URL Mapping with Routes

PyWorks 2008
Mike Naberezny

Maintainable Software

http://maintainable.com

About Me

- http://mikenaberezny.com
- http://maintainable.com
- http://ohloh.net/accounts/mnaberez

Maintainable Software

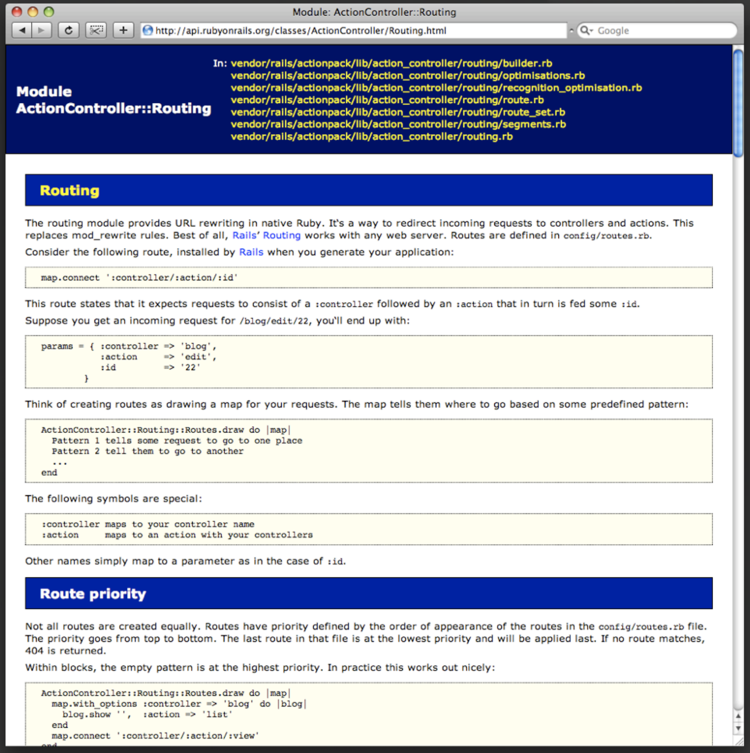
Introduction

Maintainable Software

Routes

- Answers “how do I map URLs to my code?”
- Started as a port of the routing system from Ruby on Rails, still very similar to Rails
- Routes itself has now been ported to PHP 5 as part of the Horde Project (Horde/Routes)

Routes Ecosystem



Module: ActionController::Routing

http://api.rubyonrails.org/classes/ActionController/Routing.html

Module ActionController::Routing

In: vendor/rails/actionpack/lib/action_controller/routing/builder.rb
vendor/rails/actionpack/lib/action_controller/routing/optimisations.rb
vendor/rails/actionpack/lib/action_controller/routing/recognition_optimisation.rb
vendor/rails/actionpack/lib/action_controller/routing/route.rb
vendor/rails/actionpack/lib/action_controller/routing/route_set.rb
vendor/rails/actionpack/lib/action_controller/routing/segments.rb
vendor/rails/actionpack/lib/action_controller/routing.rb

Routing

The routing module provides URL rewriting in native Ruby. It's a way to redirect incoming requests to controllers and actions. This replaces mod_rewrite rules. Best of all, Rails' Routing works with any web server. Routes are defined in `config/routes.rb`.

Consider the following route, installed by Rails when you generate your application:

```
map.connect ':controller/:action/:id'
```

This route states that it expects requests to consist of a `:controller` followed by an `:action` that in turn is fed some `:id`.

Suppose you get an incoming request for `/blog/edit/22`, you'll end up with:

```
params = { :controller => 'blog',
           :action     => 'edit',
           :id         => '22'
        }
```

Think of creating routes as drawing a map for your requests. The map tells them where to go based on some predefined pattern:

```
ActionController::Routing::Routes.draw do |map|
  Pattern 1 tells some request to go to one place
  Pattern 2 tell them to go to another
  ...
end
```

The following symbols are special:

```
:controller maps to your controller name
:action     maps to an action with your controllers
```

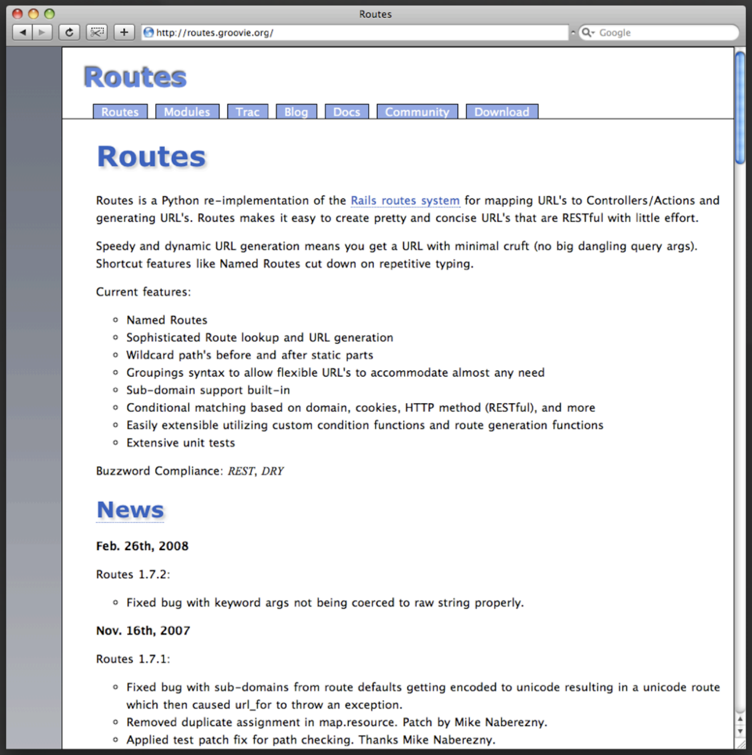
Other names simply map to a parameter as in the case of `:id`.

Route priority

Not all routes are created equally. Routes have priority defined by the order of appearance of the routes in the `config/routes.rb` file. The priority goes from top to bottom. The last route in that file is at the lowest priority and will be applied last. If no route matches, 404 is returned.

Within blocks, the empty pattern is at the highest priority. In practice this works out nicely:

```
ActionController::Routing::Routes.draw do |map|
  map.with_options :controller => 'blog' do |blog|
    blog.show '',  :action => 'list'
  end
  map.connect ':controller/:action/:view'
```



Routes

http://routes.groovie.org/

Routes

Routes

Routes is a Python re-implementation of the Rails routes system for mapping URL's to Controllers/Actions and generating URL's. Routes makes it easy to create pretty and concise URL's that are RESTful with little effort.

Speedy and dynamic URL generation means you get a URL with minimal cruft (no big dangling query args). Shortcut features like Named Routes cut down on repetitive typing.

Current features:

- Named Routes
- Sophisticated Route lookup and URL generation
- Wildcard path's before and after static parts
- Groupings syntax to allow flexible URL's to accommodate almost any need
- Sub-domain support built-in
- Conditional matching based on domain, cookies, HTTP method (RESTful), and more
- Easily extensible utilizing custom condition functions and route generation functions
- Extensive unit tests

Buzzword Compliance: *REST*, *DRY*

News

Feb. 26th, 2008

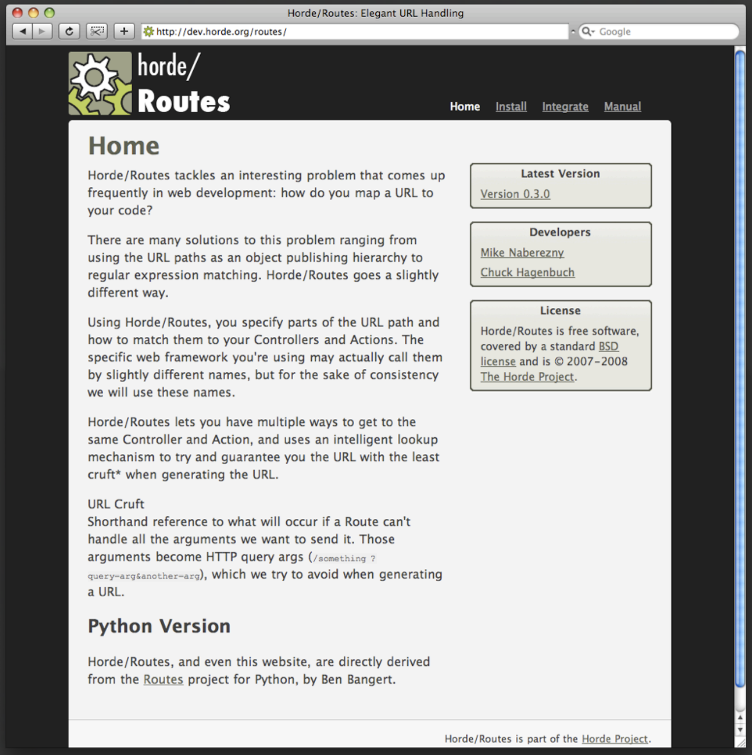
Routes 1.7.2:

- Fixed bug with keyword args not being coerced to raw string properly.

Nov. 16th, 2007

Routes 1.7.1:

- Fixed bug with sub-domains from route defaults getting encoded to unicode resulting in a unicode route which then caused url_for to throw an exception.
- Removed duplicate assignment in map.resource. Patch by Mike Naberezny.
- Applied test patch fix for path checking. Thanks Mike Naberezny.



Horde/Routes: Elegant URL Handling

http://dev.horde.org/routes/

horde/ Routes

Home

Horde/Routes tackles an interesting problem that comes up frequently in web development: how do you map a URL to your code?

There are many solutions to this problem ranging from using the URL paths as an object publishing hierarchy to regular expression matching. Horde/Routes goes a slightly different way.

Using Horde/Routes, you specify parts of the URL path and how to match them to your Controllers and Actions. The specific web framework you're using may actually call them by slightly different names, but for the sake of consistency we will use these names.

Horde/Routes lets you have multiple ways to get to the same Controller and Action, and uses an intelligent lookup mechanism to try and guarantee you the URL with the least cruft* when generating the URL.

URL Cruft
Shorthand reference to what will occur if a Route can't handle all the arguments we want to send it. Those arguments become HTTP query args (`/something ?query=arg&another=arg`), which we try to avoid when generating a URL.

Python Version

Horde/Routes, and even this website, are directly derived from the Routes project for Python, by Ben Bangert.

Latest Version
Version 0.3.0

Developers
Mike Naberezny
Chuck Hagenbuch

License
Horde/Routes is free software, covered by a standard BSD license and is © 2007-2008 The Horde Project.

Horde/Routes is part of the Horde Project.

Ruby ⟶ Python ⟷ PHP

Maintainable Software

http://routes.groovie.org

Routes

- Provides solutions for both recognizing URLs and generating URLs
- Standalone component that is easy to integrate and web framework agnostic
- Used by Pylons and others
- Developed by Ben Bangert & contributors

Maintainable Software

Installation

- Available as source distribution or egg
  http://pypi.python.org/pypi/Routes

- `easy_install routes`

Terminology

- A web application exposed by Routes is organized at the top-level into *controllers*
- Each *controller* is typically responsible for a single application resource (usually a noun)
  - `PostsController`
  - `CommentsController`
  - `AuthorsController`

Maintainable Software

Terminology

- Each *controller* responds to *actions* (usually a verb) that act on a resource

- **`PostsController`**
  - **`index, show, update, delete`***

Maintainable Software

**Rails calls this "destroy"*

Terminology

- The *action* of a *controller* may receive other pieces of the URL as *parameters*.
- `/:controller/:action/:id`
- `/posts/show/5`

Setting up the Mapper

Maintainable Software

Mapper

- Mapper is the core of the Routes system. You `connect()` routes to the mapper.
- You can then `match()` a URL against the set of routes you have connected.

Mapper

- As far as Routes is concerned, the list of controller names is just a list of names.
- Routes just performs matching. It's up to you or your framework to dispatch what it matches into your application structure.

Maintainable Software

Mapper

```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect(':controller/:action/:id')

>>> map.match('/blogs/show/1')
```

No match!

Maintainable Software

Mapper

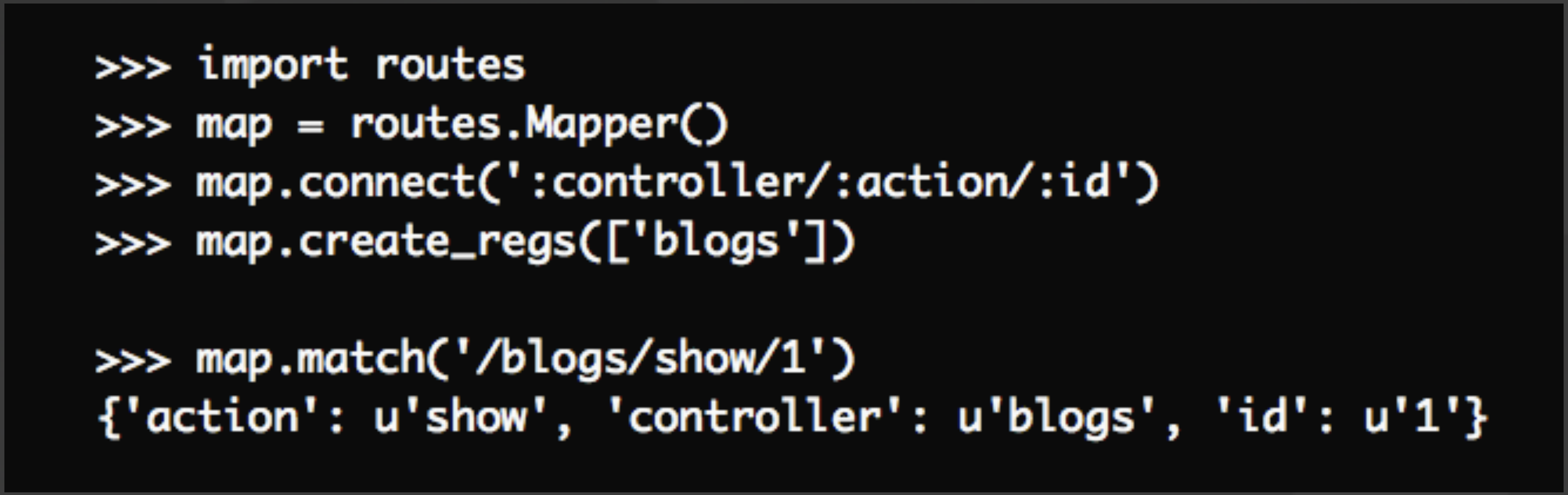
- Internally, Routes uses regular expressions to match connected routes against URLs.

- These regular expressions must be generated before routes can be matched.

Create RegExps

- You need to `create_regs()` on the Mapper before its routes can be matched.
- Controllers are special.
- Routes needs to know the name of every controller in your application to `create_regs()`.

Create RegExps



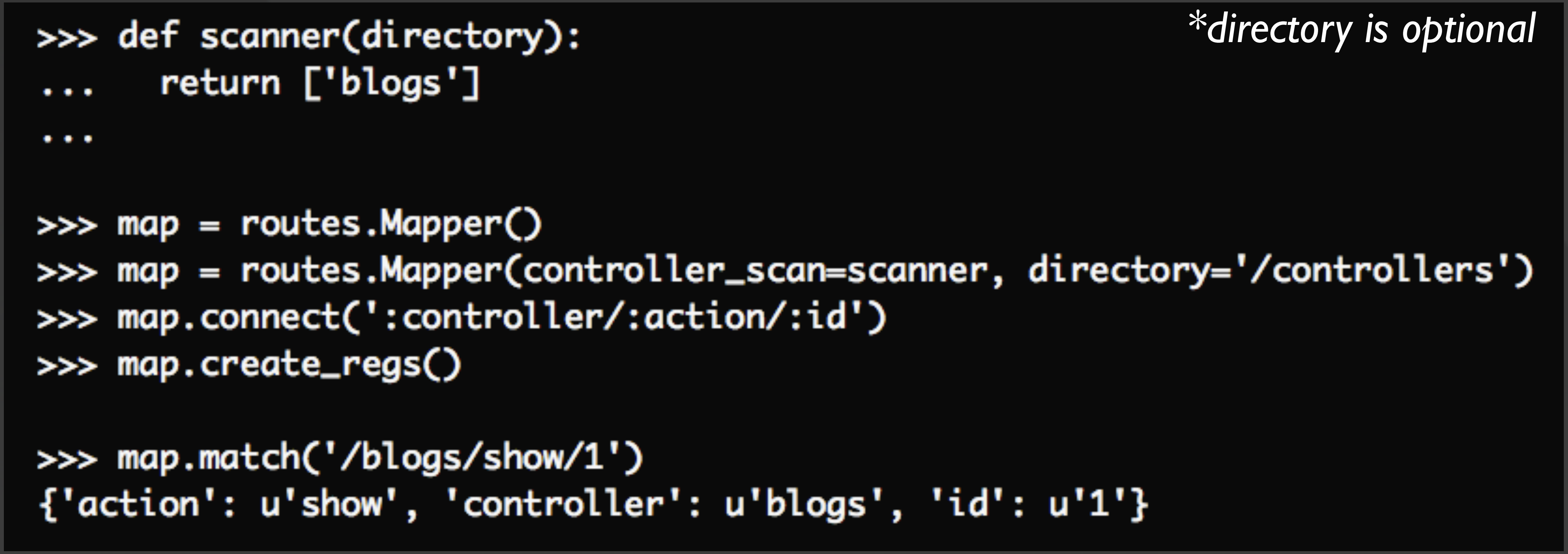
```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect(':controller/:action/:id')
>>> map.create_regs(['blogs'])

>>> map.match('/blogs/show/1')
{'action': u'show', 'controller': u'blogs', 'id': u'1'}
```

Option 1

Pass a list of all controller names to `create_regs()`

Maintainable Software

Create RegExps



**directory is optional*

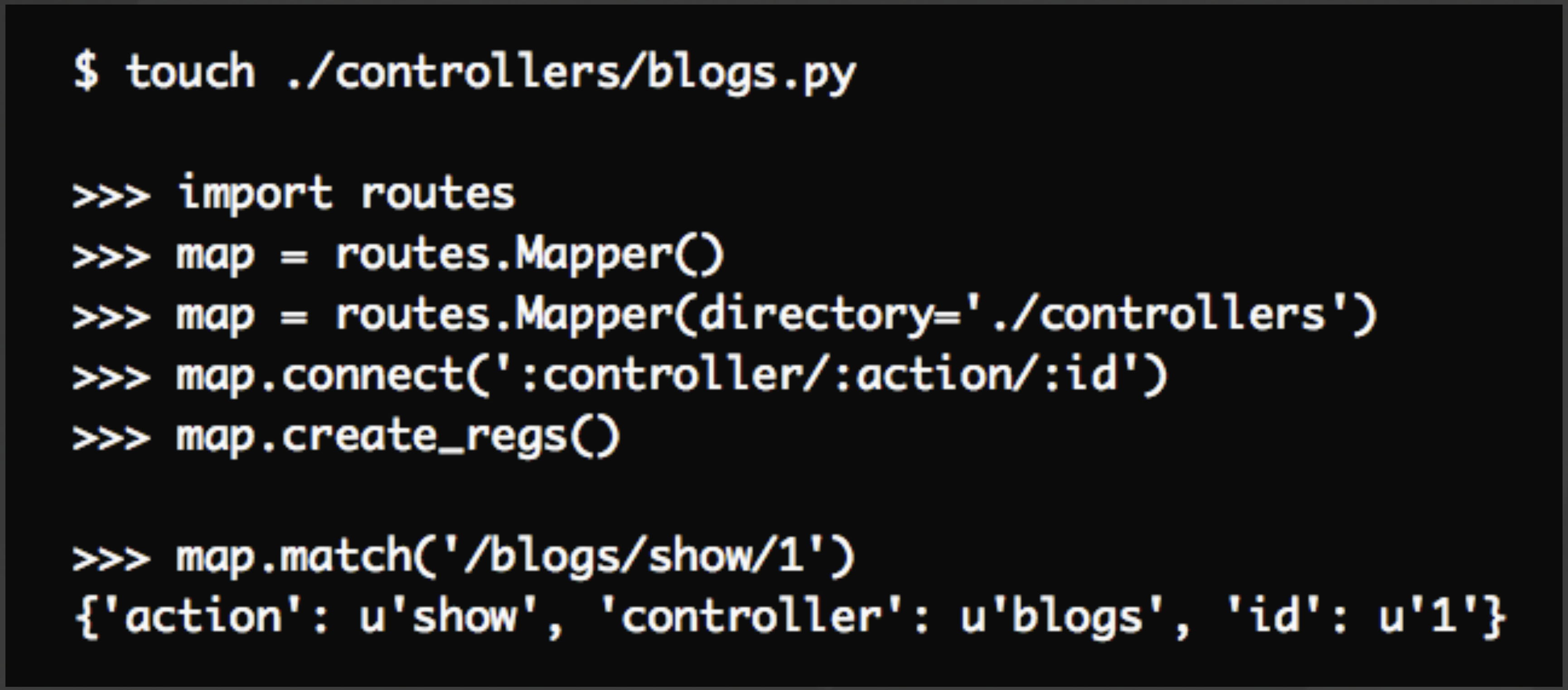
```
>>> def scanner(directory):
...     return ['blogs']
...

>>> map = routes.Mapper()
>>> map = routes.Mapper(controller_scan=scanner, directory='/controllers')
>>> map.connect(':controller/:action/:id')
>>> map.create_regs()

>>> map.match('/blogs/show/1')
{'action': u'show', 'controller': u'blogs', 'id': u'1'}
```

Option 2

`controller_scan` callback builds controller list

Maintainable Software

Create RegExps



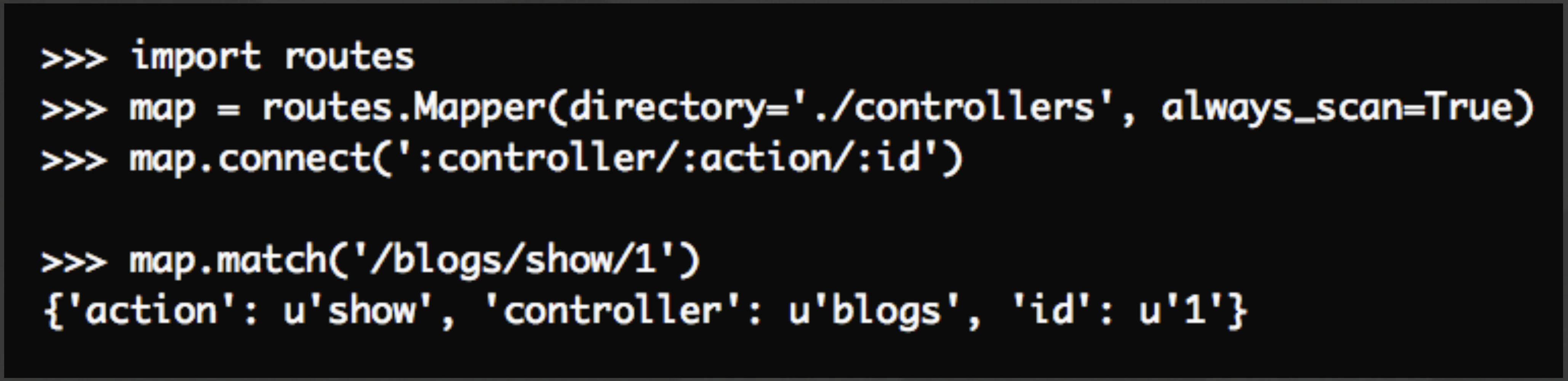
```
$ touch ./controllers/blogs.py

>>> import routes
>>> map = routes.Mapper()
>>> map = routes.Mapper(directory='./controllers')
>>> map.connect(':controller/:action/:id')
>>> map.create_regs()

>>> map.match('/blogs/show/1')
{'action': u'show', 'controller': u'blogs', 'id': u'1'}
```

Option 3

Default `routes.util.controller_scan` function

Maintainable Software

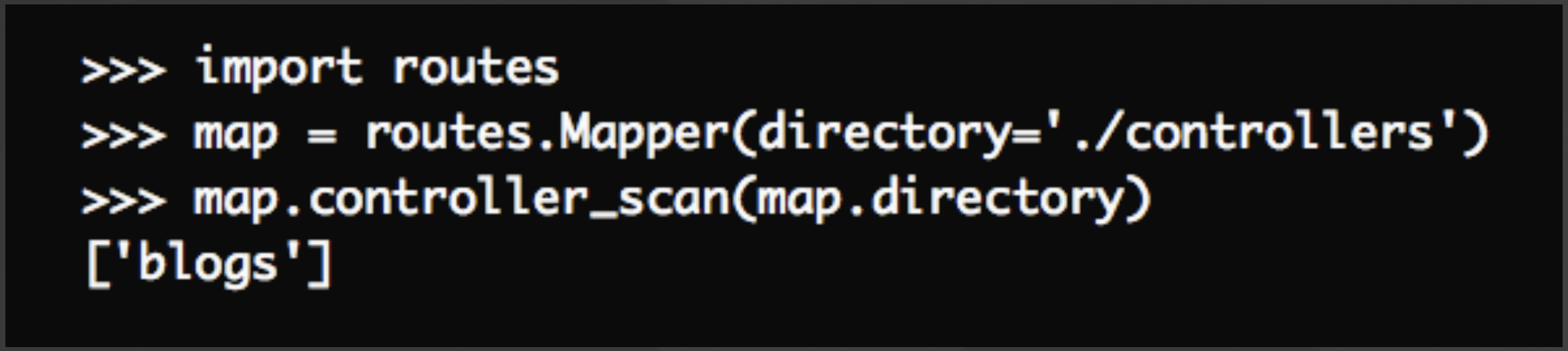
Tips



```
>>> import routes
>>> map = routes.Mapper(directory='./controllers', always_scan=True)
>>> map.connect(':controller/:action/:id')

>>> map.match('/blogs/show/1')
{'action': u'show', 'controller': u'blogs', 'id': u'1'}
```

- `always_scan` will cause `create_regs()` called before any `match()`.
- This is useful mostly during development.

Tips



```
>>> import routes
>>> map = routes.Mapper(directory='./controllers')
>>> map.controller_scan(map.directory)
['blogs']
```

- Call `controller_scan` for sanity if routes don't match when you think they should.

Maintainable Software

Review

- Create Mapper Instance
- Connect Routes to the Mapper
- Generate Regular Expressions
- Match or Generate

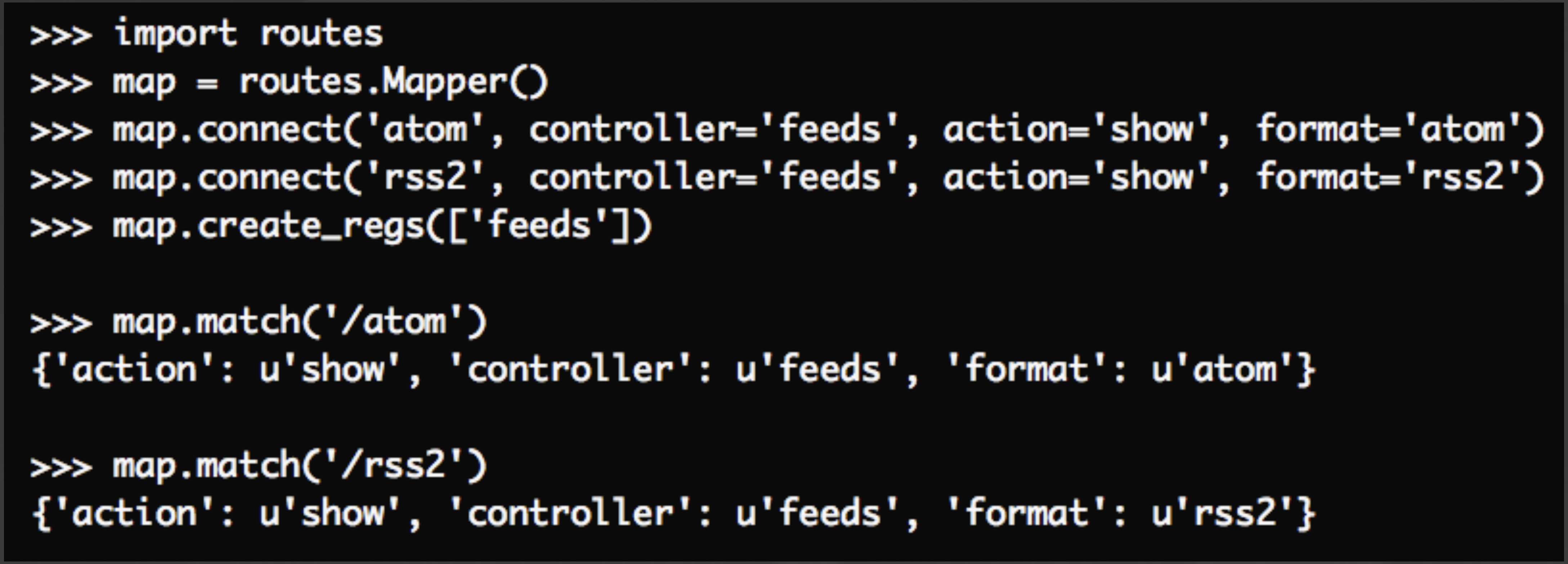
Route Recognition

Maintainable Software

Path Parts

Maintainable Software

Path Parts: Static

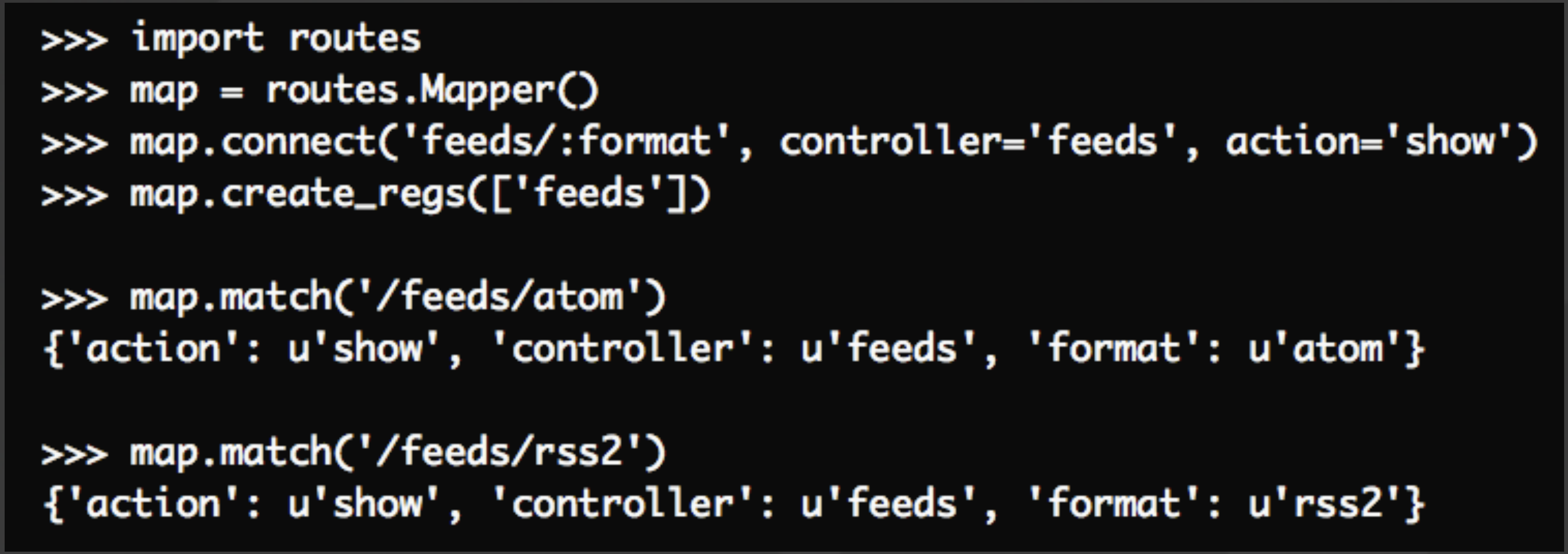


```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect('atom', controller='feeds', action='show', format='atom')
>>> map.connect('rss2', controller='feeds', action='show', format='rss2')
>>> map.create_regs(['feeds'])

>>> map.match('/atom')
{'action': u'show', 'controller': u'feeds', 'format': u'atom'}

>>> map.match('/rss2')
{'action': u'show', 'controller': u'feeds', 'format': u'rss2'}
```

- Both routes have static paths: `atom` and `rss2`

Maintainable Software

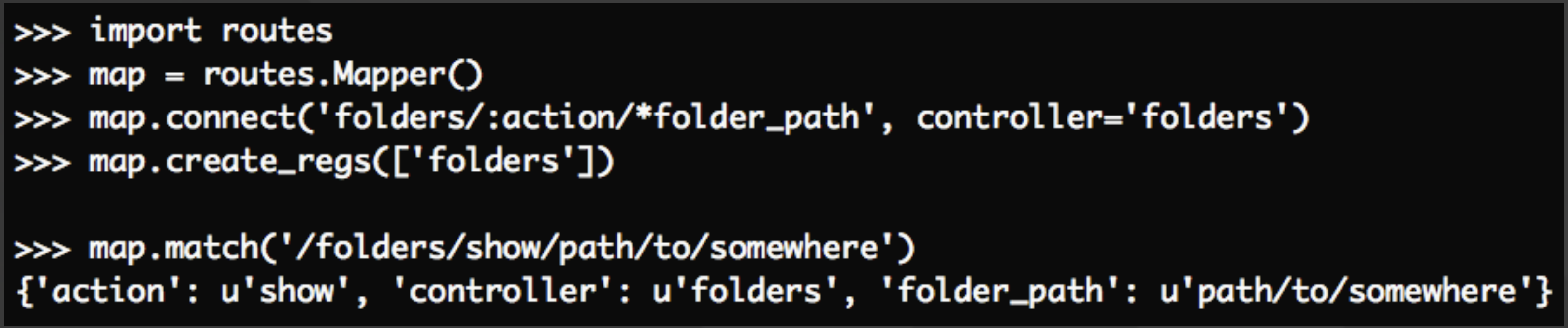
Path Parts: Dynamic



```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect('feeds/:format', controller='feeds', action='show')
>>> map.create_regs(['feeds'])

>>> map.match('/feeds/atom')
{'action': u'show', 'controller': u'feeds', 'format': u'atom'}

>>> map.match('/feeds/rss2')
{'action': u'show', 'controller': u'feeds', 'format': u'rss2'}
```

- Static part: `feeds`
- Dynamic part: `:format`

Path Parts: Wildcard



```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect('folders/:action/*folder_path', controller='folders')
>>> map.create_regs(['folders'])

>>> map.match('/folders/show/path/to/somewhere')
{'action': u'show', 'controller': u'folders', 'folder_path': u'path/to/somewhere'}
```

- Static part: `folders`
- Dynamic part: `:action`
- Wildcard part: `*folder_path`

Maintainable
Software

Defaults

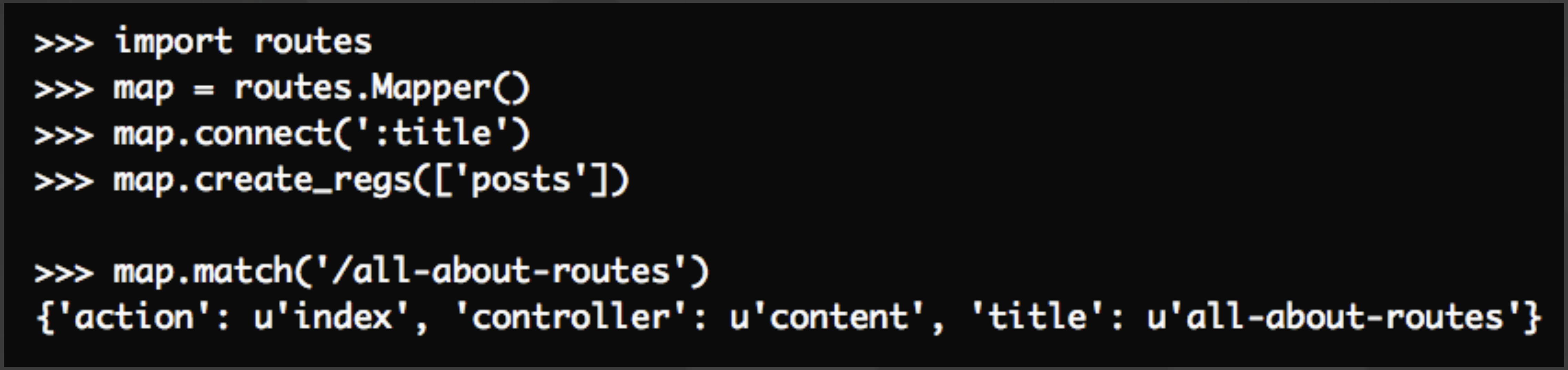
Maintainable Software

Defaults

```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect(':title', controller='posts', action='show')
>>> map.create_regs(['posts'])

>>> map.match('/all-about-routes')
{'action': u'show', 'controller': u'posts', 'title': u'all-about-routes'}
```

- Routes are free-form. Controller and action do not need to be part of the URL itself.

Implicit Defaults

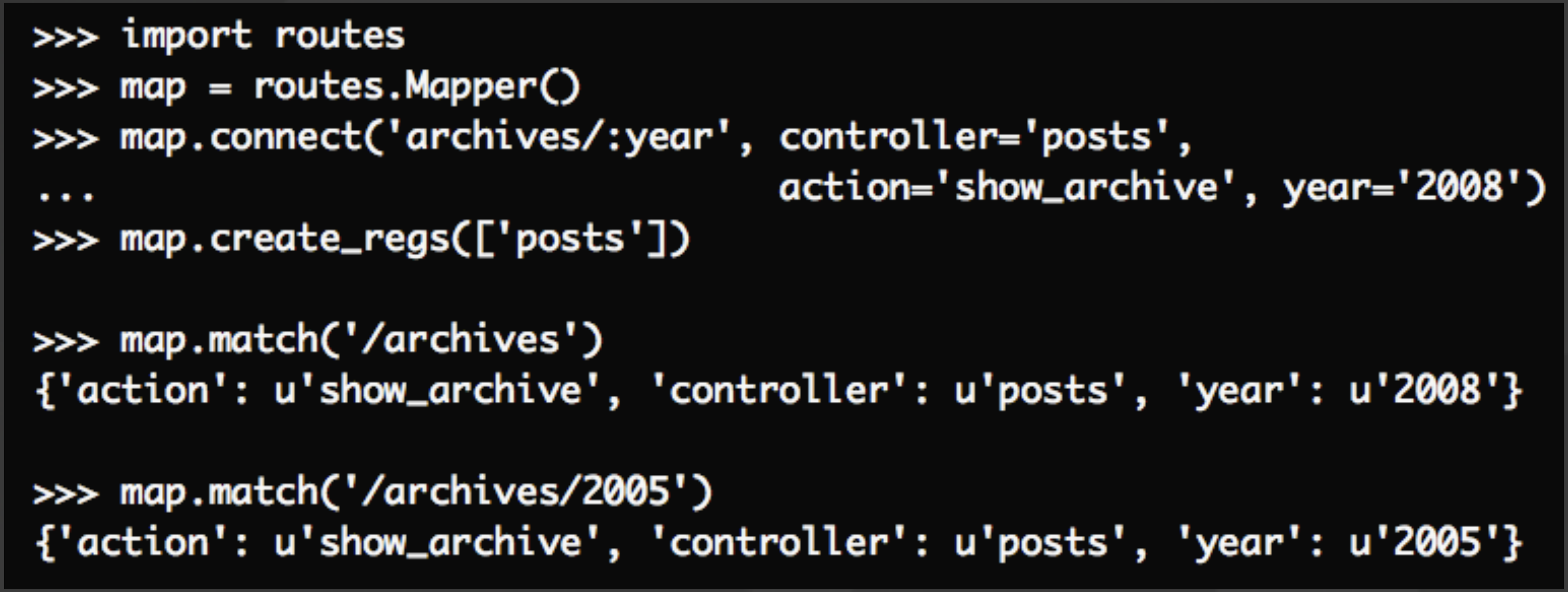


```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect(':title')
>>> map.create_regs(['posts'])

>>> map.match('/all-about-routes')
{'action': u'index', 'controller': u'content', 'title': u'all-about-routes'}
```

- Gotcha. Notice magic `content` and `index`
- `Mapper(explicit=False)` is standard, giving all routes implicit defaults

Maintainable Software

Defaults



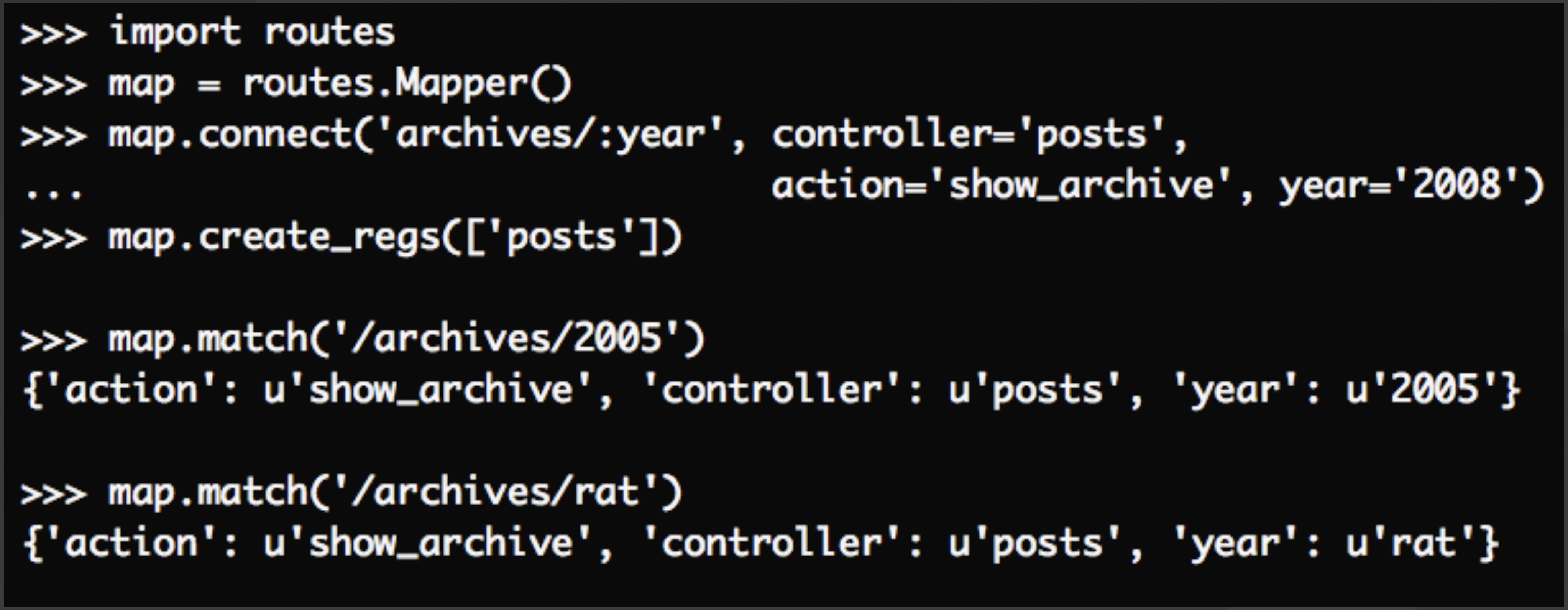
```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect('archives/:year', controller='posts',
...                               action='show_archive', year='2008')
>>> map.create_regs(['posts'])

>>> map.match('/archives')
{'action': u'show_archive', 'controller': u'posts', 'year': u'2008'}

>>> map.match('/archives/2005')
{'action': u'show_archive', 'controller': u'posts', 'year': u'2005'}
```

- Defaults are used to implement optional parts of the URL (`year`)

Requirements & Conditions

Maintainable Software

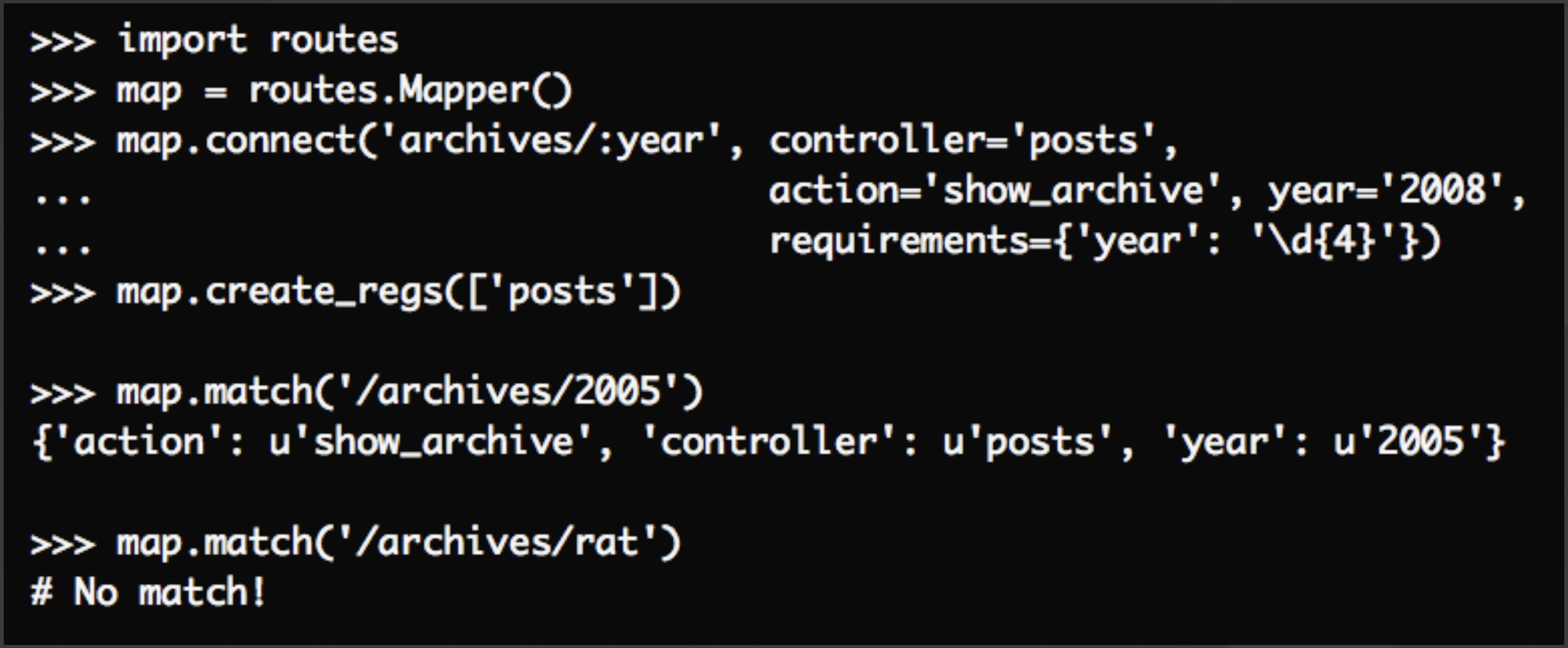
Requirements



```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect('archives/:year', controller='posts',
...                               action='show_archive', year='2008')
>>> map.create_regs(['posts'])

>>> map.match('/archives/2005')
{'action': u'show_archive', 'controller': u'posts', 'year': u'2005'}

>>> map.match('/archives/rat')
{'action': u'show_archive', 'controller': u'posts', 'year': u'rat'}
```

- “Year of the `rat`” is probably not something that we want to support.

Requirements



```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect('archives/:year', controller='posts',
...                              action='show_archive', year='2008',
...                              requirements={'year': '\d{4}'})
>>> map.create_regs(['posts'])

>>> map.match('/archives/2005')
{'action': u'show_archive', 'controller': u'posts', 'year': u'2005'}

>>> map.match('/archives/rat')
# No match!
```

- Requirements help cut down on validation in application code. Be specific.

Maintainable Software

Conditions

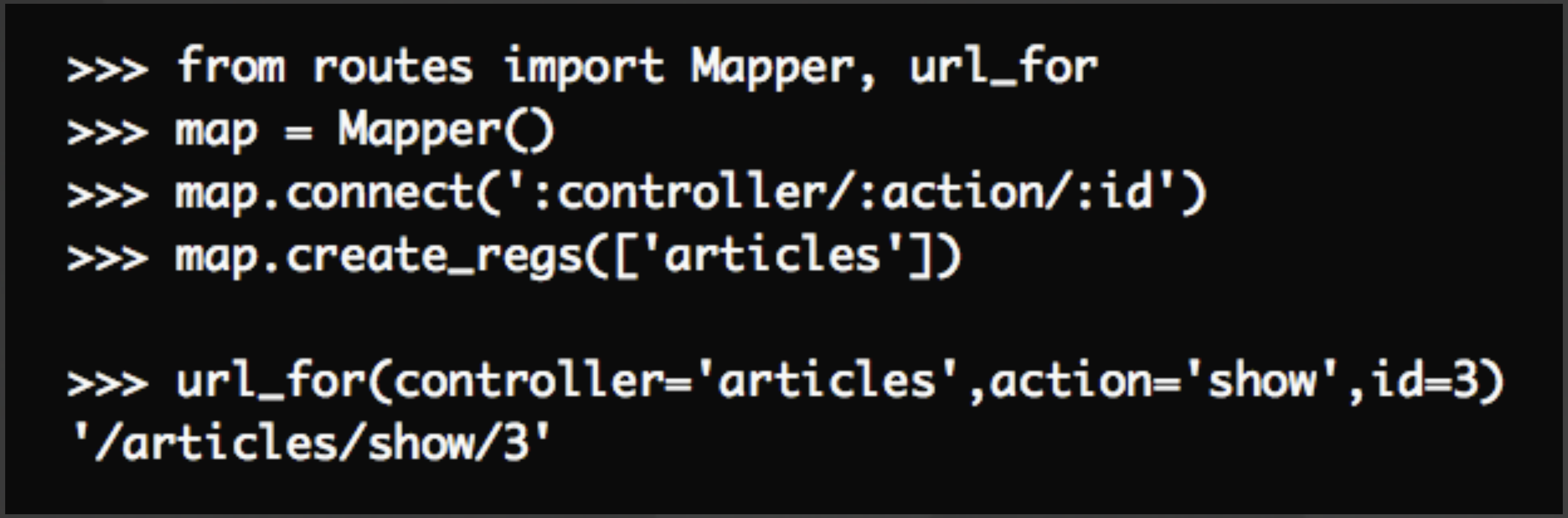
```
>>> import routes
>>> map = routes.Mapper()
>>> map.connect('posts/create', controller='posts', action='create',
...                              conditions={'method':'POST'})
>>> map.create_regs(['posts'])

>>> map.environ = {'REQUEST_METHOD': 'POST'}
>>> map.match('/posts/create')
{'action': u'create', 'controller': u'posts'}

>>> map.environ = {'REQUEST_METHOD': 'GET'}
>>> map.match('/posts/create')
# No match!
```

- Routes can enforce conditions on the request environment in addition to requirements on the URL itself.

URL Generation

Maintainable Software

URL Generation

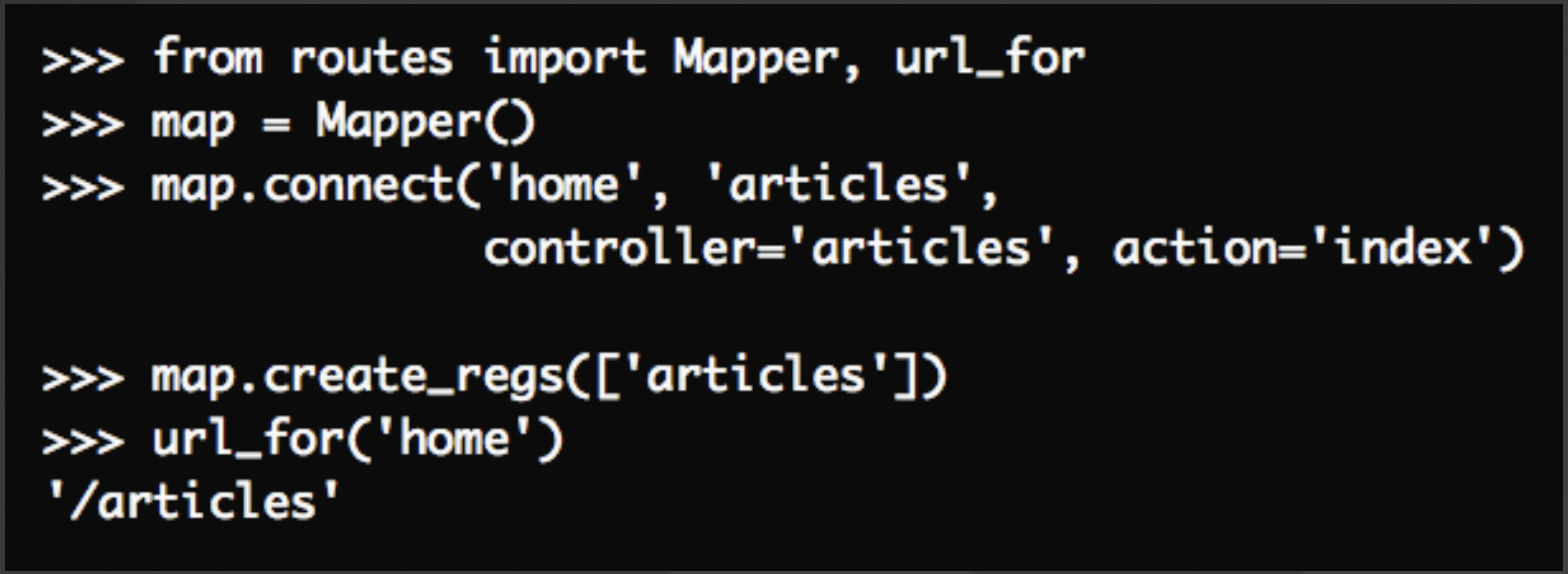


```
>>> from routes import Mapper, url_for
>>> map = Mapper()
>>> map.connect(':controller/:action/:id')
>>> map.create_regs(['articles'])

>>> url_for(controller='articles',action='show',id=3)
'/articles/show/3'
```

- Generating URLs allows the structures to change without changing the application code

Maintainable Software

Named Routes



```
>>> from routes import Mapper, url_for
>>> map = Mapper()
>>> map.connect('home', 'articles',
                controller='articles', action='index')

>>> map.create_regs(['articles'])
>>> url_for('home')
'/articles'
```

- We can give a name to each route as we connect them. This should be considered a best practice and makes generation easier.

Maintainable Software

More

- Static Named Routes
- Filter Functions
- Grouping Path Parts
- More conditions: subdomain, function
- Minimization

- Encoding
- RESTful Routes
- Mapper . routematch()
- Alternate syntax {controller}/{action}
- Redirects

Maintainable Software

Resources

- Narrative and API documentation
  http://routes.groovie.org
- Issue tracking and Subversion mirror
  http://routes.groovie.org/trac
- Developed with Mercurial at
  https://www.knowledgetap.com/hg/routes/

Maintainable Software

Q & A

Maintainable Software