the most recent exception.
```

```
> show-doc Array#sort

> ? Array#sort
```

```
[1] pry(main)> show-doc Array#sort

From: array.c (C Method):
Owner: Array
Visibility: public
Signature: sort()
Number of lines: 15

Returns a new array created by sorting self.

Comparisons for the sort will be done using the <=> operator
or using an optional code block.

The block must implement a comparison between a and b, and return
+-1+, when a follows b, 0 when a and b are equivalent, or +1
if b follows a.


See also Enumerable#sort_by.

   a = [ "d", "a", "e", "c", "b" ]
   a.sort                    #=> ["a", "b", "c", "d", "e"]
   a.sort { |x,y| y <=> x }  #=> ["e", "d", "c", "b", "a"]
```

```
> show-source Array#sort

> $ Array#sort
```

```
[1] pry(main)> show-source Array#sort

From: array.c (C Method):
Owner: Array
Visibility: public
Number of lines: 7

VALUE
rb_ary_sort(VALUE ary)
{
    ary = rb_ary_dup(ary);
    rb_ary_sort_bang(ary);
    return ary;
}
```

```
> edit HTTP.get
```

```
> cd []
```

```
[1] pry(main)> cd []
[2] pry(#<Array>):1> whereami
Inside #<Array>.
[3] pry(#<Array>):1>
```

```
[1] pry(main)> cd []
[2] pry(#<Array>):1> whereami
Inside #<Array>.
[3] pry(#<Array>):1> show-source

From: /Users/at/.rbenv/versions/2.2.1/lib/ruby/2.2.0/pp.rb @ line 355:
Class name: Array
Number of monkeypatches: 3. Use the `-a` option to display all available monkeypatches
Number of lines: 13

class Array # :nodoc:
  def pretty_print(q) # :nodoc:
    q.group(1, '[', ']') {
      q.seplist(self) {|v|
        q.pp v
      }
    }
  end

  def pretty_print_cycle(q) # :nodoc:
    q.text(empty? ? '[]' : '[...]')
  end
end
```

```
> ls
> ls --grep sor*
```

```
[4] pry(#<Array>):1> ls
Enumerable#methods:
  all?            each_entry        find       inject   min         one?          slice_when
  chunk           each_slice        find_all   lazy     min_by      partition     sort_by
  collect_concat  each_with_index   flat_map   max      minmax      reduce
  detect          each_with_object  grep       max_by   minmax_by   slice_after
  each_cons       entries           group_by   member?  none?       slice_before
Array#methods:
  &       collect      each_index  inspect              push                    select      to_a
  *       collect!     empty?      join                 rassoc                  select!     to_ary
  +       combination  eql?        keep_if              reject                  shelljoin   to_h
  -       compact      fetch       last                 reject!                 shift       to_s
  <<      compact!     fill        length               repeated_combination    shuffle     transpose
  <=>     concat       find_index  map                  repeated_permutation    shuffle!    uniq
  ==      count        first       map!                 replace                 size        uniq!
  []      cycle        flatten     pack                 reverse                 slice       unshift
  []=     delete       flatten!    permutation          reverse!                slice!      values_at
  any?    delete_at    frozen?     place                reverse_each            sort        zip
  assoc   delete_if    hash        pop                  rindex                  sort!       |
  at      drop         include?    pretty_print         rotate                  sort_by!
  bsearch drop_while   index       pretty_print_cycle   rotate!                 take
  clear   each         insert      product              sample                  take_while
self.methods: __pry__
locals: _  __  _dir_ _ex_  _file_  _in_  _out_  _pry_
```

How can those tools help you?

Pry extended

Commands

Plugins

Commands

Modify the user input

The syntax is more intuitive than building a plugin

They are local to your Pry console and do not require monkey patches

Help to form your workflow

```
Pry::Commands.command /^$/, "repeat lst” do
  _pry_.run_command Pry.history.to_a.last
end

Pry.commands.alias_command 'c', 'continue'
```

Plugins

Allow to extend Pry

Use the power of other libraries and Gems

Help to form your workflow

pry-**byebug**

pry-**exception_explorer**

pry-**macro**

pry-**rails**

Recap

Great tool to navigate code

Documentation at your fingertips

Can be extended and can be customized to shape your workflow

RDD & Debugging

```
puts @current_user.inspect

p @current_user
```

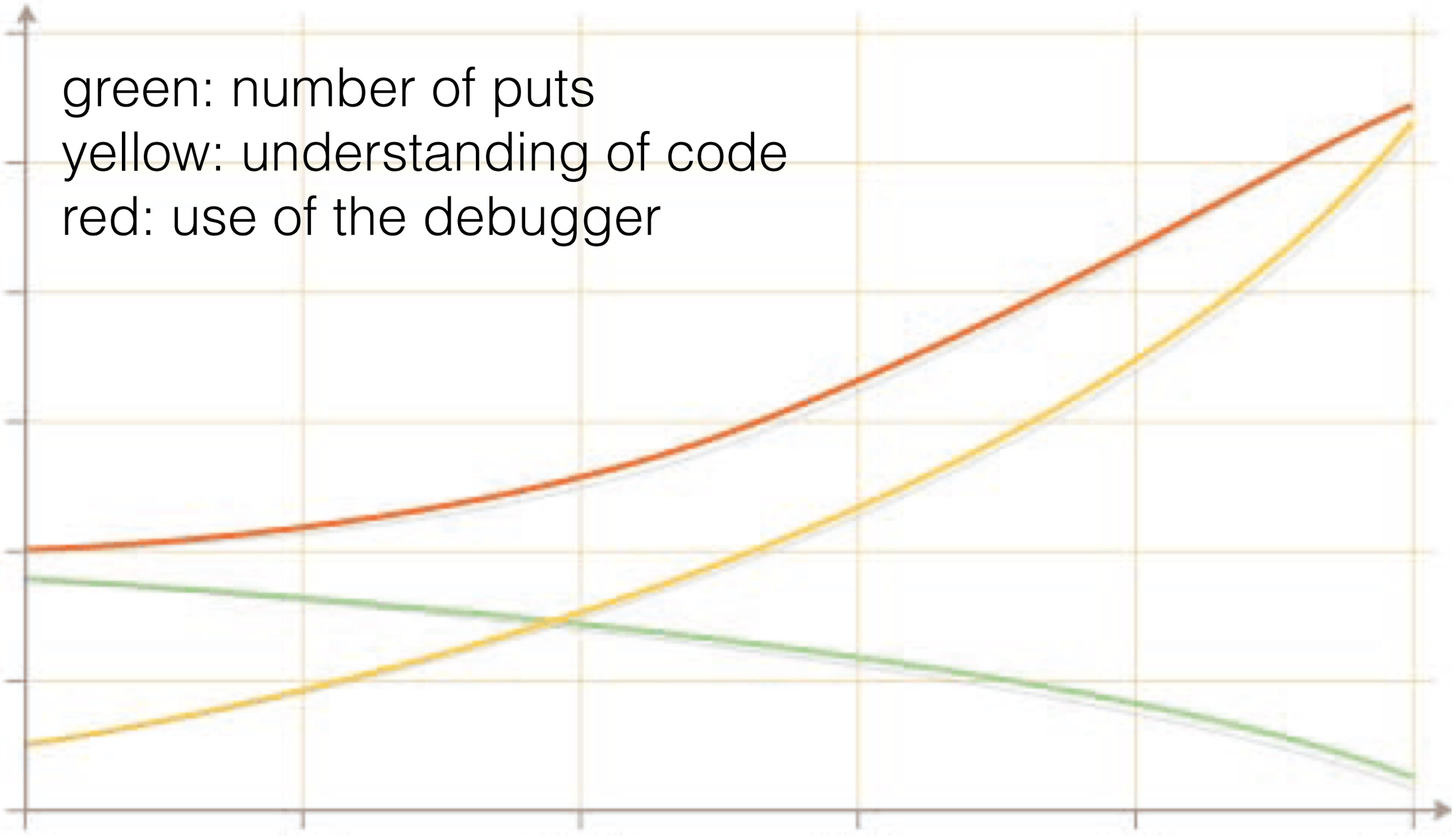
```
if is_this_true?
  puts “Will I be called?”
  some_method
end
```

```
puts “WTF!?!?!?!”
```

Context

State

“Bigger Picture”



Warning: Fictional chart

Debugger

debugger |diːˈbʌgə| (noun)

a computer program that assists in the detection and correction of errors in other computer programs.

Breakpoints

Navigate Code

```
❯ be ruby -rpry lib/user.rb

From: /Users/at/src/private/talks/pry-talk/console/lib/user.rb @ line 16 User#fullname:

    14: def fullname
    15:   binding.pry
 => 16:   first_name + last_name
    17: end

[1] pry(#<User>)> first_name
=> "First"
[2] pry(#<User>)> up

From: /Users/at/src/private/talks/pry-talk/console/lib/user.rb @ line 23 :

    18: end
    19:
    20: __FILE__ == $0
    21: opts = {first_name: 'First', last_name: 'Last'}
    22: user = User.new(opts)
 => 23: user.fullname
```

Optimize Workflow

RDD

REPL Driven Development

Understanding the Problem

Exploring & Teaching

Recap

Debugging is essential and can avoid frustration

A better understanding of state and context

Quicker feedback cycle -> Better understanding

Powerful for beginners and advanced programmers

Future

Random Ideas

REPL driven development with Pry

Conrad Irwin

http://pryrepl.org/

Thank you

Andreas Tiefenthaler / pxlpnk

contentful

Take a REPL with you

```
#!/usr/bin/env ruby

require 'pathname'

ENV['BUNDLE_GEMFILE'] ||= File.expand_path('../../
Gemfile', Pathname.new(__FILE__).realpath)

require 'english'
require 'rubygems'
require 'bundler/setup'
Bundler.require(:default, :development)

$LOAD_PATH << 'lib'
require 'user'

Pry.start
```