

def ruby()
 20.times do
 liqueur { :type => :cocoa, :name =>
:Malibu }
 liqueur { :type => :lychee }
 end
 if liqueur.exist? { :type => :brandy
} then l = :brandy else l = :cognac
 40.times do liqueur { :type => l }
end
 lemon
 ice
end

Cocktails For Programmers

HACKERMONTHLY

Issue 42 November 2013



RUBY

Curator

Lim Cheng Soon

Contributors

Ilya Zykin

Vulpyne

Trevor McKendrick

Jack McDade

Job Vranish

Pete Keen

Kerrick Long

Matt Wright

Miles Bader

Stephane Epardaud

Chong Kim

Proofreaders

Emily Griffin

Sigmarie Soto

Ebook Conversion

Ashish Kumar Jha

Printer

MagCloud

HACKER MONTHLY is the print magazine version of Hacker News — *news.ycombinator.com*, a social news website wildly popular among programmers and startup founders. The submission guidelines state that content can be “anything that gratifies one’s intellectual curiosity.” Every month, we select from the top voted articles on Hacker News and print them in magazine format. For more, visit *hackermonthly.com*

Advertising

ads@hackermonthly.com

Contact

contact@hackermonthly.com

Published by

Netizens Media
46, Taylor Road,
11600 Penang,
Malaysia.



Hacker Monthly is published by Netizens Media and not affiliated with Y Combinator in any way.

Contents

FEATURES

04 Cocktails for Programmers

By ILYA ZYKIN

12 How to Choose a Profitable Niche

By TREVOR MCKENDRICK



SPECIAL

16 Why I Play Video Games

By JACK MCDADE



PROGRAMMING

20 Unix Commands I Wish I'd Discovered Years Earlier

By JOB VRANISH

24 DNS: The Good Parts

By PETE KEEN

31 FTP is so 90s. Let's deploy via Git instead!

By KERRICK LONG

34 How I Structure My Flask Applications

By MATT WRIGHT

42 What Makes Lua Tick

By MILES BADER

44 What Every Web Developer Must Know About URL Encoding

By STÉPHANE ÉPARDAUD

54 Using Katas to Improve Your Coding

By CHONG KIM

For links to Hacker News discussions, visit hackermoonly.com/issue-42

Cocktails for Programmers

By ILYA ZYKIN

Code contributed by VULPYNE

A CULINARY PROJECT FOR a professional holiday, “Programmers Day,” celebrated on the 256th day of the year.

Ruby

The drink will be sweet, fragrant, and fresh. This ruby-colored cocktail perfectly matches the Ruby logo.

Ingredients

- 20 mL Malibu (coconut liqueur)
- 20 mL Lychee Liqueur (a fruit)
- 40 mL Cognac or Brandy
- 150 mL Cherry juice
- Lemon
- Ice





Directions

1. Add basic ingredients into a tall glass filled with ice.
2. Add lemon juice to taste.
3. Thoroughly mix until the glass is misted.
4. Garnish with a slice of orange and a maraschino cherry.

Code

```
def ruby()
  20.times do
    liqueur { :type => :cocoa,
:name => :Malibu }
    liqueur { :type => :lychee }
  end
  if liqueur.exist? { :type =>
:brandy } then l = :brandy else l
= :cognac
  40.times do liqueur { :type =>
l } end
  lemon
  ice
end
```

Python

This cocktail looks like a green python. It must be served fast — its froth disappears quickly. This drink has a fresh pineapple taste.

Ingredients

- 50 mL White Rum
- 30 mL Mint Liqueur
- 30 mL Pineapple Juice
- Juice of half a Lemon
- Sprite or 7 Up

Directions

- 1.Shake together with ice and strain.
- 2.Pour into a glass and add Sprite or 7 Up.
- 3.Garnish with a slice of lime.



Code

```
def python():
    d = Drink()
    d.addIngredient(50, type = 'rum', name = 'white')
    d.addIngredient(30, type = 'liqueur', name = 'mint')
    d.addIngredient(30, type = 'juice', name = 'pineapple')
    d.addIngredient(Lemon.unitsPerFruit() / 2.0, type = 'juice', name =
'lemon')
    while not d.glassOverflowed():
        d.addIngredient(1, type = 'soda', name = 'sprite')
    return d
```



Severe Perl

Associations: severe, dry, desert, camel.

Ingredients

- 30 mL Gin
- 20 mL Dry Vermouth Rosso
- 20 mL Lemon Juice
- 10 mL Syrup

Directions

1. Shake together with ice and strain.
2. Garnish with physalis.

Code

Vulpyne: Well, I don't actually know Perl and I don't feel like learning it for this. So here is my best attempt:

```
$%!#$$$%^@##$!@##$!@#!%$$$%^#@##$@##$
@##$"gin"
$$$%@##$%^$%@##$/=|$"dry vermouth
rosso"
""<>.(79348*&("lemon juice"({}
)}}{{}}{$$$$$"syrup"*#(*#$83 ||
die();
```

JMP (aka Assembler)

Ingredients

- 20 mL Jagermeister (herbal liqueur)
- 20 mL Midori (melon liqueur) + Lime
- 20 mL Peach Syrup

Directions

1. Pour all the ingredients with a bar spoon in a high shot glass layer-by-layer.

Code

```
.global _start  
.text
```

```
_start:  
mov $0xfeed, %rax  
mov $0x14, %rcx  
mov jager, %rdi  
cld  
rep movmd ; md = make drunk, natu-  
rally.  
mov $0x14, %rcx  
mov midori, %rdi  
rep movmd  
mov peach, %rdi  
mov $0x14, %rcx  
rep movmd  
jmp $0xfeedface  
  
:jager  
.ascii "Jagermeister"  
:midori  
.ascii "Midori"  
peach:  
.ascii "peach syrup"
```



Profit!

Profit! should be sweet and airy. That's how we saw this cocktail.



Ingredients

- 20 mL Creamy Liqueur
- 20 mL Crème de Cassis
- 20 mL Triple Sec
- Whipped Cream
- Cocoa Powder

Directions

- 1.Shake together with ice and strain.
- 2.Garnish with whipped cream and dust with cocoa powder (use a sifter for better results).
- 3.Put a cherry on top.

Epic Fail

By design, the lemon and Coke conceal the taste of alcohol. But if you go too far, it will be a real epic fail. Be careful if you want to try something like this!



Ingredients

- 50 mL Vodka
- 100 mL Coke
- Juice of Half a Lemon
- Ice

Directions:

- 1.Fill a glass of ice with all ingredients.
- 2.Thoroughly mix till the glass is misted.
- 3.Garnish with a lemon slice.

Memory Leak

Ingredients

- 50 mL Tequila
- 50 mL White Rum
- 50 mL Triple Sec
- 50 mL Kahlua
- Lime
- Coke

Directions

1. Fill a glass with ice and small pieces of lime.
2. Add the rest of the ingredients and mix.
3. Garnish with a slice of lime and you've got tasty and stunning drink.

Code

```
struct Drink *make_drink() {  
    struct Drink *drink;  
    struct Ingredient *ingredient;  
  
    drink = malloc(sizeof(struct  
Drink));  
    drink->ingredients = ingredient =  
malloc(sizeof(struct Ingredient) *  
7);  
    *ingredient.amount = 50;  
    *ingredient.name = "tequila";  
    ingredient++;  
    *ingredient.amount = 50;  
    *ingredient.name = "white rum";  
    ingredient++;
```



```

    *ingredient.amount = 50;
    *ingredient.name = "triple sec";
    ingredient++;
    *ingredient.amount = 50;
    *ingredient.name = "kahlua";
    ingredient++;
    addLime(ingredient++);
    *ingredient.amount = sizeof(Glass) - 100;
    *ingredient.name = "coke";
    *ingredient++;
    memset(ingredient, 0, sizeof(struct Ingredient));
    return drink;
}

void free_drink(struct Drink *) {
    free(drink);
} ■

```

Ilya is a Ruby on Rails and front-end developer living in Saint-Petersburg, Russia. In the past, he was a school teacher of informatics. Ilya dreams to create a social-oriented CMS for food-bloggers and publishes his pet projects in his GitHub account [github.com/the-teacher].

Vulpyne started programming on a TRS-80 when he was 9. These days, he mostly codes in Haskell (his poison of choice) and Python. Vulpyne never had any formal education and have been programming professionally for about 14 years. He lives in a little cabin in the mountains of Colorado with his three dogs.

Additional Credits: Artem (making the cocktails), Anna Nechaeva (photo), Sergey Romanov (english translation), Trevor Strieber (english translation).

Reprinted with permission of the original author.
 First appeared in *hn.my/cocktails*

How to Choose a Profitable Niche

By TREVOR MCKENDRICK

WE CAN ESTIMATE how profitable an app is using Gross and Paid Rankings. This helps us decide whether it's a good niche to get into.

A Framework for Evaluating Potential Niches

When I was considering building my Spanish Bible app I wanted to be as sure as possible that people were going to be able to find it and buy it. With that in mind I came up with the idea of the ideal target niche.

The ideal niche:

1. Is profitable
2. Can be found through search
3. Has crappy competitors

Today I'm going to explain how to figure out #1.

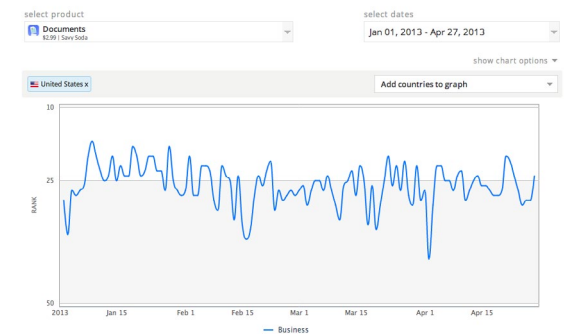
1 Find an app that ranks #25 Paid
I found two apps in the Business category that have a history of ranking around #25 Paid. You can see

their historical Paid rankings below via AppFigures:

PDF Expert



Documents



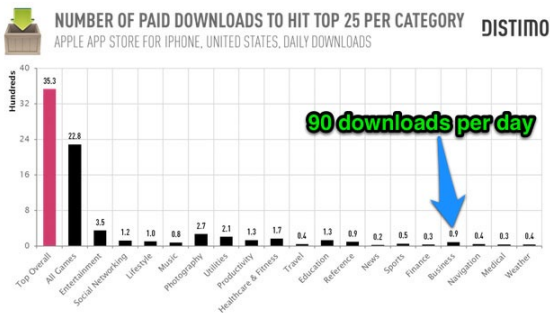
Why #25 Paid? See step 2.

2 Calculate the apps' daily revenue using Distimo

Last year I asked the nice folks at Distimo to analyze how many downloads it takes an app to rank #25 Paid by category.

Gert Jan Spriensma, a Distimo analyst at the time, was nice enough to respond with this post [hn.my/distimo] which got picked up by TechCrunch.

You don't even need to read the whole post, just this one chart:



This means our two example apps are being downloaded roughly 90 times a day. We can estimate their daily revenue by multiplying their prices by 90.

	Documents	PDF Expert
Price	\$3	\$10
Downloads (Distimo)	90	90
Revenue (after Apple)	\$190	\$630

3 Look up Gross Ranking

Easy with AppFigures:

Eyeballing the charts it looks like their average Gross Ranks are #13 and #50, respectively.



4 Plot the Data

This is what we've collected:

	Gross Rank	Estimated Revenue
Documents	50	\$190
PDF Expert	13	\$630

Plotting it we get this:



This gives us an estimate on how much apps in the Business category make.

Will Customers Find You Via Search?

You need to find something that can be found with a frequently searched keyword that doesn't have a lot of competitors.

This is a tough one because Apple doesn't release keyword data. While there do exist tools now that approximate keyword search frequency, I didn't know about them when I picked my niche.

One in particular that I've started using lately is *Straply.com*. I've talked to the cofounder and he calls it the first "Google Keyword Tool" for App Stores. The interface isn't very good yet, but the data he's collecting is remarkable. It'll tell you how often a term is searched and how many competitors also appear in the App Store Search Results (ASSRs).

Do It Yourself

But I'd also recommend doing your own tried-and-true research. I did the following before most any App Store optimization tools went mainstream:

For each niche I brainstormed a bunch of keywords/phrases. I plugged those words into the Google Keyword Tool and clicked the "mobile only" option.

From there I selected the top 30 or so keywords. And I plugged those into the search bar of the App Store.

Then I go through the ASSRs.

For the top 5 or 10 results for each keyword I consider a few metrics:

- Does the app have a lot of reviews?
How recent are the reviews?
- When did the developer last update the app?
- Are there any apps that make good money and only rank high in the ASSRs for a few keywords?

In an ideal world you'd find an app that has lots of reviews and that ranks well in the ASSRs for one keyword phrase, and of course is making money. That means the phrase is likely to be something users are searching for.

If a profitable app ranks well for a few keywords look at the other apps in the ASSRs. Do they appear to be making money, too? Generally, the more money the top results are making, the more likely the keyword is searched by users.

You do have to be careful here though: some apps will rank well for many keywords and it requires much more detective work to figure out which keywords are the ones users are actually searching.

Is The Keyword Competitive?

Simply look at the number of apps in the ASSRs for the keyword phrases research above.

Example below are the number of results for different keywords with the word “calculator”:

- “calculator” = 10,930
- “tip calculator” = 777
- “scientific calculator” = 336
- “graphing calculator” = 81

I consider anything less than 100 to be great. Anything over 500 is probably too much.

Are The Competitors Any Good?

Again, the App Stores are great because they give public reviews. You already know what users do and don’t like about your competitors. If you decide to get into that niche, you know where the improvements need to be made.

Also, subjectively look at competitors: does it appear the developer is putting time/care/love into the product? Maybe she has become apathetic because she has so little competition and users can’t find anything else. That’s exactly how the landscape looked with Spanish Bibles (It’s worth noting that since then the competition has picked up significantly.)

If it looks like the competition isn’t trying very hard but they’re still making money, it’s likely you’ve found a niche worth investing some more time in. ■

Trevor started Salem Software with \$500 as a side project hoping to just pay his rent. You can follow him on Twitter [@trevmckendrick](#) and read his blog at [trevormckendrick.com](#)

Reprinted with permission of the original author.
First appeared in [hn.my/niche](#) (trevormckendrick.com)



Why I Play Video Games

By JACK MCDADE

I'VE BEEN PLAYING video games again. I feel fantastic and my head is clearer than it has been in a while. Here are my thoughts on the topic, and why I think video games are important and not even remotely a waste of time. Like every other guy who grew up in the '80s, I used to play video games all the time. Super Mario, Double Dragon, Zelda, Street Fighter, GoldenEye, you name it. Later came the Halo and Counter Strike LAN parties. In college I played a lot of World of Warcraft (back when the level cap was 60). Hell, I played games with some semblance of regularity up until I quit my job and started working for myself in 2009.

It was that point when every hour I existed had a dollar value attached to it. Why play 2 hours of Assassin's Creed if you could bill two hours of work? If you're self-employed you either have, or have had this mind-set at some point.

Why play 2 hours of Assassin's Creed if you could bill two hours of work?

So I gave up gaming. I scoffed at gamers, looking down on them as lesser beings not dedicated enough to their craft. Foolish peons! You can't get your time back! Perhaps an exaggeration, but not a large one.

So how did I go from one extreme to the other, and back again?

I started building my own product/app called Statamic [statamic.com]. A neat little (okay, not so little anymore) flat-file CMS. This became my passion (and still is in many ways) for many months, as I fanned the flames of a fragile little web app into a robust tool powering thousands of websites across the internet and around the world.

I rode the emotional tidal waves that come with pouring all your energy into a product, and was elated by every sale, devastated by every refund request.

So I worked even harder, staying up until 4:30am regularly to fix the bugs, answer support tickets, build new features, write and rewrite documentation, design and redesign the website. I kept it up for an incredible amount of time.

And then I crashed. This was probably somewhere around February or March of this year. I entered the fourth month of development on v1.5 with the end no nearer in sight, and I couldn't handle the pressure.

Sounds familiar to some of you, right?

So I decided to take a few nights off. I needed to get my head clear somehow if I was ever going to make this thing sustainable.

Well, I couldn't get Statamic out of my head. I started dream-coding, waking up exhausted without having actually accomplished anything. A new low. I was becoming an insomniac.

So another night goes by, and I'm trying to stay away from my computer. I can't get into any books for whatever reason, so I decide to turn on my Xbox 360 and throw in *Borderlands 2*, which had barely been touched since Christmas.

Boom!

Things changed almost overnight. I had a blast playing *BL2*. It's an awesome game, with just the right balance of action, humor, and speed. I played until 2am every night for a few nights and started sleeping better. It kept

my mind from wandering into "work mode" and stressing me out. I was able to chill out and relax for a while, enjoy myself, and sleep like a human again.

So what happened to Statamic? Did the product suffer?

Just the opposite. I found I had renewed energy, and the time I spent was more productive, more creative, and higher caliber.

I was happier supporting customers, more enthusiastic when promoting and marketing it, and we were able to get v1.5 roughly 6 weeks after that. We've made huge improvements to the platform, sales are up, we have more users than ever, and we're getting great press. And I'm still gaming, and loving every minute of it.

I ended up picking up a PS3 on Craigslist with 20 or so games, and have been plowing through them with vigor. My buddy and neighbor Dave is an avid gamer, and made a number of recommendations, and I've been enjoying gaming like never before. I had missed out on so many incredible titles, and now I'm making up for lost time. I really, really dig games with killer storylines and character development, so I immediately played through everything by Naughty Dog [naughtydog.com] once I got my PS3.

I generally play for an hour or two each night after my wife goes to bed and sometimes a few hours on lazy Sunday afternoons.

This year so far I've beaten (in order):

- Borderlands 2
- Diablo III
- L.A. Noire
- The Last of Us (best game of all time)
- Uncharted
- Uncharted 2
- Uncharted 3
- Tomb Raider
- Battlefield 3

And as of today I'm about halfway through Far Cry 3.

I've started a few others, but if the storyline doesn't grab me or the game-play feels too repetitive I tend to jump into the next game, planning to come back (but I don't always):

- Assassins Creed III
- God of War III
- Bioshock
- Deadspace
- Max Payne 3
- Red Dead Redemption

So there you have it. Do what you will.

Gaming clears my head, makes me more productive when I am working, and, frankly, just makes me happy. Do fun things. You rarely regret it. ■

Jack McDade is a self-employed designer and developer from Upstate New York. He founded Statamic, a developer and client friendly, flat-file CMS, and is a husband and father of two young boys. He's been known to play video games and fish in his spare time, though rarely at the same time.

Reprinted with permission of the original author.
First appeared in *hn.my/videogames* (jackmcdade.com)

Without affiliate.io...



Just you - 7 sales/week

With affiliate.io...



Affiliate #042
- Lisa, Marketing expert

Affiliate #094
- Diana, owns 7 blogs

Affiliate #011
- Tim, power user & ambassador

Affiliate #027
- Tom, industry expert

The Easiest & Quickest Affiliate System

Recruit, track, and promote your business

AFFILIATE.IO

The fast and easy way to accept affiliates into your online business

Visit affiliate.io/hacker for discount

Unix Commands I Wish I'd Discovered Years Earlier

By JOB VRANISH

I'VE BEEN USING *nix systems for quite a while. But there are a few commands that I somehow overlooked and I wish I'd discovered years earlier.

1 man ascii This prints out the ascii tables in octal, hexadecimal and decimal. I can't believe I didn't know about this one until a month ago. I'd always resorted to googling for the tables. This is much more convenient.

NAME

ascii -- octal, hexadecimal and decimal ASCII character sets

DESCRIPTION

The octal set:

000 nul	001 soh	002 stx
003 etx	004 eot	005 enq
006 ack	007 bel	010 bs

011 ht	012 nl	013 vt
014 np	015 cr	016 so
017 si	020 dle	021 dc1
022 dc2	023 dc3	024 dc4
025 nak	026 syn	027 etb
030 can	031 em	032 sub
033 esc	034 fs	035 gs
036 rs	037 us	

2 cal Pulling up a calendar on most systems is almost always a multi-step process by default. Or you can just use the `cal` command.

```
> cal
```

August 2013

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

3 xxd

```
> xxd somefile.bin
```

```
0000000: 83ff 0010 8d01 0408 d301
0408 a540 0408 .....@..
0000010: d701 0408 d901 0408 db01
0408 0000 0000 .....
0000020: 0000 0000 0000 0000 0000
0000 1199 0508 .....
0000030: df01 0408 0000 0000 e199
0508 1d9a 0508 .....
0000040: e501 0408 2912 0508 e901
0408 eb01 0408 ....).....
0000050: ed01 0408 ef01 0408 39e0
0408 55e0 0408 .....9...U...
0000060: 71e0 0408 8de0 0408 a9e0
0408 39f7 0408 q.....9...
0000070: 6df7 0408 a5f7 0408 ddf7
0408 15f8 0408 m.....
```

This is another command I can't believe I didn't know about until recently. `xxd` can generate a hex dump of a given file, and also convert an edited hex dump back into its original binary form. It can also output the hex dump as a C array, which is also super handy:

```
> xxd -i data.bin
```

```
unsigned char data_bin[] = {
    0x6d, 0x61, 0x64, 0x65, 0x20,
    0x79, 0x6f,      0x75, 0x20,
    0x6c, 0x6f, 0x6f, 0x6b, 0x0a
};
unsigned int data_bin_len = 14;
```

I've also used it to compare binary files by generating a hex dump of two files and then diff'ing them.

4 ssh

`ssh` was one of the first non-trivial UNIX utilities that I got familiar with, but it was a while before I realized that it can be used for a lot more than just logging into remote machines.

`ssh` and its accompanying tools can be used for:

- Copying files between computers (using `scp`).
- X-forwarding — connect to a remote machine and have any gui applications started, displayed as if they were started locally, even if the remote machine doesn't have an X server.
- Port forwarding — forward a connection with a local port to a port on a remote machine OR forward connections with a port on a remote machine to a local port.
- SOCKS proxy — forward any connections of an application that supports SOCKS proxies through the remote host. Useful for more secure browsing over public Wi-Fi and for getting around overly restrictive firewalls.

- Typing a password on your local machine once, then using a secure identity to login to several remote machines without having to retype your password by using an ssh key agent. This is awesome.

5

mdfind

This one is specific to mac, as there are other *nix equivalents. It has similar functionality to `find` but uses the Spotlight index. It allows you to search your entire filesystem in seconds. You can also use it to give you live updates when new files that match your query appear. I use it most often when I'm trying to find the obscure location where an application stores some critical file.

```
> mdfind -name homebrew
```

```
/usr/local/Library/Homebrew  
/Users/job/Library/Logs/Homebrew
```



Job Vranish is a software developer at Atomic Object. He tries to bring modern tools and practices to embedded software development in an effort to make it not terrible.

Reprinted with permission of the original author.
First appeared in *hn.my/unixcommands* (atomicobject.com)



Now you can hack on DuckDuckGo

DuckDuckHack

Create instant answer plugins for DuckDuckGo

duckduckhack.com

DNS: The Good Parts

By PETE KEEN

FREQUENTLY I COME across confusion with domain names. Why doesn't my website work? Why is this stupid thing broken, everything I try fails, I just want it to work!! Invariably the question asker either doesn't know what DNS is or doesn't understand how something fundamental works. More generally, people think that DNS is scary or complicated. This article is an attempt at quelling that fear. DNS is easy once you understand a few basic concepts.

What is DNS?

First things first: DNS stands for Domain Name System. Fundamentally, it's a globally distributed key value store. Servers around the world can give you the value associated with a key, and if they don't know they'll ask other servers for the answer.

That's it. That's all there is to it. You (or your web browser) ask for the value associated with the key `www.example.com` and get back `1.2.3.4`.

Basic Exploration and Fundamental Types

The great thing about the DNS is that it's completely public and open, so it's easy to poke around. Let's do a little exploring, starting with this domain, `petekeen.net`. Note that you can run all of these examples from an OS X or Linux command line.

First, let's look at a simple domain name to IP address mapping:

```
$ dig empoknor.bugsplat.info
```

The `dig` command is a veritable Swiss Army knife for querying DNS servers and we'll be using it quite a bit. Here's the first part of the response:

```
> <<>> DiG 9.7.6-P1 <<>> empoknor.bugsplat.info
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51539
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
```

There's only one interesting thing in here. We asked for one record and got exactly one response. Here's the question we asked:

```
;; QUESTION SECTION:
;empoknor.bugsplat.info. IN A
```

dig defaults to asking for A records. A stands for address and is one of the basic fundamental types of records in the DNS. An A record holds exactly one IPv4 address. There's an equivalent record for IPv6 addresses named AAAA. Next, let's look at the answer our DNS server gave us:

```
;; ANSWER SECTION:
empoknor.bugsplat.info. 300 IN A
192.30.32.165
```

This says the host `empoknor.bugsplat.info.` has exactly one A address: `192.30.32.165`. The `300` is called the TTL value, or time to live. It's the number of seconds that this record can be cached before it needs to be checked again. The `IN` component stands for Internet and is meant to disambiguate between the various types of networks that the DNS historically was responsible for. The rest of the response tells you things about the response itself:

```
;; Query time: 20 msec
;; SERVER:
192.168.1.1#53(192.168.1.1)
;; WHEN: Fri Jul 19 20:01:16 2013
;; MSG SIZE rcvd: 56
```

Specifically, it tells you how long it took for your server to respond, what that server's IP address is (`192.168.1.1`), what port **dig** asked (`53`, the default DNS port), when the query completed, and how many bytes the response contained.

As you can see, there's an awful lot going on in a single DNS query. Every time you open a web page your browser makes literally dozens of these queries to resolve the web host, all of the hosts where external resources like images and scripts are located, etc. Every single resource involves at least one DNS query, which would involve an awful lot of traffic if DNS wasn't designed to be heavily cached.

What you probably can't see, however, is that the DNS server at `192.168.1.1` contacted a whole chain of other servers in order to answer that simple question of what address does `empoknor.bugsplat.info` map to. Let's run a trace to see all of the servers that **dig** would have to contact if they weren't already cached:

```
$ dig +trace empoknor.bugsplat.info

; <<>> DiG 9.7.6-P1 <<>> +trace
empoknor.bugsplat.info
;; global options: +cmd
. 137375 IN NS 1.root-servers.net.
. 137375 IN NS m.root-servers.net.
. 137375 IN NS a.root-servers.net.
. 137375 IN NS b.root-servers.net.
```



```
. 137375 IN NS c.root-servers.net.  
. 137375 IN NS d.root-servers.net.  
. 137375 IN NS e.root-servers.net.  
. 137375 IN NS f.root-servers.net.  
. 137375 IN NS g.root-servers.net.  
. 137375 IN NS h.root-servers.net.  
. 137375 IN NS i.root-servers.net.  
. 137375 IN NS j.root-servers.net.  
. 137375 IN NS k.root-servers.net.  
;; Received 512 bytes from 192.168.1.1#53(192.168.1.1) in 189 ms
```

```
info. 172800 IN NS c0.info.aflias-nst.info.  
info. 172800 IN NS a2.info.aflias-nst.info.  
info. 172800 IN NS d0.info.aflias-nst.org.  
info. 172800 IN NS b2.info.aflias-nst.org.  
info. 172800 IN NS b0.info.aflias-nst.org.  
info. 172800 IN NS a0.info.aflias-nst.info.  
;; Received 443 bytes from 192.5.5.241#53(192.5.5.241) in 1224 ms
```

```
bugsplat.info. 86400 IN NS  
ns-1356.awsdns-41.org.  
bugsplat.info. 86400 IN NS  
ns-212.awsdns-26.com.  
bugsplat.info. 86400 IN NS  
ns-1580.awsdns-05.co.uk.  
bugsplat.info. 86400 IN NS  
ns-911.awsdns-49.net.  
;; Received 180 bytes from 199.254.48.1#53(199.254.48.1) in 239 ms
```

```
empoknor.bugsplat.info. 300 IN A 192.30.32.165  
bugsplat.info. 172800 IN NS  
ns-1356.awsdns-41.org.  
bugsplat.info. 172800 IN NS  
ns-1580.awsdns-05.co.uk.  
bugsplat.info. 172800 IN NS  
ns-212.awsdns-26.com.  
bugsplat.info. 172800 IN NS  
ns-911.awsdns-49.net.
```

```
;; Received 196 bytes from 205.25
1.195.143#53(205.251.195.143) in
15 ms
```

The DNS is arranged in a hierarchy. Remember how **dig** inserted a single . after the hostname we asked for before, `empoknor.bugsplat.info`? Well, that . is pretty important and stands for the root of the hierarchy. The root DNS servers are run by various companies and governments around the world. Originally there were only a handful of these servers, but as the Internet has grown more have been added, so that now there are notionally 13. Each one of these servers, however, has dozens or hundreds of physical machines hiding behind a single IP.

So, at the top of the trace we see the root servers, each represented by an NS record. An NS record maps a domain name, in this case the root, to a DNS server. When you register a domain name with a registrar like NameCheap or GoDaddy they create NS records for you.

dig randomly picked one of the root server responses, in this case `f.root-servers.net.`, and asked it what the A record for `empoknor.bugsplat.info` is and the root server responded with another set of NS servers. This time the ones responsible for the info top level domain . **dig** asks one of these servers for the A record for `empoknor.bugsplat.info`, gets back another set of NS servers, and then asks one of those

servers for the A record for `empoknor.bugsplat.info`. and finally receives an actual answer.

Whew! That would be a heck of a lot of traffic, except that almost all of these entries are cached for a long time by every server in the chain. Your computer caches too, as does your browser. Most of the time DNS resolution will never touch the root servers because their IP addresses hardly ever change. The top level domains: `com`, `net`, `org`, etc., are also generally heavily cached.

Other Types

There are a few other types that you should be aware of. The first is MX, which maps a domain name to one or more email servers. Email is so important to the functioning of the Internet that it gets its own record type. Here's the MX records for `petekeen.net`:

```
$ dig petekeen.net mx

; <<>> DiG 9.7.6-P1 <<>> petekeen.
net mx
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY,
status: NOERROR, id: 18765
;; flags: qr rd ra; QUERY: 1,
ANSWER: 2, AUTHORITY: 0, ADDI-
TIONAL: 0

;; QUESTION SECTION:
;petekeen.net. IN MX
```

```
;; ANSWER SECTION:
petekeen.net. 86400 IN MX 60
empoknor.bugsplat.info.
petekeen.net. 86400 IN MX 60
teroknor.bugsplat.info.

;; Query time: 272 msec
;; SERVER:
192.168.1.1#53(192.168.1.1)
;; WHEN: Fri Jul 19 20:33:43 2013
;; MSG SIZE rcvd: 93
```

Note that this time we got two answers because petekeen.net has two mail servers set up. The response is basically the same as the response for A.

The other record type that you should be familiar with is CNAME, which stands for Canonical Name and maps one name onto another. Let's look at the response we get for a CNAME:

```
$ dig www.petekeen.net
```

```
; <<>> DiG 9.7.6-P1 <<>> www.pete-
keen.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY,
status: NOERROR, id: 16785
;; flags: qr rd ra; QUERY: 1,
ANSWER: 2, AUTHORITY: 0, ADDI-
TIONAL: 0
```

```
;; QUESTION SECTION:
;www.petekeen.net. IN A
```

```
;; ANSWER SECTION:
www.petekeen.net. 86400 IN
```

```
CNAME empoknor.bugsplat.info.
empoknor.bugsplat.info. 300 IN A
192.30.32.165
```

```
;; Query time: 63 msec
;; SERVER:
192.168.1.1#53(192.168.1.1)
;; WHEN: Fri Jul 19 20:36:58 2013
;; MSG SIZE rcvd: 86
```

The first thing to notice is that we get back two answers. The first says that www.petekeen.net maps to empoknor.bugsplat.info. The second gives the A record for that server. One way to think about a CNAME is as an alias for another domain name.

Why CNAME is Messed Up

CNAMEs are incredibly useful, but they have one very important gotcha: if a CNAME exists for a particular name, that is the only record allowed for that name. No MX, no A, no NS, no nothing. This is because the DNS substitutes the CNAME's target for its own value, so every record valid for the target is also valid for the CNAME. This is why you can't have a CNAME on a root domain like petekeen.net, because you generally have to have other records for that domain like MX.

Querying Other Servers

Let's say for sake of argument that you messed up a DNS configuration. You think you've fixed the problem, but you don't want to wait for the cache to

expire to see. With **dig** you can actually query one of a number of public DNS servers instead of your default server like this:

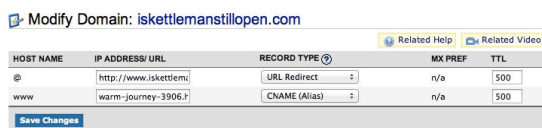
```
$ dig www.petekeen.net @8.8.8.8
```

The **@** symbol followed by an IP address or hostname tells **dig** to query that server on the default DNS port. I use this a lot to query Google's public DNS servers or Level 3's sort-of-public servers at 4.2.2.2.

Common Situations

Redirect bare domain to www

Almost always you'll want to redirect a bare domain like `iskettlemanstillopen.com` to `www.iskettlemanstillopen.com`. Registrars like Namecheap and DNSimple call this a URL Redirect. In Namecheap you would set up a URL Redirect like this:



HOST NAME	IP ADDRESS/ URL	RECORD TYPE	MX PREF	TTL
@	http://www.iskettlemanstillopen.com	URL Redirect	n/a	500
www	warm-journey-3906	CNAME (Alias)	n/a	500

The **@** stands for the root domain `iskettlemanstillopen.com`. Let's look at the A record for that domain:

```
$ dig iskettlemanstillopen.com
;; QUESTION SECTION:
;iskettlemanstillopen.com. IN A
```

```
;; ANSWER SECTION:
iskettlemanstillopen.com. 500 IN
A 192.64.119.118
```

That IP is owned by Namecheap and is running a small web server that just serves up an HTTP-level redirect to `http://www.iskettlemanstillopen.com`:

```
$ curl -I iskettlemanstillopen.com
curl -I iskettlemanstillopen.com
HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Fri, 19 Jul 2013 23:53:21 GMT
Content-Type: text/html
Connection: keep-alive
Content-Length: 154
Location: http://www.iskettlemanstillopen.com/
```

CNAME to Heroku or Github

Notice in the screenshot above that there's a second row defining a CNAME. In this case, `www.iskettlemanstillopen.com` maps to an application running on Heroku. You'll have to set up Heroku with a similar domain mapping, of course:

```
$ heroku domains
=== warm-journey-3906 Domain Names
warm-journey-3906.herokuapp.com
www.iskettlemanstillopen.com
```

Github is similar, except that the mapping lives in a file called CNAME at the root of your pages, as described in their documentation.

Wildcards

Most DNS servers allow you to set up DNS wildcards. For example, I have a wildcard CNAME set up for *.empoknor.bugsplat.info that maps to empoknor.bugsplat.info. That way I can host arbitrary things on empoknor and not have to create new DNS entries for them every time:

```
$ dig randomapp.empoknor.bugsplat.  
info
```

```
;; QUESTION SECTION:  
;randomapp.empoknor.bugsplat.info.  
IN      A
```

```
;; ANSWER SECTION:  
randomapp.empoknor.bugsplat.info.  
300 IN CNAME  empoknor.bugsplat.  
info.  
empoknor.bugsplat.info. 15  IN  A  
192.30.32.165
```

Wrap Up

Hopefully this gives you a good beginning understanding of what DNS is and how to go about exploring and verifying your configuration. ■

Pete Keen is a software developer currently residing in Ann Arbor. He recently published a book titled *Mastering Modern Payments: Using Stripe with Rails* and writes articles about a variety of technology issues at *petekeen.net*

Reprinted with permission of the original author.
First appeared in *hn.my/dnsgood* (petekeen.net)

FTP is so 90s. Let's deploy via Git instead!

By KERRICK LONG

FIRST, CREATE A directory on your server and initialize an empty git repository. I like to serve my websites from `~/www/`, so that's what I'll do in this example.

```
mkdir ~/www/example.com && cd ~/www/example.com
git init
```

Next, let's set up your server's git repo to nicely handle deployment via `git push`.

```
git config core.worktree ~/www/example.com
git config receive.denycurrent-branch ignore
```

Finally, we'll set up a post-receive hook for git to check out the `master` branch so your web server can serve files from that branch. (Remember, `^D` is Control+D, or whatever your shell's EOT character is).

```
cat > .git/hooks/post-receive
#!/bin/sh
git checkout -f
^D
chmod +x .git/hooks/post-receive
```

Keep in mind that you can add whatever you like to the post-receive hook if you have a build process. For example, one of my `sinatra` projects uses the following post-receive hook:

```
#!/bin/sh
git checkout -f
bundle install
touch ~/www/example.com/tmp/restart.txt
```

Back on your local machine, let's get your git repo ready for deployment.

```
cd ~/www-dev/example.com
git remote add origin \
ssh://user@example.com/home/user/
www/example.com
```

For the first push to your server, run the following command:

```
git push origin master
```

Now, whenever you want to deploy changes you've made locally, simply run the following command!

```
git push ■
```

With five years of experience in a combination of back- and front-end web development, an eye for user interface design, and a passion for learning, Kerrick bridges the gap between design and programming. He strives to make the web usable and accessible to all with open source, web standards, and good user experience.

Reprinted with permission of the original author.
First appeared in *hn.my/ftp* (coderwall.com)



Metrics and monitoring for people who know what they want

We know from experience that monitoring your servers and applications can be painful, so we built the sort of service that we would want to use. Simple to set up, responsive support from people who know what they're talking about, and reliably fast metric collection and dashboards.



Dashboards



StatsD



Happiness

Why Hosted Graphite?

- **Hosted metrics and StatsD:** Metric aggregation without the setup headaches
- **High-resolution data:** See everything like some glorious mantis shrimp / eagle hybrid*
- **Flexible:** Lots of sample code, available on Heroku
- **Transparent pricing:** Pay for metrics, not data or servers
- **World-class support:** We want you to be happy!

Promo code: **HACKER**

Grab a free trial at <http://www.hostedgraphite.com>

*Hosted Graphite's mantis shrimp / eagle breeding program has been unsuccessful thus far



HOSTEDGRAPHITE

How I Structure My Flask Applications

By MATT WRIGHT

FLASK [FLASK.POCOO.ORG] HAS been my preferred web framework as of late. I think it has a great core feature set, and Armin, the main author, has done well to keep its API minimal and easy to digest even for developers that are relatively new to Python. However, given that it is a rather minimal framework, it can often be difficult to decide how to structure an application after it reaches a certain level of complexity. It tends to be a common question that comes up in the #pocoo IRC channel.

In this article I intend to share how I structure Flask applications. To help support this article I've written a very basic application that I've arbitrarily named Overholt. [hn.my/overholt]

High Level Concepts

Platform vs. Application

Web applications can encapsulate a lot of different functionality. Most commonly when you think of a web application you probably think of a user interface rendered as HTML and JavaScript and displayed to a user in the browser. However, web applications can be infinitely more complicated. For instance, an application can expose a JSON API designed specifically for a Backbone.js front end application. There could also be a tailored JSON API for the native iOS or Android application. The list goes on. So when starting a project I try to think of it as a platform instead of an application. A platform consists of one or more applications. As a side note, this concept is not quite apparent in the Overholt application source code.

A Flask application is a collection of views, extensions, and configuration

This concept supports the previous. I look for logical contexts within the larger scope of my platform. In other words, I try to find patterns in the endpoints I will be exposing to various clients. Each of these contexts has slightly different concerns and thus I encapsulate their functionality and configuration into individual Flask applications. These applications can reside in the same code repository or can be separated. When it comes time to deploy the applications I then have the option to deploy them individually or combine them using Werkzeug's `DispatcherMiddleware`. In the case of Overholt, the platform consists of two Flask applications which are organized into separate Python packages: `overholt.api` and `overholt.frontend`.

Application logic is structured in logical packages and exposes an API of its own

I try to think of the application logic as a core “library” for my platform. Some developers call this a “service” layer. Regardless of what you call it, this layer sits on top of the data model and exposes an API for manipulating the data model. This allows me to encapsulate common routines that may be executed in multiple contexts within the platform. Using this approach also tends to lead to “thinner” view functions. In other words it allows me to

keep my view functions small and focused on transforming request data into objects that my the service layer expects in its API. Application logic within Overholt is primarily located in the `overholt.users`, `overholt.products`, and `overholt.stores` Python packages.

View functions are the layer between an HTTP request and the application logic

A view function is where an HTTP request meets the application logic. In other frameworks this layer is often called a “controller” or a “handler” and sometimes even a “command.” It is here that the data in the HTTP request is able to be accessed and used in conjunction with the API exposed by the application logic. The view function then renders a response according to the results of the application logic that was used. View functions within Overholt are organized using Blueprints. Each Blueprint has its own module within the application's Python package. An example of such a module is `overholt.api.products` or `overholt.frontend.dashboard`.

Patterns and Conventions

Flask tends not to push any patterns or conventions on the developer. This is one of the things I like most about Flask compared to large frameworks like Django and Rails. However, any developer not willing to establish patterns and conventions for their Flask

apps would be doing themselves or any other developers working on the project a disservice. Without patterns or conventions your applications will lose architectural integrity and be difficult for others to understand. After working with Flask for almost two years now I've settled on a few patterns and conventions of my own. The following is an overview of what I commonly use.

Application Factories

The factory pattern is the first pattern to be implemented and used in any of my Flask applications. There is a small amount of documentation regarding application factories already. While the documentation is limited in scope, I believe it is there to encourage the usage of this pattern. That being said, there is not an established convention for implementing a factory method. Chances are your app will have its own unique requirements and thus your factory method should be tailored accordingly. Regardless of your implementation the factory method is, in my opinion, indispensable as it gives you more control over the creation of your application in different contexts such as in your production environment or while running tests.

Within the Overholt source code you will find three different factory methods. There is one factory for each application and an additional factory which is shared by the individual application factories. The shared

factory instantiates the application and configures the application with options that are shared between apps. The individual app factories further configure the application with options that are more specific to their use. For example, the api application factory registers a custom `JSONEncoder` class and custom error handlers that render JSON responses. Whereas the frontend application factory initializes an assets pipeline and custom error handlers for HTTP responses.

Blueprints

Blueprints are crucial to my Flask applications as they allow me to group related endpoints together. I honestly couldn't live without Blueprints. The Flask documentation [[hn.my/blueprints](https://flask.palletsprojects.com/en/1.1.x/blueprints/)] provides the best overview of what Blueprints are and why they are useful. There isn't much else I can describe about Blueprints themselves that Armin hasn't already. In the context of the Overholt source code, each application package contains various modules containing Blueprint instances. The API application contains three Blueprints located at `overholt.api.products`, `overholt.api.stores` and `overholt.api.users`. The frontend application contains but one Blueprint located at `overholt.frontend.dashboard`. All Blueprint modules are located in the same package as the application which allows me to use a simple method of registering them on

their respective application. Within the shared application factory you should notice the `register_blueprints` helper method. This method simply scans all the modules in the application package for Blueprint instances and registers them on the app instance.

Services

Services are how I follow my third high level concept: *“Application logic is structured in logical packages and exposes an API of its own.”* They are responsible for connecting and interacting with any external data sources. External data sources include (but are not limited to) such things as the application database, Amazon’s S3 service, or an external RESTful API. In general each logical area of functionality (products, stores, and users) contains one or more services depending on the required functionality. Within the Overholt source code you will find a base class for services that manage a specific SQLAlchemy model. Furthermore, this base class is extended and additional methods are added to expose an API that supports the required functionality. The best example of this is the `overholt.stores.StoresService` class. Instances of service classes can instantiate at will, but as a convenience instances are consolidated into the `overholt.services` module.

API Errors/Exceptions

Dealing with errors in a RESTful API can be kind of annoying at times, but Flask makes it truly simple. Armin has already written a little bit about implementing API exceptions which I recommend you read. My implementation is not quite the same as his, but that’s the beauty of Flask. Overholt has a base error class and a slightly more specific error class related to form processing. Perhaps you recognize these errors if you view the source referenced in the application factories section. More specifically, the API application registers error handlers for these errors and returns a JSON response depending on the error that was raised. Dig around the source and see if you can find where they are raised.

View Decorators

Decorators in Python are very useful functional programming tools. In the context of a Flask application they are extremely useful for view functions. The Flask documentation provides a few examples [hn.my/viewdec] of some useful view decorators. Within the Overholt source there are two examples of view decorators that I commonly use. Each is tailored for using Blueprints and specific to each of the two applications. Take a look at the API view decorator. [hn.my/viewapi] This type of view decorator allows me to add all the other common decorators to my view methods. This prevents

me from having to repeat decorators, such as `@login_required`, across all the API views. Additionally, the decorator serializes the return value of my view methods to JSON. This also allows me to simply return objects that can be encoded by the API application's custom `JSONEncoder`.

Middleware

WSGI middlewares are pretty handy and can be used for all sorts of things. I have one middleware class that I always copy from project to project called `HTTPMethodOverrideMiddleware`. You can find it in the `overholt.middleware` module. This middleware allows an HTTP client to override the request method. This is useful for older browsers or HTTP clients that don't natively support all the modern HTTP verbs such as `PUT`, `DELETE` and `HEAD`.

JSON Serialization

If you've ever developed a JSON API you'll inevitably need to have control over how objects are represented as a JSON document. As mentioned earlier, the API application uses a custom `JSONEncoder` instance. This encoder adds additional support for objects that include the `JSONSerializer` mixin. This mixin defines a few "magic" variables which allow me to be explicit about the fields or attributes that are visible, hidden, or modified before being encoded as JSON. I simply need to extend this mixin, override the magic

variables with my options and include the new, extended mixin in the data model's inheritance chain. Examining any of the model modules within the `overholt.stores`, `overholt.products`, or `overholt.users` packages will illustrate how this mixin is used.

Database Migrations

In addition to using SQLAlchemy I always use Alembic. [alembic.readthedocs.org] Alembic is a nice database migration tool made specifically for SQLAlchemy by Mike Bayer, the author of SQLAlchemy. What's nice about Alembic is that it includes a feature to autogenerate database versions from the model metadata. If you examine the `alembic.env` module you should notice the application specific imports. Further down is where the application's database URI and model metadata is handed off to Alembic.

Configuration

Configuration is always important for an application, especially for sensitive details such as API keys and passwords. I always provide a default configuration file that is checked into the project repository so that a developer can get up and running as quick as possible. This file contains default values that are specific to the virtual machine settings specified in the Vagrantfile. This default file is used to configure any apps created by the shared application factory. Additionally, the factory

method attempts to override any default settings from a `settings.cfg` file located in the application's instance folder. This additional file can be created by any developer working on the project to tweak any settings to be more specific to their local development environment. When it comes time to deploy the application to a development or production server the `settings.cfg` file will be created by the deployment tool, such as Chef or Fabric.

Management Commands

Management commands often come in handy when developing or managing your deployed application. The Flask-Script extension makes setting up management commands pretty easy. Commands are useful in many ways such as manipulating data or managing the database. It's really up to you and your application's needs. Overholt contains a simple `manage.py` module at the top level of the project. There are three commands for managing users. As my applications grow, management commands tend to as well.

Asynchronous Tasks

Running code asynchronously is a common way of improving the responsiveness of a web application. Celery [celeryproject.org] is, arguably, the de facto library for doing this with Python. Similar to creating Flask apps, I also use a factory method for creating my

Celery apps. The thing to note about this factory method is that it specifies a custom task class. This custom class creates an application context before any task is run. This is necessary because task methods will most likely be using code that is shared by the web application. More specifically, a task might query or modify the database via the Flask-SQLAlchemy extension which requires an application context to be present when interacting with the database. Beyond this, tasks queued from within view functions. Overholt contains just a few example tasks to illustrate how they might be used.

Frontend Assets

When it comes to frontend assets, I always use webassets in conjunction with the Flask-Assets extension. These libraries allow me to create logical bundles of assets that, once compiled and minified, offers optimized versions for web browsers to keep the download times to a minimum. When it comes time to deploy the assets, there are two approaches. The first is simply to compile the assets locally and commit them to the project repository. The other is to compile the assets on the web server when the application is deployed. The first option has the advantage of not having to configure your web server with various tools (CoffeeScript, LESS, SASS, etc.) to compile the assets. The second option keeps compiled files out of the project repository and could

potentially prevent an error resulting from someone forgetting to compile new assets.

Testing

Testing your Flask applications is “important.” I’ve quoted the word “important,” though, and that’s because tests, while very useful, should not be your first concern. Regardless, when it comes time to write tests it should be relatively easy to do so. Additionally, I rarely write unit tests for my Flask applications. I generally only write functional tests. In other words, I’m testing that all application endpoints work as expected with valid and invalid request data.

Tools

In the Python world there are countless testing tools and libraries, and it’s often difficult to decide which ones to use. The only thing I strive for is to find the right balance of fewest dependencies and ease of testing. That being said, I’ve found that it’s pretty easy to get by using only the following tools:

nose [nose.readthedocs.org]

Running tests is a breeze with nose. It has a lot of options, and there is a wide variety of plugins that you may find useful. This library also seems to be widely used in the community, so I’ve settled on it as my preferred, top-level test tool.

factory_boy

[factoryboy.readthedocs.org]

Without test data/fixtures it will be difficult to test any app. **factory_boy** is a nice library that makes it trivial to create test data from the application’s models. Lately I’ve been using an older version and configured it to support SQLAlchemy. However, as of writing this, there is a newer version on the horizon that will support SQLAlchemy out of the box.

mock [mock.readthedocs.org]

I use this library the least, but it still comes in handy from time to time. This is why you’ll see it listed in the `requirements.txt` file but not yet used in the tests.

Structure

Without exception my Flask projects always contain a package named `tests` where all test-related code is placed. In the top level of the `test` package you will see a few base classes for test cases. Base classes are extremely useful for testing because there is inevitably always repeated code in tests.

There are also a few modules in this package. One being `tests.settings` which is a testing-specific configuration module. This module is passed to each application’s factory method to override any default settings. The `tests.factories` module contains factory classes which utilize the aforementioned `factory_boy` library. Lastly you’ll

find the `tests.utils` module. This module will hold all reusable test utilities. For now it contains a simple function to generate a basic HTTP auth header and a test case mixin class that has many useful assertion and request methods.

Also within the top level `tests` package are two other packages, `tests.api` and `tests.frontend` which map to the two applications that are part of Overholt. Within the top level of each package is another base class which inherits from `tests.OverholtAppTestCase`. This class can then be modified to add common testing code for the respective application. Each application then has a varying amount of test modules that group the testing of endpoints. For instance, the `tests.api.product_tests` module contains the `ProductApiTestCase` class which tests all the product-related endpoints of the API application.

Documentation

The last and most commonly neglected part of any project is documentation. Sometimes you can get away with a small README file. The Overholt project happens to contain a small README file that explains how to setup the local development environment. However, README files are not necessarily sustainable as a project's complexity grows. When this is the case I always turn to Sphinx. [sphinx-doc.org]

All documentation files reside in the `docs` folder. These files can then be used by Sphinx to generate HTML (and other formats). There are also a lot of extensions out there for Sphinx. The extension I most commonly use is `sphinxcontrib-httpdomain`. This extension is geared specifically for documenting HTTP APIs and even has the ability to generate documentation for a Flask application. You can see this extension in action in the Overholt API documentation file.

Wrap Up

I believe that the age-old saying “there is more than one way to skin a cat” holds true to developing any application, let alone a web application with Flask. The approach outlined here is based on my personal experience developing, what I would consider, relatively large applications with Flask. What works for me might not work for you, but I'd like to think there is some useful information here for developers getting into Flask. ■

Matt Wright is a software and devops engineer at ChatID [chatid.com]. Prior to ChatID he has worked as a developer for a variety of companies which include Local Projects [localprojects.net] and Rokkan [rokkan.com]. In his spare he maintains a variety of extensions for Flask and chronicles his work experiences on his blog at mattupstate.com

Reprinted with permission of the original author.
First appeared in *hn.my/flask* (mattupstate.com)

What Makes Lua Tick

By MILES BADER

L_{UA}:

- 1** Is very small, both source and binary, an order of magnitude or more smaller than many more popular languages (Python, etc.). Because the Lua source code is so small and simple, it's perfectly reasonable to just include the entire Lua implementation in your source tree, if you want to avoid adding an external dependency.
- 2** Is very fast. The Lua interpreter is much faster than most scripting languages (again, an order of magnitude is not uncommon), and LuaJIT2 is a very good JIT compiler for some popular CPU architectures (x86, ppc). Using LuaJIT can often speed things up by another order of magnitude, and in many cases, the result approaches the speed of C. LuaJIT is also a “drop in” replacement for standard Lua: no application or user code changes are required to use it.
- 3** Has LPEG. LPEG is a “Parsing Expression Grammar” library for Lua, which allows very easy, powerful, and fast parsing, suitable for both large and small tasks; it's a great replacement for yacc/lex/hairy-regexps. [I wrote a parser using LPEG and LuaJIT, which is much faster than the yacc/lex parser I was trying emulate, and was very easy and straight-forward to create.] LPEG is an add-on package for Lua, but it is well-worth getting (it's one source file).
- 4** Has a great C-interface, which makes it a pleasure to call Lua from C, or call C from Lua. For interfacing large/complex C++ libraries, one can use SWIG, or any one of a number of interface generators (one can also just use Lua's simple C interface with C++ of course).

5 Has liberal licensing (“BSD-like”), which means Lua can be embedded in proprietary projects if you wish, and is GPL-compatible for FOSS projects.

6 Is very, very elegant. It’s not lisp, in that it’s not based around cons-cells, but it shows clear influences from languages like scheme, with a straight-forward and attractive syntax. Like scheme (at least in its earlier incarnations), it tends towards “minimal” but does a good job of balancing that with usability. For somebody with a lisp background (like me!), a lot about Lua will seem familiar, and “make sense,” despite the differences.

7 Has a simple, attractive, and approachable syntax. This might not be such an advantage over lisp for existing lisp users but might be relevant if you intend to have end-users write scripts.

8 Has a long history, and responsible and professional developers who have shown good judgment in how they’ve evolved the language over the last 2 decades.

9 Has a vibrant and friendly user-community. ■

Miles is a long-time user and developer of free software.

Reprinted with permission of the original author.
First appeared in *hn.my/lua* (lua-users.org)

What Every Web Developer Must Know About URL Encoding

By STÉPHANE ÉPARDAUD

THIS ARTICLE DESCRIBES common misconceptions about Uniform Resource Locator (URL) encoding, then attempts to clarify URL encoding for HTTP, before presenting frequent problems and their solutions. While this article is not specific to any programming language, we illustrate the problems in Java and finish by explaining how to fix URL encoding problems in Java, and in a web application at several levels.

Introduction

There are a number of technologies we use every day when we browse the web. There is the data itself (the web pages) obviously, the formatting of this data, and the transport mechanism which allows us to retrieve this data.

Then there is the foundation, the root, the thing that makes the web a web: links from one page to the other. These links are URLs.

General URL syntax

Everyone by now has seen a URL at least once in his life. Take “http://www.google.com” for instance. This is a URL. A URL is a Uniform Resource Locator and is really a pointer to a web page (in most cases). URLs actually have a very well-defined structure since the first specification in 1994.

We can extract detailed information about the “http://www.google.com” URL:

Part	Data
Scheme	http
Host address	www.google.com

If we look at a more complex URL such as “https://bob:bobby@www.lunatech.com:8080/file;p=1?q=2#third” we can extract the following information:

Part	Data
Scheme	https
User	bob
Password	bobby
Host address	www.lunatech.com
Port	8080
Path	/file
Path params	p=1
Query params	q=2
Fragment	third

The Scheme (here http and https (secure HTTP)) define the structure of the rest of the URL. Most internet URL schemes have a common first part which indicates the user, password, host name and port, followed by a scheme-specific part. This common part deals with authentication and being able to know where to connect in order to request data.

HTTP URL syntax

For HTTP URLs (with the http or https schemes), the scheme-specific part of the URL defines the path to the data, followed by an optional query and fragment.

The path part consists of a hierarchical view similar to a file system hierarchy with folders and files. The path starts with a “/” character, then each folder is separated from one another

by a “/” again until we reach the file. For example “/photos/egypt/cairo/first.jpg” has four path segments: “photos”, “egypt”, “cairo” and “first.jpg”, which can be extrapolated as: the “first.jpg” file in the “cairo” folder, which is in the “egypt” folder located in the “photos” folder at the root of the web site.

Each path segment can have optional path parameters (also called matrix parameters) which are located at the end of the path segment after a “;”, and separated by “,” characters. Each parameter name is separated from its value by the “=” character like this: “/file;p=1” which defines that the path segment “file” has a path parameter “p” with the value “1”. These parameters are not often used — let’s face it — but they exist nonetheless, and we’ve even found a very good justification for their use in a Yahoo RESTful API document:

Matrix parameters enable the application to retrieve part of a collection when calling an HTTP GET operation. See Paging a Collection for an example. Because matrix parameters can follow any collection path segment in a URI, they can be specified on an inner path segment.

After the path segments we can find the query which is separated from the path with a “?” character, and contains a list (separated by “&”) of parameter names and values separated by “=”. For example “/file?q=2” defines a query

parameter “q” with the value “2”. This is used a lot when submitting HTML forms, or when calling applications such as Google search.

Last in an HTTP URL is the fragment which is used to refer not to the whole HTML page but to a specific part within that file. When you click on a link and the browser automatically scrolls down to display a part which was not visible from the top of the page, you have clicked a URL with a fragment part.

URL grammar

The http URL scheme was first defined in RFC 1738 (actually even before in RFC 1630) and while the http URL scheme has not been redefined, the whole URL syntax has been generalized into Uniform Resource Identifiers (URIs) from a specification that has been extended a few times to accommodate for evolutions.

There is a grammar which defines how URLs are assembled, and how parts are separated. For instance, the “://” part separates the scheme from the host part. The host and path fragments parts are separated by “/”, while the query part follows a “?”. This means that certain characters are reserved for the syntax. Some are reserved for all URIs, while some are only reserved for specific schemes. All reserved characters that are used in a part where they are not allowed (for instance a path segment — a file name for example

— which would contain a “?” character) must be URL-encoded.

URL-encoding is the transformation of a character (“?”) into a harmless representation of this character which has no syntactic meaning in the URL. This is done by converting the character into a sequence of bytes in a specific character encoding, then writing these bytes in hexadecimal preceded by “%”. A question mark in URL-encoding is therefore “%3F”.

We can write a URL pointing to the “to_be_or_not_to_be?.jpg” image as such: “http://example.com/to_be_or_not_to_be%3F.jpg” which makes sure that nobody would think there might be a query part in there.

Most browsers nowadays display the URLs by decoding (converting percent-encoded bytes back to their original characters) them first, while keeping them encoded when fetching them for the network. This means users are almost never aware of such encoding.

Developers or web page authors, on the other hand, have to be aware of it, because there are many pitfalls.

Common pitfalls of URLs

If you are working with URLs, it pays to know some of the most common traps you should avoid. Here we give a non-exhaustive list of some of those traps.

Which character encoding?

URL-encoding does not define any particular character encoding for percent-encoded bytes. Generally ASCII alphanumeric characters are allowed unescaped, but for reserved characters and those that do not exist in ASCII (the French “œ” from the word “nœud” — “knot” — for instance), we have to wonder which encoding to use when converting them to percent-encoded bytes.

Of course the world would be easier if they were just Unicode, because every character exists in this set, but this is a set — a list if you will — and not an encoding per se. Unicode can be encoded using several encodings such as UTF-8 or UTF-16 (there are several others), but then the problem is still there: which encoding should URLs (generally URIs) use?

The standards do not define any way by which a URI might specify the encoding it uses, so it has to be deduced from the surrounding information. For HTTP URLs it can be the HTML page encoding or HTTP headers. This is often confusing and a source of many errors. In fact, the latest version of the URI standard defines that

new URI schemes use UTF-8, and that host names (even on existing schemes) also use this encoding, which really rouses my suspicion: can the host name and the path parts really use different encodings?

The reserved characters are different for each part

Yes they are. Yes they are. Yes they are.

For HTTP URLs, a space in a path fragment part has to be encoded to “%20” (not, absolutely not “+”), while the “+” character in the path fragment part can be left unencoded.

Now in the query part, spaces may be encoded to either “+” (for backwards compatibility: do not try to search for it in the URI standard) or “%20” while the “+” character (as a result of this ambiguity) has to be escaped to “%2B”.

This means that the “blue+light blue” string has to be encoded differently in the path and query parts: “http://example.com/blue+light%20blue?blue%2Blight+blue”. From there you can deduce that encoding a fully constructed URL is impossible without a syntactical awareness of the URL structure.

Suppose the following Java code to construct a URL:

```
String str = "blue+light blue";
String url = "http://example.com/"
+ str + "?" + str;
```

Encoding the URL is not a simple iteration of characters in order to escape those that fall outside of the reserved set: we have to know which reserved set is active for each part we want to encode.

This means that most URL-rewriting filters would be wrong if they decide to take a URL substring from one part into another without proper encoding care. It is impossible to encode a URL without knowing about its specific parts.

The reserved characters are not what you think they are

Most people ignore that “+” is allowed in a path part and that it designated the plus character and not a space. There are other surprises:

- “?” is allowed unescaped anywhere within a query part,
- “/” is allowed unescaped anywhere within a query part,
- “=” is allowed unescaped anywhere within a path parameter or query parameter value, and within a path segment,
- “: @ - . _ ~ ! \$ & ' () * + , ; =” are allowed unescaped anywhere within a path segment part,
- “/? : @ - . _ ~ ! \$ & ' () * + , ; =” are allowed unescaped anywhere within a fragment part.

While this is slightly nuts and “http://example.com/:@-._~!\$&'()*+;:@-._~!\$&'()*+;=@-._~!\$&'()*+;=?/?:@-._~!\$'()*+;=?/@-._~!\$'()*+;==#/?:@-._~!\$&'()*+;=” is a valid HTTP URL, this is the standard.

For the curious, the previous URL expands to:

Part	Value
Scheme	http
Host	example.com
Path	/:@-._~!\$&'()*+;=
Path parameter name	:@-._~!\$&'()*+;
Path parameter value	:@-._~!\$&'()*+;==
Query parameter name	/?:@-._~!\$'()* ,;
Query parameter value	/?:@-._~!\$'()* ,;==
Fragment	/?:@-._~!\$&'()*+;=

Nuts.

A URL cannot be analyzed after decoding

The syntax of the URL is only meaningful before it is URL-decoded: after URL-decoding, reserved characters may appear.

For example “http://example.com/blue%2Fred%3Fand+green” has the following parts before decoding:

Part	Value
Scheme	http
Host	example.com
Path segment	blue%2Fred%3Fand+green
Decoded Path	blue/red?and+green

Thus, we are looking for a file called “blue/red?and+green”, not for the “red?and+green” file of the “blue” folder.

If we decode it to “http://example.com/blue/red?and+green” before analysis the parts would give:

Part	Value
Scheme	http
Host	example.com
Path segment	Blue
Path segment	Red
Query params name	and green

This is clearly wrong: analysis of reserved characters and URL parts has to be done before URL-decoding. The implication is that URL-rewriting filters should never decode a URL before attempting to match it if reserved characters are allowed to be URL-encoded (which may or may not be the case depending on your application).

Decoded URLs cannot be re-encoded to the same form

If you decode “http://example.com/blue%2Fred%3Fand+green” to “http://example.com/blue/red?and+green” and proceed to encode it (even with an encoder which knows about each syntactical URL part) you will get “http://example.com/blue/red?and+green” because that is a valid URL. It just happens to be very different from the original URL we decoded.

Handling URLs correctly in Java

When you have mastered your black belt in URL-fu you will notice that there are still quite a few Java-specific pitfalls when it comes to URLs. The road to URL handling correctness is not for the faint of heart.

Do not use `java.net.URLEncoder` or `java.net.URLDecoder` for whole URLs

We are not kidding. These classes are not made to encode or decode URLs, as their API documentation clearly says:

Utility class for HTML form encoding. This class contains static methods for converting a String to the application/x-www-form-urlencoded MIME format. For more information about HTML form encoding, consult the HTML specification.

This is not about URLs. At best it resembles the query part encoding. It is wrong to use it to encode or decode entire URLs. You would think the standard JDK had a standard class to deal with URL encoding properly (part by part, that is) but either it is not there, or we have not found it, which lures a lot of people into using `URLEncoder` for the wrong purpose.

Do not construct URLs without encoding each part

As we have already stated: fully constructed URLs cannot be URL-encoded.

Take the following code for instance:

```
String pathSegment = "a/b?c";
String url = "http://example.com/"
+ pathSegment;
```

It is impossible to convert “http://example.com/a/b?c” back to what it should have been if “a/b?c” was meant to be a path segment, because it happens to be a valid URL. We have already explained this earlier.

Here is the proper code:

```
String pathSegment = "a/b?c";
String url = "http://example.com/"
+ URLEncoder.encode(pathSegment);
```

We are now using a utility class `URLUtils` which we had to make ourselves for lack of finding an exhaustive one available online fast enough. The previous code will give you the

properly encoded URL “http://example.com/a%2Fb%3Fc”.

Note that the same applies to the query string:

```
String value = "a&b==c";
String url = "http://example.
com/?query=" + value;
```

This will give you “http://example.com/?query=a&b==c” which is a valid URL, but not the “http://example.com/?query=a%26b==c” we wanted.

Do not expect `URI.getPath()` to give you structured data

Since once a URL has been decoded, syntactical information is lost, the following code is wrong:

```
URI uri = new URI("http://example.
com/a%2Fb%3Fc");
for(String pathSegment : uri.get-
Path().split("/"))
    System.err.println(pathSegment);
```

It would first decode the path “a%2Fb%3Fc” into “a/b?c”, then split it where it should not have been split into path segment parts.

The correct code of course uses the undecoded path:

```
URI uri = new URI("http://example.
com/a%2Fb%3Fc");
for(String pathSegment : uri.get-
RawPath().split("/"))
    System.err.println(URLUtils.dec-
odePathSegment(pathSegment));
```

Do note that path parameters will still be present: deal with them if required.

Do not expect Apache Commons HTTPClient's URI class to get this right

The Apache Commons HTTPClient 3's URI class uses Apache Commons Codec's `URLEncoder` for URL-encoding, which is wrong as their API documentation mentions since it is just as wrong as using `java.net.URLEncoder`. Not only does it use the wrong encoder, but it also decodes each part as if they all had the same reserved set.

Fixing URL encoding at every level in a web application

We have had to fix quite a few URL-encoding issues in our application lately, from support in Java down to the lower level of URL rewriting. We will list here a few of changes which were required.

Always encode URLs as you build them

In our HTML files, we replaced all occurrences of this:

```
var url =
"#{v1:encodeURL(contextPath + '/view/' + resource.name)}";
```

with:

```
var url = "#{contextPath}/view/#{v1:encodeURLPathSegment(resource.name)}";
```

And similarly for query parameters.

Make sure your URL-rewrite filters deal with URLs correctly

Url Rewrite Filter is a URL rewriting filter we use in Seam to transform pretty URLs into application-dependent URLs.

For example, we use it to rewrite `http://beta.visiblelogistics.com/view/resource/FOO/bar` into `http://beta.visiblelogistics.com/resources/details.seam?owner=FOO&name=bar`.

Obviously this involves taking some strings from one URL part to another, which means we have to decode from the path segment part and reencode as a query value part.

Our initial rule looked as follows:

```
<urlrewrite decode-using="utf-8">
  <rule>
    <from>^/view/resource/(.*)/
    (.*)$</from>
    <to encode="false">/resources/
    details.seam?owner=$1&name=$2</to>
  </rule>
</urlrewrite>
```

It turns out that there are only two ways to deal with URL-decoding in Url Rewrite Filter: either every URL is decoded prior to doing the rule matching (the `<to>` patterns), or it is disabled and each rule has to deal with decoding. In our opinion the latter is the sanest option, especially if you move URL parts around, and/or want to match path segments which may contain URL-encoded path separators.

Within the replacement pattern (the `<to>` patterns) you can then deal with URL encoding/decoding using the inline functions `escape(String)` and `unescape(String)`.

As of this writing, Url Rewrite Filter Beta 3.2 contains several bugs and limitations which blocked our progress towards URL-correctness:

- URL decoding was done using `java.net.URLDecoder` (which is wrong),
- the `escape(String)` and `unescape(String)` inline functions used `java.net.URLDecoder` and `java.net.URLEncoder` (which is not specific enough and will only work for entire query strings, bearing in mind any “&” or “=” will not be encoded).

We therefore made a big patch fixing a few issues like URL decoding, and adding the inline functions `escapePathSegment(String)` and `unescapePathSegment(String)`.

We can now write the almost correct:

```
<urlrewrite decode-using="null">
<rule>
  <from>^/view/resource/(.*)/(.*)$</from>
  <-- Line breaks inserted for readability -->
  <to encode="false">/resources/details.seam
    ?owner=${escape:${unescapePath:$1}}
    &name=${escape:${unescapePath:$2}}</to>
</rule>
</urlrewrite>
```

It is only almost correct because our patch still lacks a few things:

- the inline escaping/unescape functions should be able to specify the encoding as either fixed (this is already done) or by determining it from the HTTP call (not supported yet),
- the old `escape(String)` and `unescape(String)` inline functions were left intact and still call `java.net.URLDecoder` which is wrong as it will not escape “&” or “=”,
- we need to add more part-specific encoding/decoding functions,
- we need to add a way to specify the decoding behavior per-rule as opposed to globally in `<urlrewrite>`.

As soon as we get the time, we will send a second patch.

Using Apache mod-rewrite correctly

Apache mod-rewrite is an Apache web server module for URL-rewriting which we use to proxy all our `http://beta.visiblelogistics.com/foo` traffic to `http://our-internal-server:8080/v1/foo` for instance.

This is the last thing to fix, and just like `Url Rewrite Filter`, it defaults to decoding the URL for us, and reencoding the rewritten URL for us, which is wrong, as decoded URLs cannot be reencoded.

There is one way to get around this, however. Since we are not switching one URL part for another in our case, we do not need to decode a path part and reencode it into a query part. For example: do not decode and do not reencode.

We accomplished it by using `THE_REQUEST` for URL-matching which is the full HTTP request (including the HTTP method and version) undecoded. We just take the URL part after the host, change the host and prepend the `/v1` prefix, and tada:

...

```
# This is required if we want to
allow URL-encoded slashes a path
segment
```

```
AllowEncodedSlashes On
```

```
# Enable mod-rewrite
RewriteEngine on
```

```
# Use THE_REQUEST to not decode
the URL, since we are not moving
# any URI part to another part so
we do not need to decode/reencode
```

```
RewriteCond %{THE_REQUEST} "^[a-
zA-Z]+ /(.*?) HTTP/\d\.\d$"
RewriteRule ^(.*)$ http://
our-internal-server:8080/v1/%1
[P,L,NE] ■
```

From deep into the Nice mountains, Stéphane works for Red Hat on the Ceylon project. Passionate hacker in Java, C, Perl or Scheme. A web standards and database enthusiast, he implemented among other things a WYSIWYG XML editor, a multi-threading library in C, a mobile-agent language in Scheme (compiler and virtual machines), and some Web 2.0 RESTful services and rich web interfaces with JavaScript and HTML 5.

Reprinted with permission of the original author.
First appeared in hn.my/urlencoding (lunatech.com)

Using Katas to Improve Your Coding

By CHONG KIM

THERE IS AN experiment from Richard Held and Alan Hein who raised kittens in total darkness. For a short period during the day, the kittens were placed in a carousel apparatus where the lights were turned on. One basket allowed the kitten to see and interact with its environment (the active kitten). The other had a hole for the head so the kitten (the passive kitten) can have the same visual experience but without the interaction.

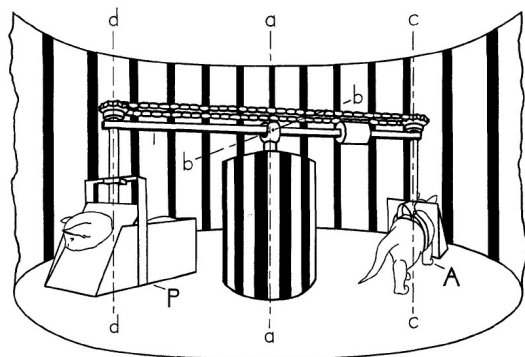


FIG. 1. Apparatus for equating motion and consequent visual feedback for an actively moving (A) and a passively moved (P) S.

At the end of the experiment, the passive kitten was functionally blind whereas the active kitten was normal.

This idea has stuck with me — the idea that you need to interact with your environment. You are functionally blind when you only have book knowledge. I need to code (and code a lot) to really get that knowledge at the instinctive level.

It is also important to do things quickly to develop fluency. Fluency allows you the freedom from the mechanics of what you are doing so you can focus on the main ideas.

First Kata Experiment

When I joined 8th Light, I came across katas. The idea of a kata is to practice coding by doing it repetitively. You build muscle memory in the mechanics of coding — setting up the editor, reacting to errors, letting your fingers get used to the controls. Initially I thought it was an amusing

little activity. Then I started to wonder if I can do a kata on something more than coin-changer or roman numerals, something with a little more substance. I thought it could be possible to write tic-tac-toe as a kata. My main goal is to develop a workflow so I can write it in under an hour. I also wanted to record myself and bought Screenflow. After all, the kata is meant to be a performance.

I did my first tic-tac-toe kata in Ruby, a language I know well. I used RSpec as my testing framework, something I was less familiar with.

In the beginning I spent a lot of time setting up my testing environment and researching the web when I got stuck. For example, I forgot to add an “it” block in RSpec, which generated an error message I couldn’t understand. I worked around it by making the test pass. I didn’t figure it out until the next day when a co-worker (Meagan) pointed it out after she saw the video of my kata.

I saw myself steadily improve in my next version of the kata. I made 6 attempts before I was finally able to do it in less than an hour. I was able to interpret error messages better. I was able to set up my testing environment faster. I improved my code by finding more elegant ways to solve a problem. For instance, if I wanted to separate a list into groups of 3, I would write:

```
lst.group_by.with_index {|e,i|  
i/3}.values
```

After having written it several times, I thought there was probably a better way. I finally came up with:

```
lst.each_slice(3).to_a
```

I don’t know if I would have revisited this problem if it weren’t for the kata.

I also saw the effect when I forgot to add a test. This brings another important point. I saw probably every type of error/bug because each time I do a kata, I make different mistakes. You get a richer experience from it. You’re able to focus on the source of the error/bug rather than wonder about the correctness of your code; after all, your code is similar to your previous versions so you know it should have worked. You can always use diff to compare your versions if you get completely stuck.

Since the kata is repetitive, it allows you to reflect on how you use your editor. You wonder if there is a better way to get from one point to another. You can test out new keystrokes and see if it makes a difference.

In the end, I would say the kata has vastly improved my workflow.

Using Kata to Learn a New Language

I tried another experiment. What would it be like to use a kata to learn a new language? Would I be able to do get it done under an hour? I tried it out with Haskell.

I've heard people mention Haskell so I wanted to give it a go. The first step was to find a resource to read up on it. I read through Learn You A Haskell [learnyouahaskell.com]. After a few days, I was ready to start coding. Just like the passive kitten, I was functionally blind. I knew about Haskell, but I couldn't code it. I needed to interact with the language. What better way than to do a kata?

I already had a set of routines I wanted to code and I knew the algorithm. The only thing standing in my way was the language, and the kata gave me a controlled environment so I could focus on it. It also gave me a good way to reflect on the problems I encountered. Since my errors were recorded, I didn't have to remember things I needed to look into or remember what error messages led me to a particular fix. It's like having superhuman memory.

After I got the setup for testing out of the way (using HSpec), syntax became my main problem. Every time I wrote something, I would get parse errors. I would backtrack to a simpler form until I got it to work. After about an hour, I was only able to write a constructor. I also had to set up guard (for automatic testing), which took up a good chunk of time. I kept my recordings to about an hour for the rest of my katas whether I finished or not.

When I reviewed the video of my first kata, I saw long pauses where I was thinking about a particular issue. Then I saw myself researching and eventually solving the problem. The video reinforced everything I had learned. I didn't have to take notes or remember how the problem originated. It was all recorded for me. This allowed me the freedom to try new techniques and go beyond my comfort zone. I can always review it and see where things went wrong.

When you learn a new language, you have a feeling that you know enough to do small things but you have the uneasy feeling that your knowledge is tenuous, that it can slip away from you if you're not paying attention. That feeling started to evaporate on the second kata. I developed idioms so I could do things automatically. That gave me a base to build on. By the third kata, the video showed a steep increase in my performance. I was no longer hesitating and going off to Google. I was still making a lot of errors, but they were different errors. The ones I had encountered before were quickly dismissed since I had already solved it in the past.

Since I was new to the language, I was not able to complete the program in the early katas. It did give me good research points when I finished recording. I knew that I could at least get to the same point as the previous kata. I

needed just a little bit more knowledge to go further. It is very encouraging when you can see yourself actually improve over each version.

It took me 10 tries before I was able to get a complete working version of my code. I was able to do it just a little over my hour target. I sped up the time on my Screenflow so it played for a bit over 16 minutes. When I tried uploading it to YouTube, it got rejected because they had a limit of 15 minutes. I knew I had to shave off 20 minutes so my video could run in 14 minutes. I was already typing as fast as I could. Then I realized that I could use the abbreviate command in vim. I made it so when I typed “p”, it would type out “position” and that would save me keystrokes. I would add these abbreviations as I went along. I spent my off-kata time looking into shortcuts in vim.

I did some unrecorded katas to test out some new key bindings and some config changes for vim. My 13th kata was the charm. I was able to finish in a little under 50 minutes, which reduced my time-compressed video to 14 minutes. I uploaded it to YouTube and was approved finally.

[hn.my/tictactoe]

I encourage everyone to try using katas to improve their workflow. You’ll be amazed at what you can get accomplished. ■

Chong Kim is a Software Craftsman. He is interested in programming, math and chess.

Reprinted with permission of the original author.
First appeared in hn.my/kata (8thlight.com)



EMAIL FOR YOUR APPS

SEND. TRACK. DELIVER.



Your one stop shop for **ALL** your email needs.

Manage lists as well. No extra fees for **Newsletters**.

Priority headers to deliver notifications in **real time**.

Go for

mailjet.com

You push it,
we test it,
& deploy it.



CircleCI is offering a special discount for Hacker Monthly readers. Follow the URL below and take advantage of a 50% discount for your first three months.
<https://circleci.com?=hackermonthly>