50 Lego Designs with 50 Pieces

HACKERMONTHLY

Issue 16
September 2011

Frustrated with customer support?

Imagine how your customers feel. Our simple engagement tools help you understand your customers, prioritize feedback, and give great customer support faster.

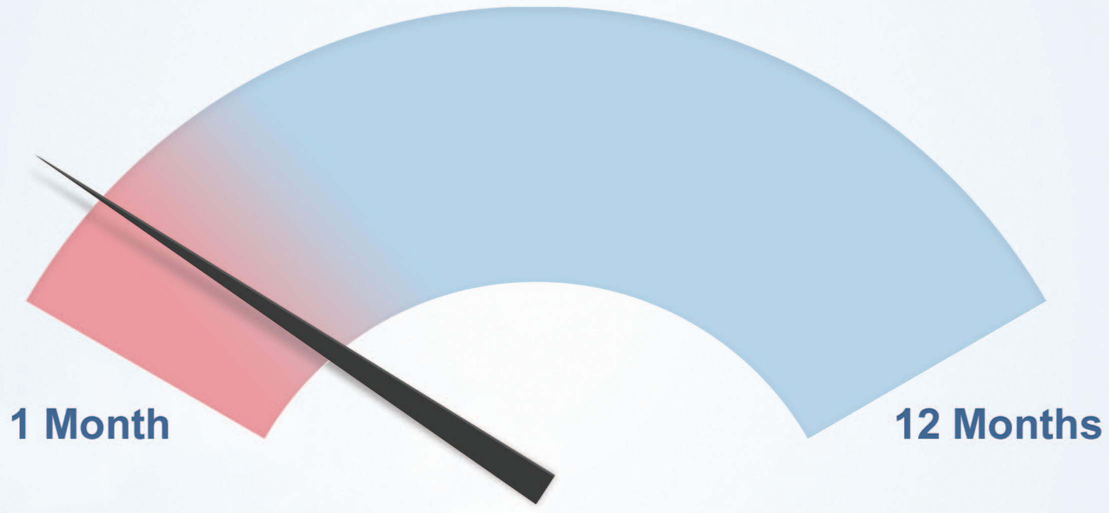
Spend more time building a product your customers will love!

uservoice

Get 50% off your first 3 months* with the code **happymonkeys** at UserVoice.com.

* Offer good for new accounts if used before 12/31/2011.

WHEN DOES YOUR STARTUP RUN OUT OF CASH?



inDinero

inDinero.com/hackermonthly

Curator
Lim Cheng Soon

Contributors
Tyler Neylon
Mark Suster
Jason Freedman
Gabriel Weinberg
Alex Feinberg
Joe Armstrong
Dan Benjamin
Kalid Azad

Proofreader
Emily Griffin

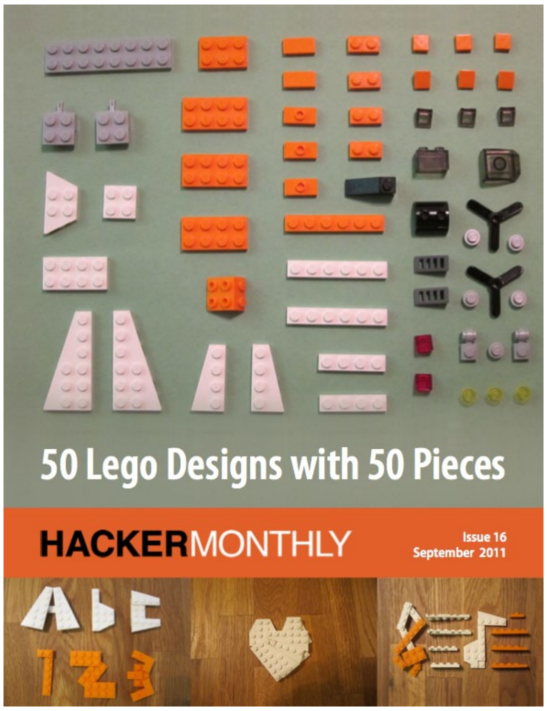
Printer
MagCloud

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

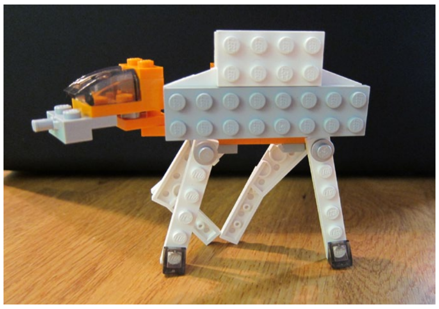
Contact
contact@hackermonthly.com

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



Cover Images: Tyler Neylon

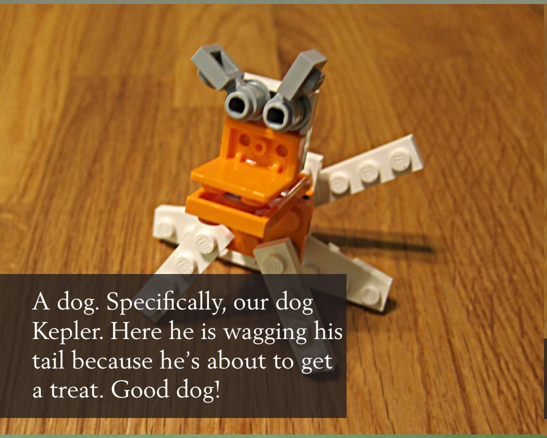
Contents



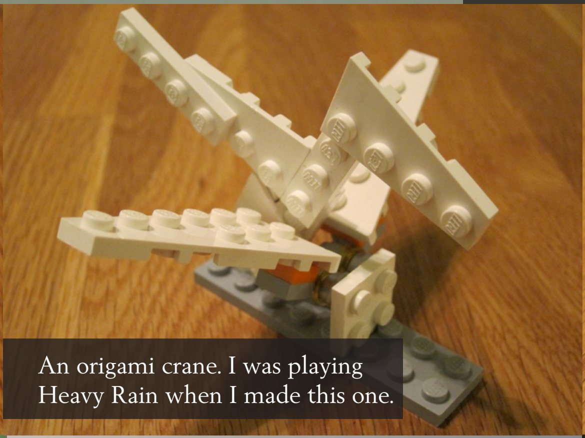
50 Lego Designs with 50 Pieces

By TYLER NEYLON

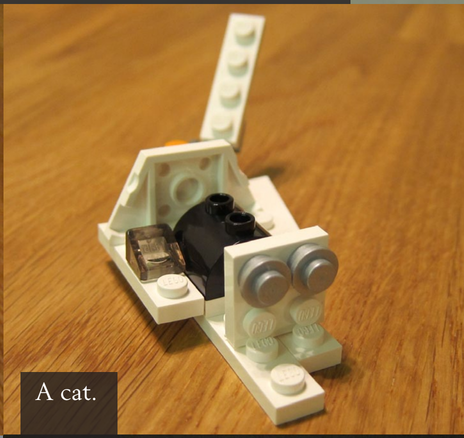
LAST CHRISTMAS, A friend gave me a small Lego set just for fun. It's part of a Creator series, where each set comes with instructions for three different models. I love the versatility and expressiveness of Lego, and I think 3 is far too small a number for what can be done with these, so I challenged myself to create 50 original designs with this one set of about 50 pieces. These images are the result.



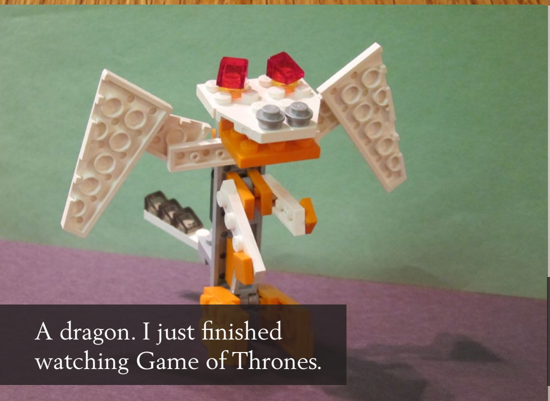
A dog. Specifically, our dog Kepler. Here he is wagging his tail because he's about to get a treat. Good dog!



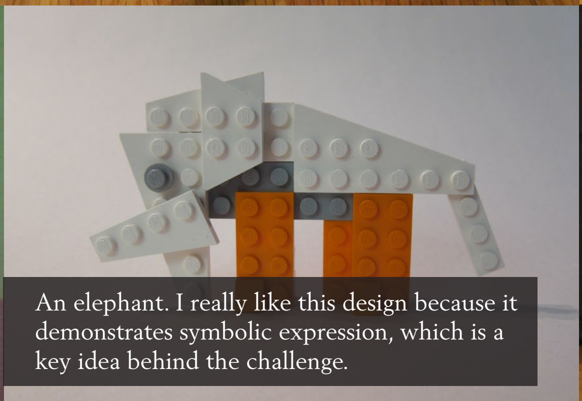
An origami crane. I was playing Heavy Rain when I made this one.



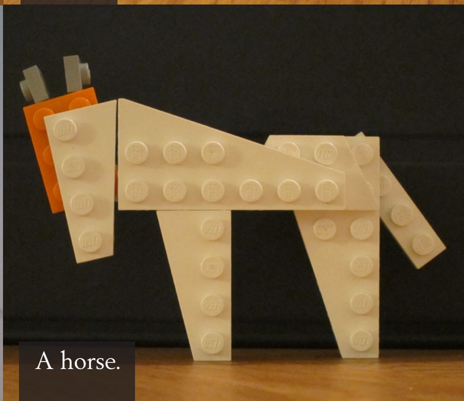
A cat.



A dragon. I just finished watching Game of Thrones.

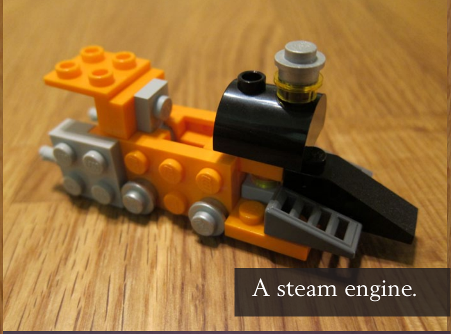


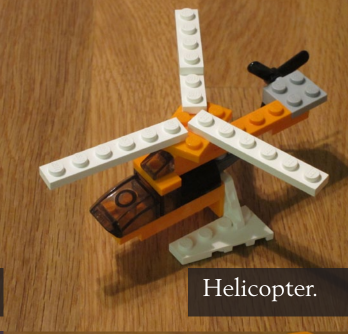
An elephant. I really like this design because it demonstrates symbolic expression, which is a key idea behind the challenge.

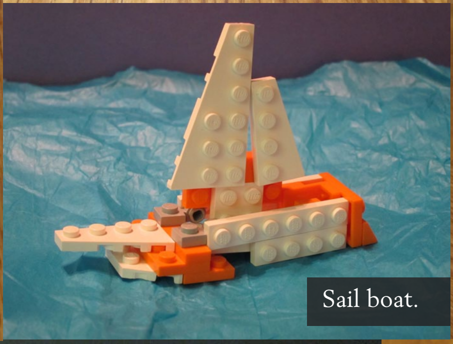


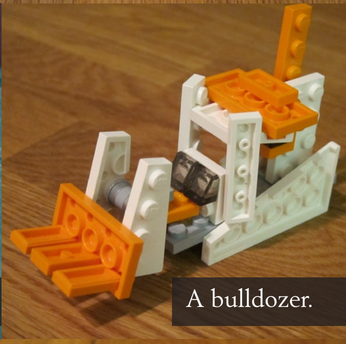
A horse.

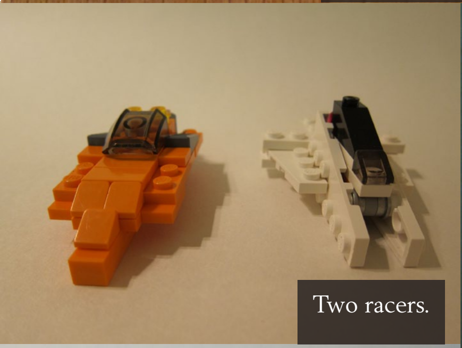
An old biplane. The pieces were begging me to make this one.

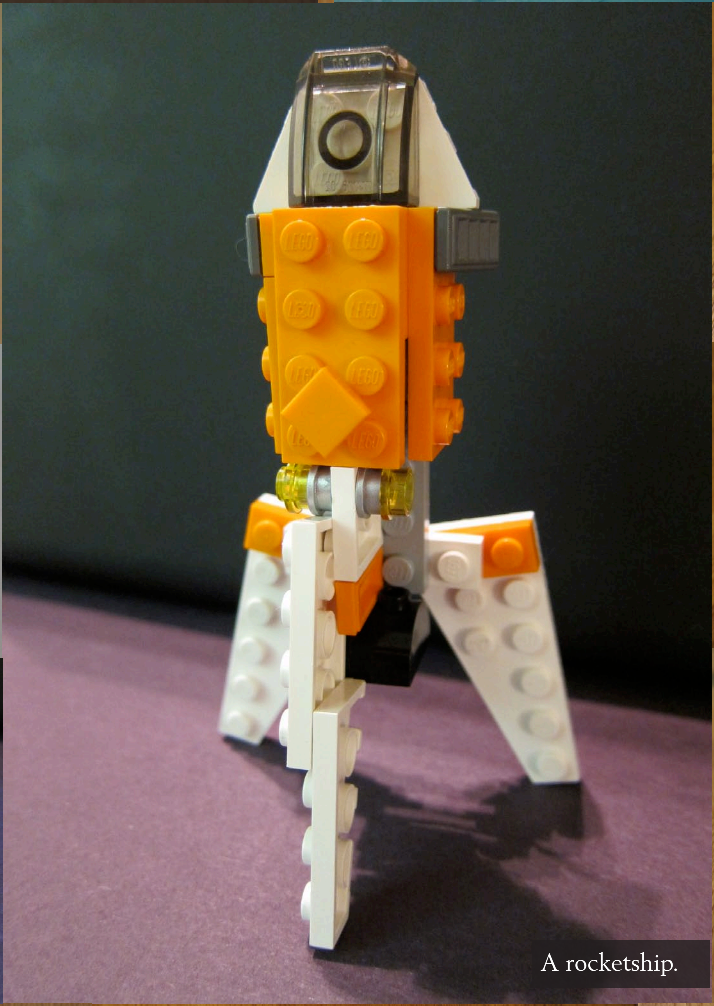

A steam engine.

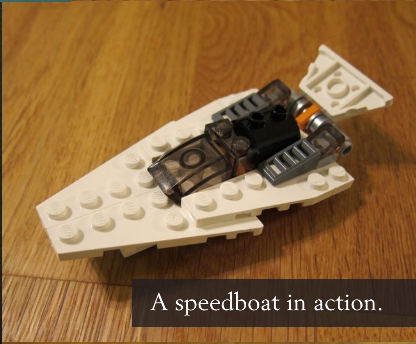

Helicopter.

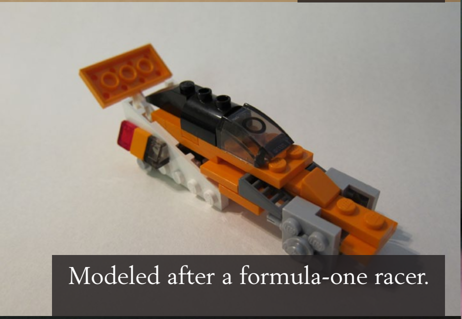

Sail boat.

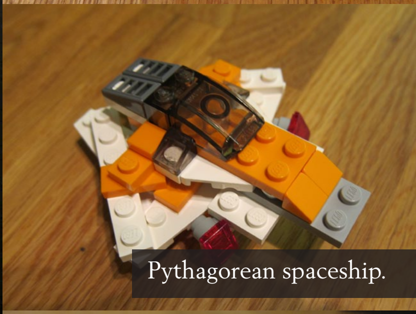

A bulldozer.


Two racers.


A rocketship.


A speedboat in action.

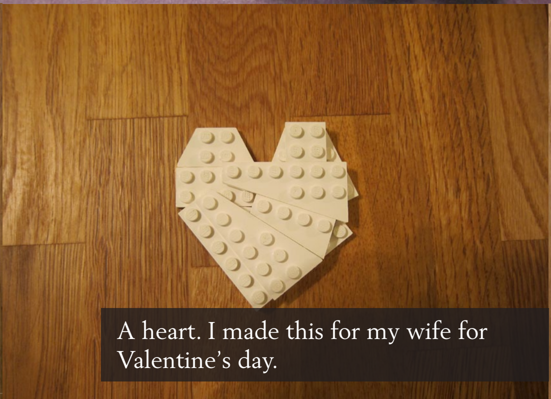

Modeled after a formula-one racer.

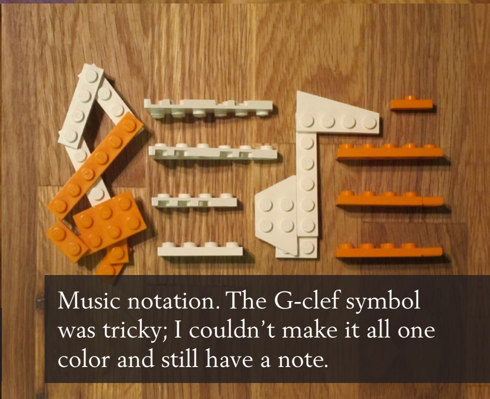

Pythagorean spaceship.

Spaceship.

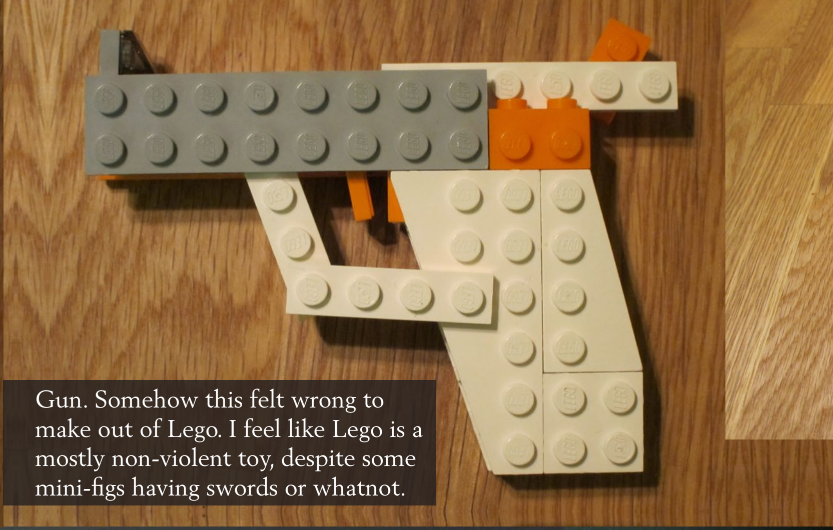
A hovercrafty thing.

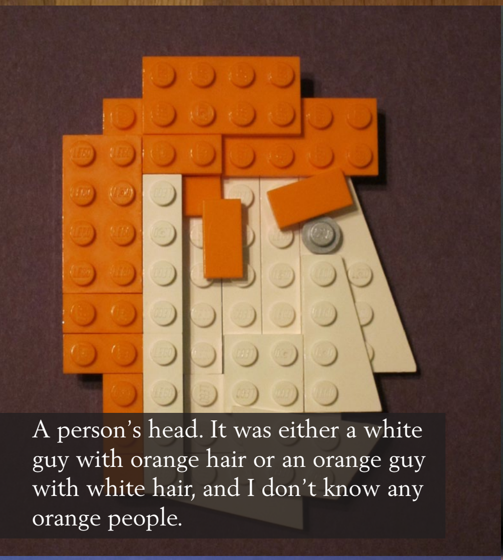
ABC-123.

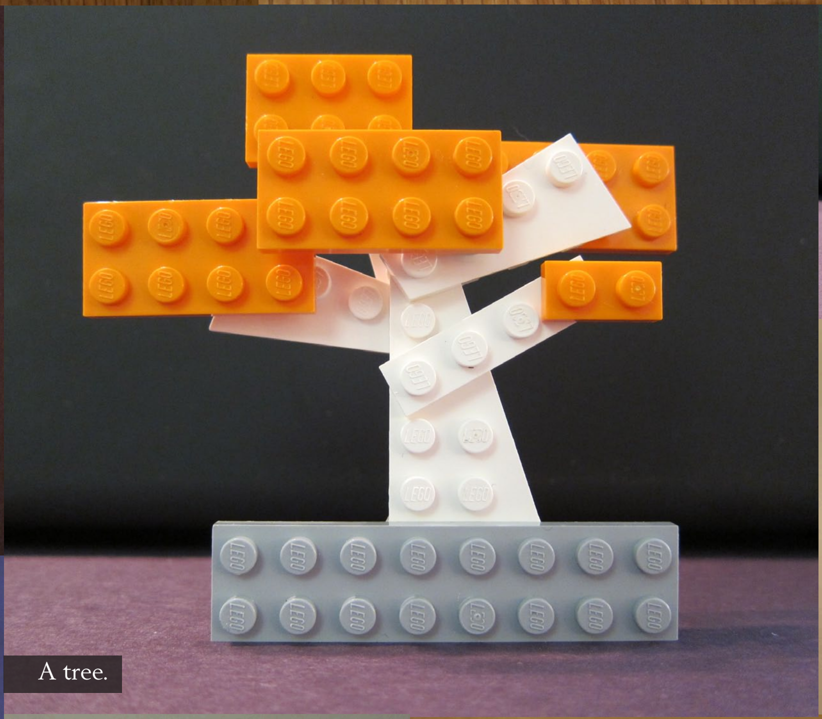

A heart. I made this for my wife for Valentine's day.


Music notation. The G-clef symbol was tricky; I couldn't make it all one color and still have a note.

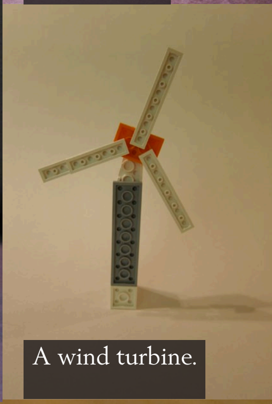
Rock, paper, scissors! I couldn't make a good rock, but I was happy with the scissors.

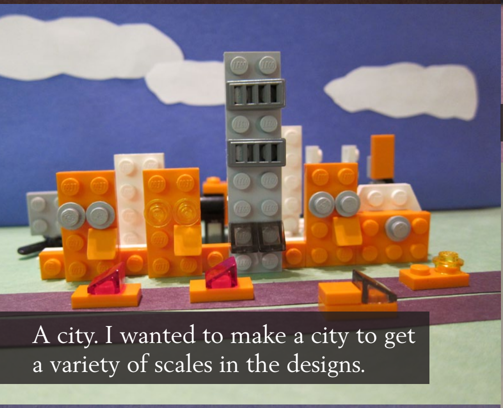

Gun. Somehow this felt wrong to make out of Lego. I feel like Lego is a mostly non-violent toy, despite some mini-figs having swords or whatnot.


A person's head. It was either a white guy with orange hair or an orange guy with white hair, and I don't know any orange people.

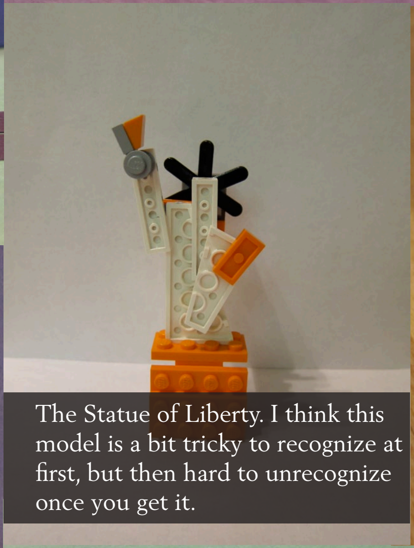

A tree.

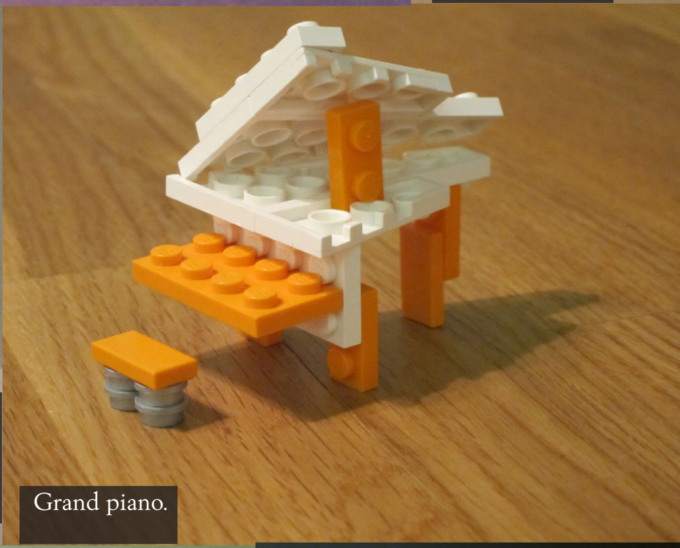
A telescope.


A wind turbine.


A city. I wanted to make a city to get a variety of scales in the designs.

A person. They're wearing orange shorts and an orange t-shirt. It's a good look.


The Statue of Liberty. I think this model is a bit tricky to recognize at first, but then hard to unrecognize once you get it.

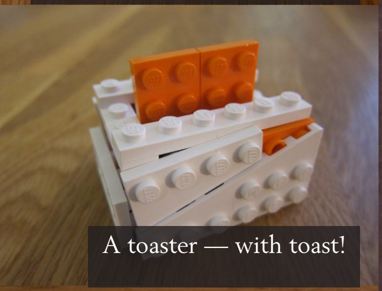

Grand piano.

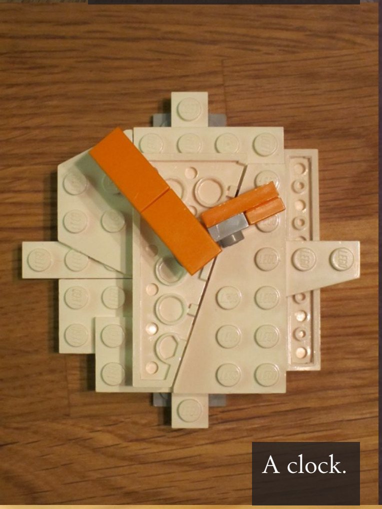
Chess pieces. I tried to get the order of heights pretty close to a real set.

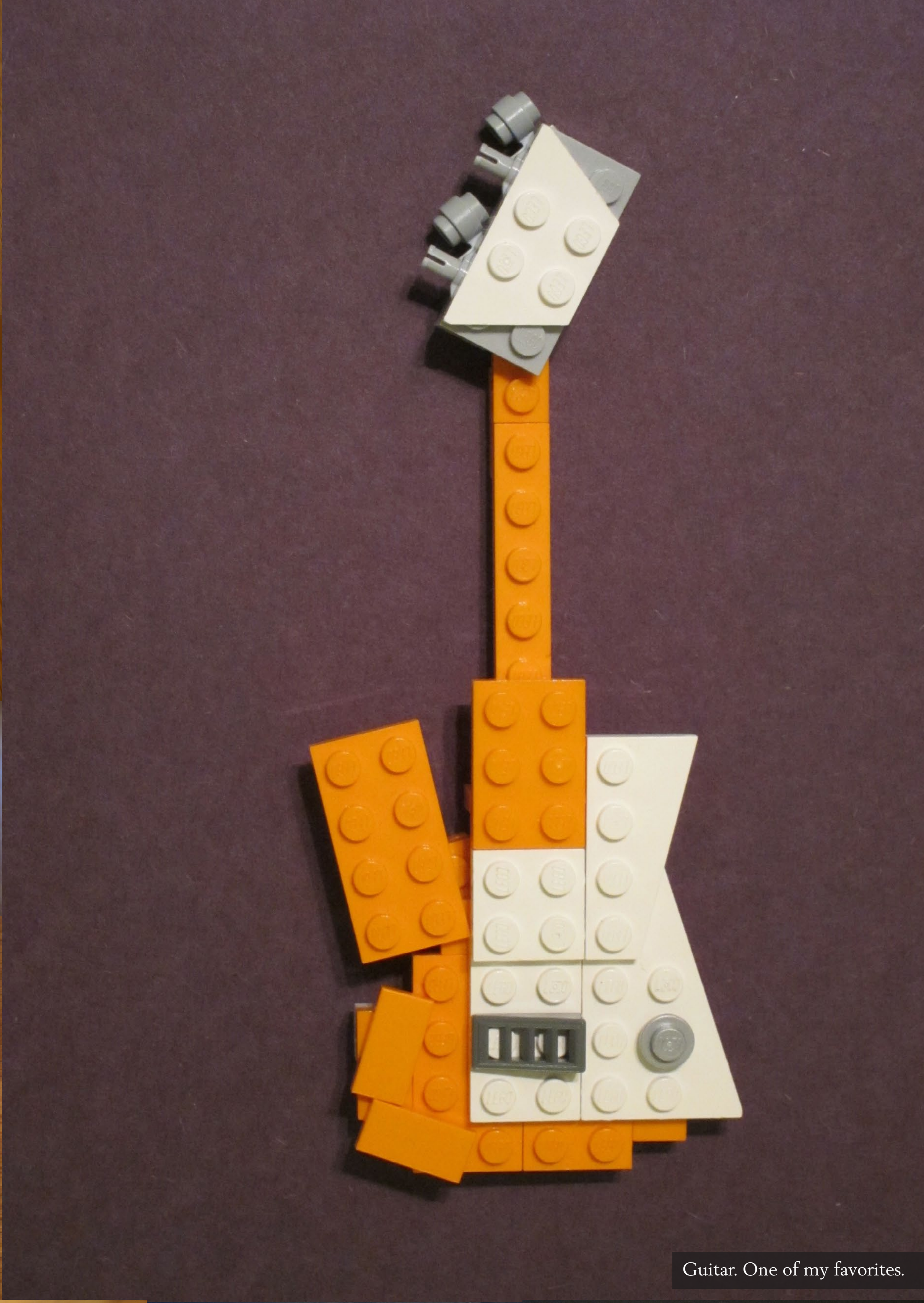
A bird.

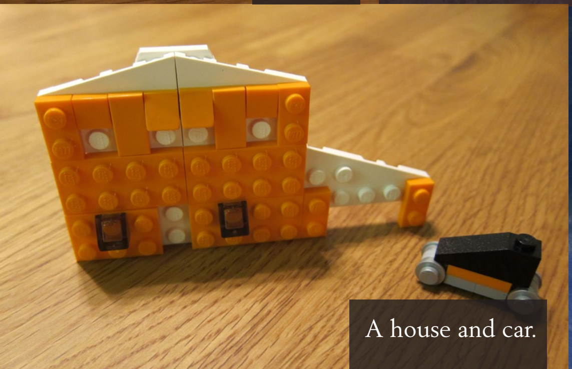
A bug. I have no idea what kind of bug this is.

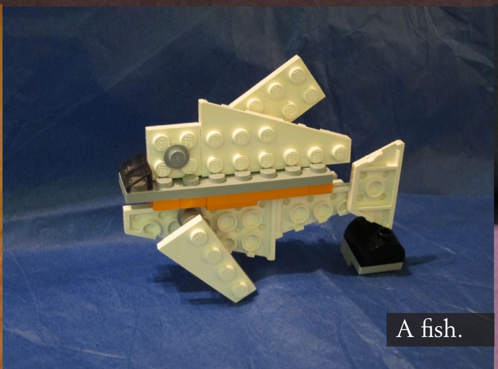
A toilet. No medium should be taken too seriously, especially toys :)

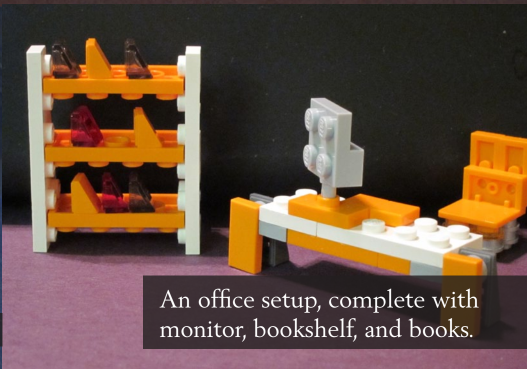

A toaster — with toast!

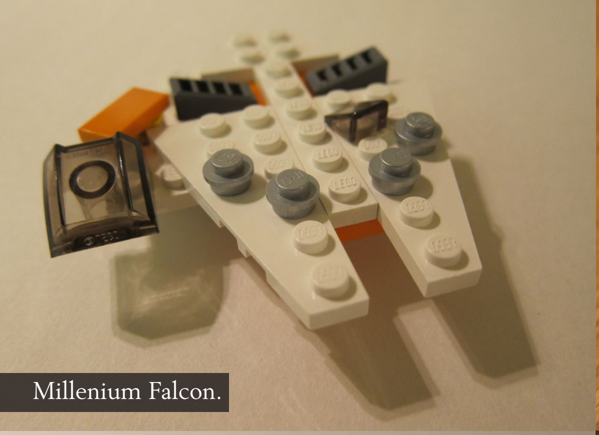

A clock.


Guitar. One of my favorites.


A house and car.

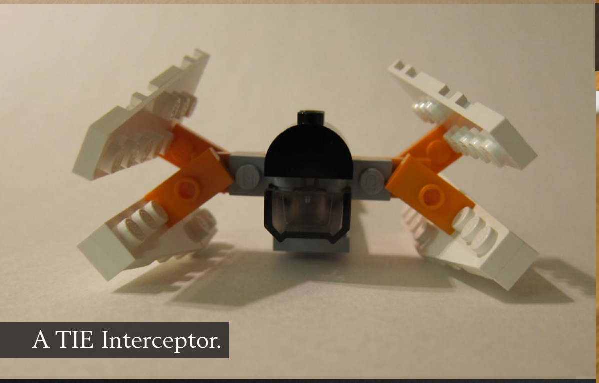

A fish.


An office setup, complete with monitor, bookshelf, and books.

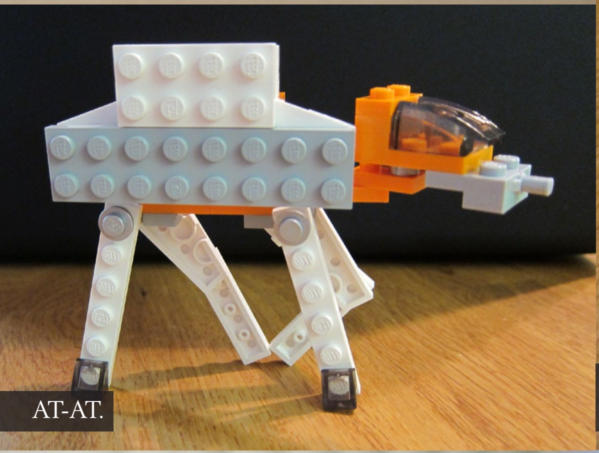

Millenium Falcon.

Snowspeeder. This was the first non-instructed model I built with these pieces.

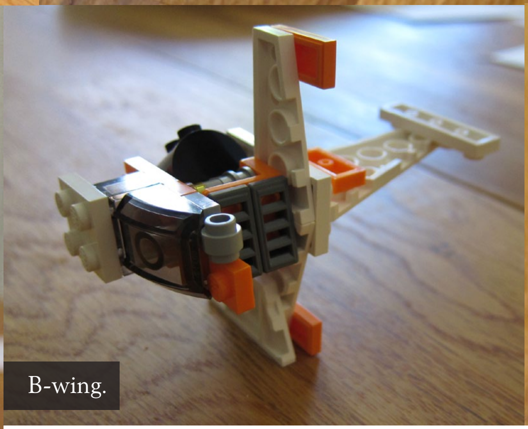
An Imperial Star Destroyer. This must be the largest scale of all the models; the smallest was probably the bug.


A TIE Interceptor.

An X-wing. That little gray nib behind the cockpit is R2-D2.


AT-AT.

AT-ST.


B-wing.

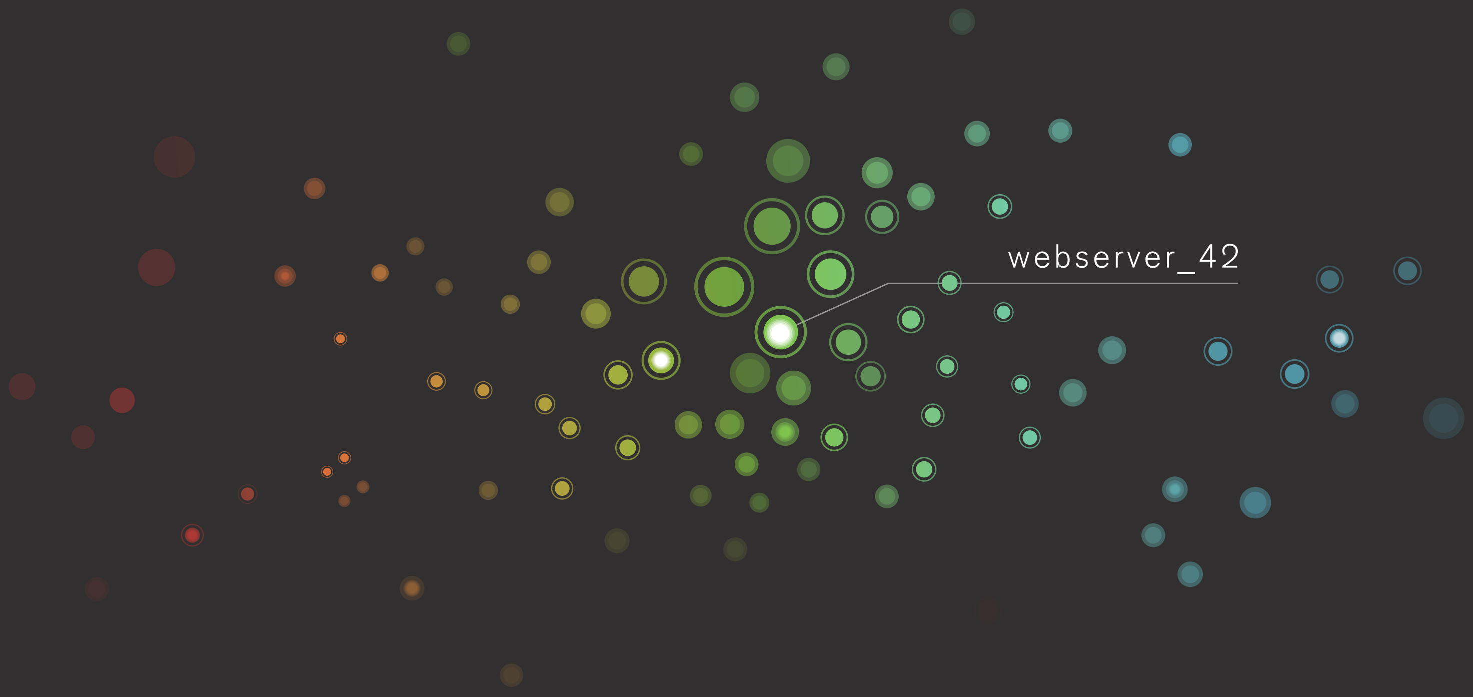
Why not finish with the kitchen sink? It was fun to try getting the faucet positioned well, and have a decent spout.

Tyler Neylon is a programmer who loves math. He is the CTO of Zillabyte, a newly-formed startup doing big data analysis. He used to code for Google, and has a PhD in applied math (machine learning). Tyler posts weekly puzzles at *fridaypuzzl.es* and occasionally post math ideas at *thetangentspace.com*

These are your servers

● ● ●

These are your servers on Cloudkick



Any questions?

cloudkick.com

415.779.5425

support for 8 clouds + dedicated hardware

cloudkick

the best way to manage the cloud

What Should You Do with Your Crappy Little Services Business?

By MARK SUSTER

There's a line of thinking in Silicon Valley that you should build product businesses rather than services businesses. This thinking is largely driven by the venture capital industry (and subsequently Wall Street) who are in search of high margin, highly scalable businesses.

It's nearly impossible to get a services company financed by VCs. You're a small fish.

So pervasive has this thinking become that on several occasions startup companies with profitable and fast-growing tech services businesses have come to me wanting to change their companies to product businesses or to create "spin outs."

A great recent example of this was a successful group of entrepreneurs who had created a company that will do $10–12 million in revenue at their system integration business in 2011 ($5 million in 2010, $2-3 million in 2009). They feel very confident they can hit $18–20 million in 2012.

They have created two internal technology "products" and wanted to figure out how they could turn their services business into a product business that could be financed. This team is talented. They wanted advice. And probably some money.

I gave them advice I don't think they were expecting from a VC,

> *"Don't raise venture capital for this business. Ever. And stop f***ing around trying to create a product company."*

It is advice I give entrepreneurs often as I have written here on why most businesses should never raise VC [hn.my/novc].

Why Shouldn't Most Services Businesses Raise VC?

Well, let's look at this exact situation:

- I don't have access to their actual financial statements, but let me make some reasonable assumptions. It would not be a big stretch to image a well-run service business like this making 15–25% net profit margins. Early in a services business there are usually no profits as the company reinvests in hiring people to grow, but by $20 million in sales the company should at least be pulling in 10% profits (if not more) depending on how much is reinvested.
- So assume that in 2012 the company would do $20 million in sales and $2 million in profits (10%) and 2013 they would do sales of $25 million and $4 million in profit (16% net margin) and then slow growth in 2014 to $30 million and $6 million in profit (20% profit). That is $12 million in profits over 3 years.
- The founders could reinvest this in growth (0% tax, focus on future equity growth) or take the profits of $12 million and divide amongst the founding partners. Assuming there are 3 founders and they own an equal amount (33%) then they've just taken $4 million each in profits and note that this is at a qualified dividend tax rate (currently 15%) versus an income tax rate (35%). True, the 15% rates will likely go up in the future, but I doubt they will approach the income tax percentage level.
- The thing is, even if your services business is a smaller scale than this, you have complete control over the decisions about where to take the business. There is no shame in making a few million dollars in profit and paying yourself dividends while still owning a large percentage (if not all) of your business. It's how things are done across the country outside of Silicon Valley.
- The minute you raise VC you have one option: grow and try to become big. No VC is interested in dividends — they want growth. That's the right answer for VCs. It may be the right answer for you. But it might not.
- Trying to turn a successful services business into a product business is getting the cart before the horse. If you really want to do a product business, then hire a professional manager for your services company, quit that job, and focus 100% on your product company.

Why Build a Services Business in the First Place?

There are at least two types of tech services businesses in my mind:

❶ Service as a bridge to a product business
One of the best ways for young startups to finance their business without any dilution is what I call "customer financing," which is mostly only possible in businesses that target businesses rather than consumers. Customer financing often comes in the form of your company agreeing to build a product with a "sponsor" customer or two and helping them with the rollout/implementation. Often in this strategy you end up giving them the product for free and bill them only services fees. You own the IP you create.

The benefits for the customer are: a mostly custom-built product addressing one of their internal needs, the focus of a very talented young startup focusing on their business need and free product — potentially for life.

The benefits for you are even more clear: you get to build a product raising significantly less external money (if any at all) and therefore no dilution, you get a customer who will help you figure out the real requirements for your business and you have your first real reference client lined up, which should help with future funding and with future sales.

“It is far easier to persuade a business to pay you for your services than buy a totally new product concept.”

If you set out to build this kind of business you just need to be sure you don’t become a permanent consulting business by default. The “customer-financed” type of tech service business is never frowned upon by VCs — unless you’ve been doing it for 2-3 years with no product business to show for it, by which point they assume you’re the second type of services business.

❷ Services for services sake
The type of business that is generally shunned in Silicon Valley is the “pure services” business like consulting, system integration, value-added resellers (VARs), customer support businesses, outsourcing companies, etc. I have already outlined some of the economic reasons these can be good businesses, one of the most important being retaining full control in you business.

But the broader reason that I often suggest to entrepreneurs is that they’re much easier to build than product businesses even though they’ll never become Google, Twitter, or Facebook. Trust me — it is far easier to persuade a business to pay you for your services (a concept they readily understand) than it is to persuade them to buy a totally new product concept and pay for that product.

“How much is that software really worth? Who else is using it? How much did they pay? Wait, I’m only paying “X” for my Salesforce.com licenses — and you want me to pay “Y” for your product? Who are your competitors — how much do they charge?”

I could go on-and-on with all of the sales-blocking messages you will hear when you try to charge for a product. I’ll repeat: everybody understands paying for services. It’s pure irony. At my first company we would have a product sale of $80,000 where the customer would grind us to get the fee down to $70,000 but would readily pay $25,000 extra for “implementation & post-sales support.”

We were building a VC-backed software business, so I had to focus on the product business. But this lesson in business was never lost on me. And some of my former teammates are now building really awesome services businesses in the exact same field and they own 100% of their companies.

Even tech blogs know this. You struggle to get advertisers to pay your CPM rates and get your page clicks up in a business where you become a near commodity to every other website out there. Yet you can run a conference and mint money. If it’s well run, people readily pay for conferences and sponsors readily pay to become platinum, gold, or silver sponsors. Tech blogs can theoretically scale; tech conferences are pure service businesses.

“Just because VCs won’t back the service business doesn’t mean angels won’t.”

But How Do Service Businesses Grow?

I’m not saying that scaling a services business is easy — it’s not. One big challenge is how to grow the company. You end up needing to add staff and take on more risk without knowing what your future demand will be. There are a couple of ways to think about this growth.

1. **Start with a network of independent contractors (1099’s).** When you’re a young company with 3–4 people and you land work that requires 7–8, it can be daunting. You don’t necessarily want to take on the extra employees and risks. I recommend that you establish a network of contractors who want to do work similar to yours but don’t know how to sell projects or to build a company. They’ll be glad for the occasional extra work.

2. **Vendor financing.** When you start to win business — let’s say as an implementation arm for tech/business products or as an ad sales team for large tech/media businesses — you can often get financed in a small way by your vendors who are all to happy to have a bigger ecosystem of implementation houses. They won’t do this before you prove yourself, but once you hit a minimum scale, this is always an option.

3. **Angel financing.** just because VCs won’t back this kind of business doesn’t mean angels won’t. If you can show a few million in sales and the ability to return dividends in the near-term, there are always smart business professionals who will consider financing this. What are their other choices these days — money in a bank at 0.5% interest?

4. **Bank financing.** OK, so this isn’t immediately likely to come from Wells Fargo, but there are tech banks like Silicon Valley Bank or Square1 Bank that are in the business of financing startups. If you can show regular cashflow and are willing to put your profits into their bank you can often fund expansion this way.

Final message on financing: just be careful not to let your fixed costs get too high as a young services business. In a booming tech market like 2011, it’s easy to think your business will always expand. The problem with service businesses is that when the economy turns, revenue and profits take a really big and quick hit. Those companies that have a largely variable cost base and make the tough decisions survive for the next boom.

“Many service businesses mistake their successes in selling services as a competency in selling products.”

Why Shouldn’t Service Businesses Become Product Businesses?

If you build a true “technology services for services sake” business, at some point you’ll likely build technology products as part of your projects where you either own the IP or you own it jointly with your customer or business partner.

This is where many service businesses make mistakes and go pear shaped. They get “product business envy” because they read too much TechCrunch about their product brethren raising money at crazy valuations and getting sold at even crazier ones. So they set out to build a product business within a services company.

A few problems arise. Firstly, they don’t realize how hard product businesses are. They mistake their successes in selling services as a competency in selling products. This is a huge mistake. Secondly, they often ramp up their cost base to accommodate these costs, which, when a down market hits, are more f***ed than those that stay focused. Finally, the focus on the product (envy) means that they take their eye off of their core business, which is services. So the core business suffers.

I saw this first hand. My first career was at Andersen Consulting (one of the largest services businesses in the world). We built a hugely successful global services business, yet we never got over our product envy from watching our tech clients. So we created internal software projects and all of the internal consultants on those projects became blowhards who thought they knew how to create software product businesses.

We stunk at every product we ever created. We had no sense for gathering real customer requirements. We over-spec’d products. We built for our over-intellectual selves. I can’t think of any great software tools ever created internally by Andersen Consulting. We were a great services business. Period.

What Should Services Businesses do with Their Product Businesses?

So back to my advice to the company I recently spoke to about spinning out their tech business or raising VC. My advice wasn't to shut down all product/IP initiatives but rather to be clear on their purpose and how to monetize them.

❶ Products as a service sales machine

My dear friend Franck Meudec in Paris knows this best. He has built some internal technology products to support his services business. They are "loss leaders" for his core business. Instead of going in and trying to hold the line on how much to charge for these products, he can tell customers, "Sure, we'll give you our planning software at cost if you decide to work with us."

His business is booming. These products help him win his core sales. He is not confused about which is the horse and which is the cart. He is building a services business. Instead of owning 1% in options to join a startup tech company, he created his own tech services business. He is the majority owner. Higher risk, higher reward than joining as a junior employee somewhere else.

❷ Products as a key differentiator

Another important reason for having internal IP in your services business is as a key differentiator against other services businesses. If a customer is faced with two equal choices for companies who can implement Salesforce.com, how do they choose one other than from references and price? Imagine if you had built a few modules on top of Salesforce.com that made that product more effective? Even if you didn't charge for these, it would sure increase your sales hit rate.

Tech services business in booming markets are mostly about how fast you can sell, implement, manage quality, hire, and sell some more. In a down market IP can become a huge differentiator.

❸ Products as a gross margin bump

Finally, it should be said that in a services business, often your implementation rate becomes a commodity relative to others in the market. If you can make an extra 10% on each sale by selling your "add on" products that are at 90% gross margins, not only will you increase your win rates but you'll also add valuable profits to your bottom line.

In summary: I'm not advocating that companies are crazy to try and be product companies. In fact, that's all that I fund as a VC. But I don't want the narrow world of venture-backed companies and the trade rags that report on them to dissuade the overwhelming masses of potential entrepreneurs from building meaningful businesses that are both fun and economically rewarding. ■

Mark Suster is a 2x entrepreneur who has gone to the Dark Side of VC. He joined GRP Partners in 2007 as a General Partner after selling his company to Salesforce. He focuses on early-stage technology companies.

Please Stop Asking How to Find a Technical Co-founder

By JASON FREEDMAN

LISTEN GUYS, I'M sorry. But, I just can't do it anymore. I can't keep having this conversation with every non-tech founder. It's just too painful. On you, on me, and on everyone else that you've approached. I was once on the search for a technical co-founder, so I can empathize.

But, seriously, please stop.

Back in the day, I remember going to my favorite startup mentor, Gregg Fairbrothers, and asking him for help finding a technical co-founder. Here's what he said:

> *I can't help you with that, but all the good entrepreneurs seem to figure it out. Hopefully you will, too.*

Man, I still love that answer. That's being a founder. If you have a problem, go figure out a way to solve it. As a professor, Gregg was always teaching me larger lessons instead of just answering my question directly. The cynic might say that he was punting because he didn't have advice to give. However, he helped me on hundreds of other startup questions. I believe he was communicating to me that putting together my team was solely on me. No additional instruction required…or possible. That's why I love going back to him for life advice.

But I digress — back to you. You have a very specific problem which you need solved. You need to find a technical co-founder. This post is my very best effort to help you think through your problem (and by selfish extension, hopefully to never have to answer this question again).

So, here's the really big mental leap that everyone seems to forget:

> *You don't **find** a technical co-founder, you **earn** one.*

And that right there is why I get so bored of this question. It's not like I can really help you "find" a technical co-founder. You have to **earn** a technical co-founder. And until you realize that, no one will want to work with you.

So now I ask you, *what have you done to earn a technical co-founder?*

And don't say that you're the idea guy. Having an idea is one piece, but it's a very, very small piece. In fact, it's so small that it's actually better to earn a technical co-founder without the idea in place so that you guys come up with it together. When neither person has an idea prepackaged with some degree of emotional attachment, it becomes far easier to engage in honest customer development, rapid iteration, and all the other lean processes that will eventually help you find product-market fit. And more importantly, earning a technical co-founder without resting on the merits of your idea forces you to prove yourself in other ways. And that's good for everyone involved.

So, here's the deal. Go out and do all of those things that people always do to find talent. Talk to friends, talk to friends of friends, go to conferences and meetups, etc. Check out the websites [hn.my/co-founder-sites] that are always popping up (though they don't generally attract quality).

When you meet people through all these various ways, realize that every technical person has one of three options:

A) Partner with you.

B) Recommend you to a friend.

C) Forget about you.

Your goal is to not continually hit Outcome C. And the way to do that is to earn their respect. The following is not a recipe you can follow that magically produces a technical co-founder in the end. However, do a bunch of this stuff and the odds that someone recommends you to a friend become much higher. And each of these steps will both make you a better entrepreneur and move your startup along.

How to Earn a Co-Founder

Learn to Code

Stop everything else that you're doing right now for your startup and learn to code. If you take the time to learn enough to build some small project, you'll learn the language of talking to hackers, and you'll earn some respect. Ninety-nine percent of non-technical guys looking for a technical co-founder won't put in the effort. This is your single best way of standing out. You'll learn to naturally see the value of Hacker News and Stack Overflow. You'll learn to appreciate how things work. And hopefully you'll enjoy it, which will allow you to have real conversations with hackers about what they do. Will Miceli wrote the best blog post [hn.my/overrated] I've ever read on exactly this strategy, including awesome links for getting started. Don't know where to start? Zed Shaw will get you started [hn.my/hardway].

Build the Front-End

What's to stop you from building the front end of your site right now? You could get design done with 99 Designs, send it off to PSD2HTML, throw it up on Wordpress, and SHABOW! You've got a website. Of course, there's no backend, no data, none of the special sauce that'll make your concept work...BUT, you'll have proved that you know how to market your idea and build a beautiful product. Hopefully, you'll learn a ton about your product, but at the very least, you can show an interested hacker more than a napkin business plan.

Throw Up a Trial Balloon

I'm sure if you think really hard about it, you can come up with some real things that you can do to test your concept hypothesis. And I'm not talking about more MBA-type research. Hopefully, you already know the importance of customer development [hn.my/custdev]. Find a way to fake your concept so that users don't know it's not actually built yet. Take that front-end you built and funnel interested users into a beta waiting list. Having real users on a waiting list will help you earn a high quality technical co-founder because you'll be pre-empting his biggest fear: that his work will be a waste of his time.

Build a Following

Let's say you're building, for example, an automotive parts marketplace. Go start a blog serving the automotive community. They are your future users anyway, and you're going to have to figure out a way to market to them. What better way than earning them now as readers and later converting them to users? And use Twitter to your advantage. Building up a following north of a 1000 people is hard because that's more than just your friends. Which means you have to say interesting things and share helpful links. It's marketing yourself. It'll prove your intelligence and your marketing abilities to your future co-founder.

Spend Some Money

When a hacker joins an unproven, non-technical entrepreneur, he's risking his most important asset: his time. Yes, you're also risking your time, but you have different risk profiles. While he already knows he can code, neither of you knows whether you'll be able to deliver as the business co-founder. You need to prove that you've got your proverbial skin in the game, too. Go spend some money on offshore coders and get a prototype built. Or offer to pay a salary to your technical partner. My first technical co-founders started as employees. I paid them cash from day one using credit card debt. Over time, I earned their trust, and we became equal co-founders.

If you're a hacker in need of some startup advice, ping me anytime — we'll grab a beer and chat startups. And if you're a business guy that earned a technical co-founder by learning to code, please tell me about it! I'll buy you a beer...you've earned it. ■

Jason is the co-founder of the recently-acquired FlightCaster. Previously, Jason co-founded and worked on several other startups that crashed and burned. Jason continues to stubbornly insist that everything he learned in life came from summer camp. You can find his non-apologetic, self-righteous blog posts at *humbledMBA.com* and on Twitter at *@jasonfreedman*

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

hackernewsletter

Days go by quick. Then an entire week. What did you miss on Hacker News?

Before you see another sunrise subscribe to **Hacker Newsletter**, a weekly email of the best articles from Hacker News curated by hand. Afterwards you won't miss another great article.

Visit http://hackernewsletter.com/hm to subscribe!

Nginx JSON Hacks

By GABRIEL WEINBERG

At DuckDuckGo we use a lot of nginx (an awesome Web server) and a lot of JSON, both for our own API [api.duckduckgo.com] and for processing external APIs [ye.gg/apis]. Here are some hacks we've been using.

Proxy external JSON calls

You can take an external API and run it through your server instead of letting the client call it directly.

```
location ^~ /ext_api2/ {
   proxy_pass http://api.server.com/;
 }
```

That means a request for

```
http://your.server.com/ext_api2/test
```

will turn into

```
http://api.server.com/test
```

Setting up a proxy can yield a number of benefits:

Proxy caching

```
proxy_cache_path  /tmp/nginx_cache
levels=1:2 keys_zone=STATIC:64m
inactive=60m max_size=128m;
proxy_cache STATIC;
proxy_cache_valid 200 204 302 1d;
```

Now it won't hit the external API if the same request is called by multiple clients. If it is a pay API, this could save you money, and it could also just speed up the responsiveness of your site.

Proxy timeouts

```
proxy_connect_timeout 5;
proxy_read_timeout 5;
proxy_send_timeout 5;
```

You can set the timeouts per proxy (or globally), thus controlling how long the client will wait for each request. With timeouts in place you can ensure the page doesn't hang on something, eventually loads, and gracefully degrades the way you want it to — even for components you don't control.

Strip headers

```
proxy_hide_header Set-Cookie;
```

Some external APIs like to do things to clients (like set cookies). You can protect your users from that by stripping them (or other headers).

Reset headers

```
proxy_set_header Referer http://duck-
duckgo.com/;
```

Similarly, you can reset your headers. This can protect privacy by zeroing out search terms (in the case of the Referrer), but you can also set custom headers.

Hide private API keys

Many APIs require use of a key, which you generally don't want to expose client-side. You can still allow for client-side calls by proxying them and then having nginx add the key.

```
location ^~ /ext_api5/ {
  rewrite ^/ext_api5/(.*) /api/check/$1/key/
e95fad09aa5091b7734d1a268b53cef5  break;
  proxy_pass http://api.server.com/;
}
```

Now a request for

```
http://your.server.com/ext_api5/test
```

will turn into

```
http://api.server.com/api/check/test/key/
e95fad09aa5091b7734d1a268b53cef5
```

Turn JSON into JSONP

JSONP is a slight modification of JSON where the object is wrapped in a callback function usually specified by you. For example, say you grab a JSON object from somewhere that looks like this:

```
{"Name": "Cheeso", "Id" : 1823, "Rank": 7}
```

With JSONP you specify a callback function like "parseResponse" and then it looks like this:

```
parseResponse({"Name": "Cheeso", "Id" :
1823, "Rank": 7})
```

This is useful for two reasons. First, the function will be called automatically when it is done loading. Second, it allows you to get around cross-domain errors.

If the above API was yours it's easy to call within a client-side script. But if it isn't yours, i.e. on another domain, and you try to call it, you'll often get lots of cross-domain errors. The way around this is to use JSONP. Then you can do something like this:

```
<script type="text/javascript"
src="http://other.server.com/api/?q=param
&callback=parseResponse"></script>
```

You could also do that via JS, e.g.

```
function add_script(url) {
    var script,scripts;
    script = document.
createElement('script');
    script.type='text/javascript';
    script.async = true;
    script.src = url;

    scripts = document.
getElementsByTagName('script')[0];
    scripts.parentNode.
insertBefore(script, scripts);
}
```

```
add_script('http://other.server.com/api/?
q=param&callback=parseResponse');
```

Now here's the problem. Most external APIs don't have JSONP capability.

No bother, with nginx you can turn JSON into JSONP.

```
location ^~ /ext_api3/ {
  echo_before_body 'parseResponse(;
  proxy_pass http://api.external.com/;
  echo_after_body ');';
}
```

This uses the HTTPEchoModule to wrap the JSON response in the callback for the external API.

Custom logs

The HTTPLogModule allows you to specify log formats within location blocks, which means you can write your API logs to a separate file. It also means if you proxy via a location block as in the examples above, you could give each proxy their own access and error logs with different parameters, e.g. error log level. ■

Gabriel Weinberg is the founder of DuckDuckGo, a search engine. He is also an active angel investor, based out of Valley Forge, PA.

Replication, Atomicity and Order in Distributed Systems

By ALEX FEINBERG

DISTRIBUTED SYSTEMS ARE an increasingly important topic in Computer Science. The difficulty and immediate applicability of this topic is what makes distributed systems rewarding to study and build.

The goal of this article is to help the reader develop a basic toolkit they could use to reason about distributed systems. The toolkit should help the reader see the well-known patterns in the specific problems they're solving, to identify the cases where others have already solved the problems they're facing, and to understand the cases where solving 100% of the problem may not be worth the effort.

Leaving a Newtonian Universe

For the most part, a single machine is a Newtonian universe: that is, we have a single frame of reference. As a result, we can impose a total Happened-Before order on events, i.e., we can always tell that one event happened before another event. Communication can happen over shared memory, access to which can be synchronized through locks and memory barriers.[1]

When we move to a client and server architecture, message passing architecture is required. In the case of a single server (with one or more clients), we can still maintain an illusion of a Newtonian universe: TCP (the transport layer used by popular application protocols) gives a guarantee that packets will be delivered to the server in the order sent by the client. As we'll later see, this guarantee can be used as a powerful primitive upon which more complex guarantees can be built.

However, there are reasons why we no longer want to run an application on a single server: in recent times it has become consensus that reliability, availability, and scalability are best obtained using multiple machines. Mission critical applications must at least maintain reliability and availability; in the case of consumer (and even many enterprise) web applications, with success often come scalability challenges. Thus, it's inevitable that we leave Newton's universe and enter Einstein's.[2]

[1] This is not to belittle the fascinating challenges of building parallel shared memory systems: the topic is very well covered outside of this post. I highly recommend "The Art of Multiprocessor Programming" (by Maurice Herlihy) and "Java Concurrency In Practice" (Goetz, Lea, et al) to those interested in shared memory concurrency.

[2] The comparison with theory of relativity is not original: Leslie Lamport and Pat Helland-have used this comparison. Several concepts in distributed systems such as Vector Clocks and Lamport Timestamps are explicitly inspired by relativity.

Intuitive Formulation of the Problem

Suppose we have a group of (physical or logical) nodes: perhaps replicas of a partition (aka a shard) of a shared nothing database, a group of workstations collaborating on a document or a set of servers running a stateful business application for one specific customer. Another group of nodes (which may or may not overlap with the first group of nodes) is sending messages to the first group. In the case of a collaborative editor, a sample message could be "insert this line into paragraph three of the document". Naturally, we would like these messages delivered to all available machines in the first group.

Question is, how do we ensure that, after the messages are delivered to all machines, the machines remain in the same state? In the case of our collaborative editor application, suppose Bob is watching Alice type over the shoulder and sees her type "The" and types "quick brown fox" after: we'd like all instances of the collaborative editor to say "The quick brown fox" and not "quick brown fox The"; nor do we want messages delivered multiple times, e.g., not "The The quick brown fox" and especially not "The quick brown fox The"!

We'd like (or, in many cases, require) that if one of the servers goes down, the accumulated state is not lost (reliability). We'd also like to be able to view the state in the case of server failures (read availability) as well as continue sending messages (write availability). When a node fails, we'd also like to be able to add a new node to take its place (restoring its state from other replicas). Ideally, we'd like the later process to be as dynamic as possible.

All of this should have reasonable performance guarantees. In the case of the collaborative editor, we'd like characters to appear on the screen seemingly immediately after they are typed; in the case of the shared nothing database, we'd like to reason about performance not too differently from how we reason about single node database performance, i.e., determined (in terms of both throughput and latency) primarily by the CPU, memory, disks and ethernet. In many cases we'd like our distributed systems to even perform better than analogous single node systems (by allowing operations to be spread across multiple nodes), especially under high load.

Problem is, however, that these goals are often contradictory.

State Machines, Atomic Multicast, and Consensus

An approach commonly used to implement this sort of behavior is state machine replication. This was first proposed by Leslie Lamport (also known as the author of LaTeX), in the paper "Time, Clocks and the Ordering of Events in a Distributed System". The idea is that if we model each node in a distributed system as a state machine, and send the same input (messages) in the same order to each state machine, we will end up in the same final state.

This leads to our next question: how do we ensure that the same messages are sent to each machine, in the same order? This problem is known as atomic broadcast or more generally atomic multicast. We should take special care to distinguish this from the IP multicast protocol, which makes no guarantees about order or reliability of messages: UDP, rather than TCP is layered on top of it.

A better way to view atomic multicast is as a special case of the publish subscribe pattern (used by message queuing systems such as ActiveMQ, RabbitMQ, Kafka and Virtual Synchrony based systems such as JGroups and Spread[3]).

A generalization of this problem is the distributed transaction problem: how do we ensure that either all the nodes execute the exact same transaction (executing all operations in the same order), or none do?

Traditionally two phase commit (2PC) algorithm has been used for distributed transactions. The problem with two phase commit is that it isn't fault tolerant: if the coordinator node fails, the process is blocked until the coordinator is repaired (Consensus on Transaction Commit).

Consensus algorithms solve the problem of how multiples nodes could arrive at a commonly accepted value in the process of failures. We can use consensus algorithm to build fault-tolerant distributed commit protocols by (this is somewhat of an over-simplification) having nodes "decide" whether or not a transaction has been committed or aborted.

[3] Virtual synchrony (making asynchronous systems appear as synchronous) is itself a research topic that is closely related to and at times complemented by consensus work. Ken Birman's group at Cornell has done a great deal of work on it. Unfortunately, it was difficult to work much of this fascinating research into a high level blog post.

Theoretic Impossibility, Practical Possibility

Problem is that it's impossible to construct a fault-tolerant consensus algorithm that will terminate in a guaranteed time-bound asynchronous system lacking a common clock: this is known (after the Fisher, Lynch, Patterson) as the FLP impossibility result. Eric Brewer's CAP theorem (a well covered topic) can be argued to be an elegant and intuitive re-statement of the FLP.

In practice, however, consensus algorithms can be constructed with reasonable liveness properties. It does, however, imply that consensus should be limited in its applications.

One thing to note is that consensus protocols can typically handle simple or clean failures (failures of minority of nodes), at the cost of greater latency: handling more complex (split brain scenarios), where a quorum can't be reached is more difficult.

Paxos and ZAB (Chubby and ZooKeeper)

The Paxos Consensus and Commit protocols are well known and are seeing greater production use. A detailed discussion of these algorithms is outside the scope of this post, but it should be mentioned that practical Paxos implementations have somewhat modified the algorithms to allow for greater liveness and performance.

Google's Chubby service is a practical example of a Paxos based system. Chubby provides a file system-like interface and is meant to be used for locks, leases, and leader elections. One example of use of Chubby (that will be discussed in further detail in the next post) is assigning mastership of partitions in a distributed database to individual nodes.

Apache ZooKeeper is another practical example of a system built on a Paxos-like distributed commit protocol. In this case, the consensus problem is slightly modified: rather than assume a purely asynchronous network, the TCP ordering guarantee is taken advantage of. Like Chubby, ZooKeeper exposes a file-system like API and is frequently used for leader election, cluster membership services, service discovery and assigning ownership to partitions in shared nothing stateful distributed systems.

Limitations of Total Transactional Replication

A question arises: why is transactional replication only used for applications such as cluster membership, leader elections, and lock managers? Why aren't these algorithms used for building distributed applications, e.g., databases themselves? Wouldn't we all like a fully transactional, fault-tolerant, multi-master distributed database? Wouldn't we like message queues that promise to deliver exactly the same messages, to exactly the same nodes, in exactly the same order, delivering each message exactly once at the exact same time?

The above mentioned FLP impossibility result provides one limitation of these systems: many practical systems require tight latency guarantees even in the light of machine and network failures. "The Dangers of Replication and a Solution" also discusses scalability issues such as increases in network traffic, potential deadlocks in what the authors called "anywhere-anytime-anyway transactional replication."

In the case of Chubby and ZooKeeper, this is less of an issue: in a well-designed distributed system, cluster membership and partition ownership changes are less frequent than updates themselves (much lower throughput, less of a scalability challenge) and are less sensitive to latency. Finally, by limiting our interaction with consensus-based systems, we are able to limit the impact of scenarios of where consensus can't be reached due to machine, software, or network failures. ■

Alex Feinberg is a Senior Software at LinkedIn, working on Project Voldemort, a reliable, distributed key-value store. He's interested in distributed systems, systems programming and functional programming languages.

Ways to Get Started in Erlang and Programming

By JOE ARMSTRONG

I'M OLD SCHOOL: you don't need any fancy tools. Just a text editor and an erlang shell.

1. Open your text editor:

2. Type in the following program.

```
-module(hello).
-compile(export_all).
start() -> "hello world".
```

3. Store it in a file called `hello.erl`

4. Start an erlang shell. It will say ">"

5. Type in two commands:

```
> c(hello).
> hello:start().
```

The first command compiles the program. The second evaluates the command `hello:start()`

That's all it takes — typing three lines of code into a file with a text editor, then typing two lines into the shell.

That's all it takes. Ninety-five percent of all the fun can be achieved with a simple text editor and the erlang shell. That's how most of the erlang system was implemented.

The erlang shell can be installed in zillions of ways: compile the sources or apt-get install it (or whatever).

Forget about git/IDEs/rebar etc.

Use this approach for all languages.

IDEs and build tools are the single biggest obstacle I know of to getting started.

Me, I use

- a shell
- makefiles
- emacs

for all known programming languages under the sun.

Ninety-eight percent of all the fun can be had with the compiler alone — all the rest is hype.

“Your brain is a zillion times better than the best IDE. Programs form in your brain, not in an IDE.”

Forget about the tools

Tools like rebar, etc., are under for automating something, but if you don’t know what it is that you are automating, and if the tool doesn’t work, you will just end up incredibly confused.

Then buy a decent book and type in the programs by hand.

One at a time, thinking as you go.

After 30 years you will get the hang of this and be a good programmer.

Tools are no substitute for typing in small programs and understanding exactly how they work. This is true for all programming language. Programming is an art form, there is no easy way.

Like playing the violin — is there an easier way to learn violin other than by practicing for thousands of hours? I think not.

Start with one-liners in the shell. Start the shell:

type

```
> A = 1
```

then

```
> A = 2
```

Ask what happens and why.

There is no quick way to learn programming — no tool will help.

Your brain is a zillion times better than the best IDE. Programs form in your brain, not in an IDE.

But then, I’m old school.

Have fun — if it’s not fun it is pointless. Don’t fight the tools; all you need is a text editor and the erlang shell to start with. ■

Joe Armstrong invented Erlang in 1986. He has a PhD in Computer Science. And still spends most of free time trying to write beautiful programs.

Top Programming Fonts

By DAN BENJAMIN

I'M A TYPEFACE geek, and when it comes to selecting a font I'll stare at all day, I tend to be pretty picky. Recently, when I discovered that a friend was using a sub par typeface (too horrible to name here) for his Terminal and coding windows, my jaw dropped, my heart sank a little, and I knew it was due time for me to compose this article.

What follows is a round-up of the top 10 readily-available monospace fonts. Many of these fonts are bundled along with modern operating systems, but most are free for download on the web. A few, notably Consolas, are part of commercial software.

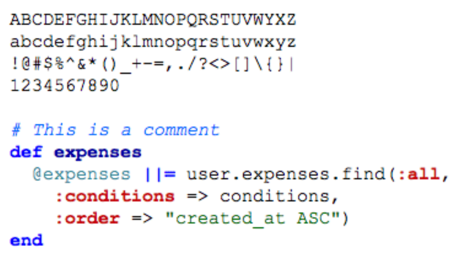
A Note About Anti-Aliasing

In the past, we've had to decide between tiny monospace fonts or jagged edges. But today, modern operating systems do a great job of anti-aliasing, making monospace fonts look great at any size. It's not 1990 anymore. Give your tired eyes a break and bump up that font size.

If you have any doubt that anti-aliased fonts are apropos for code, note that even the venerable BBEdit — which for years has shipped with un-aliased Monaco 9 set as the default — has made the jump. The app now ships with a specially licensed version of the Consolas font from Ascender, bumped up in size, and with anti-aliasing on by default. Panic includes a special anti-aliased font (Panic Sans, which is actually just a version of Deja Vu Sans Mono) with its popular Coda application.

Unless otherwise noted, I've used a larger size font, 15-point in fact, for the examples here to illustrate their legibility at larger sizes and with anti-aliasing turned on.

⑩ Courier

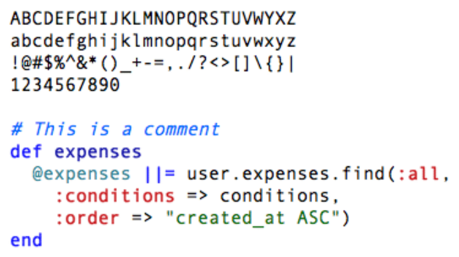


```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Courier New

All systems ship with a version of Courier (sometimes Courier New), and unfortunately, many have it set as the default font for terminal and editor windows. It does the job, but it's a bit dull and boring, lacking style and class. I don't recommend this font if you have any other choice — and fortunately, you do. If you use this font, please bump the size and turn on anti-aliasing.

⑨ Andale Mono

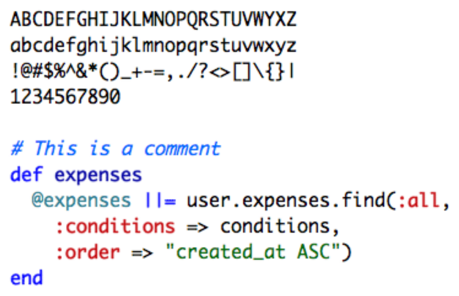


```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Andale Mono

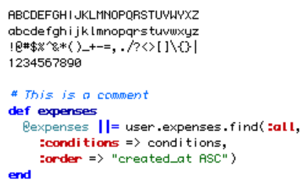
A bit better than the Courier family, Andale Mono is still relegated to the "default font" category as it ships with some systems, and you wouldn't want to download or use it if it wasn't already there. The character-spacing is a bit too clumsy and the letters are a bit too wide for my tastes.

⑧ Monaco



```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Monaco



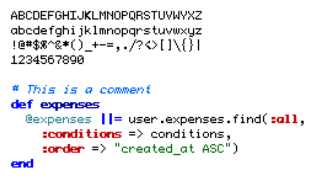
```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Monaco 9-point, without anti-aliasing

Monaco is the default monospace font on the Mac and has been since its inclusion in System 6. It's a solid, workhorse font that really shines at smaller font sizes with anti-aliasing turned off. I loved this typeface back when my eyes could tolerate staring at a 9-point font for hours, but those days are behind me. This font looks great at 9 or 10-points, and doesn't look too shabby anti-aliased at higher sizes.

As far as I know, you can only get Monaco as a part of Mac OS, but there are alternatives, so keep reading.

7 Profont



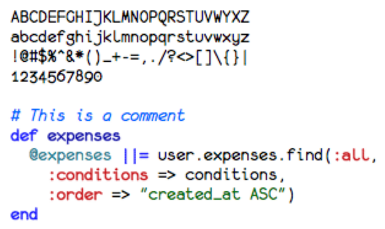
```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Profont 9-point, without anti-aliasing

Profont is a Monaco-like bitmap font available for Mac, Windows, and Linux (there's also a modified version for Mac OS X called ProFontX by a different author). They're best at smaller sizes, and make a great alternative to Monaco if you're on a non-Mac platform and want really tiny fonts and the eyestrain that goes along with them.

Profont (and ProFontX) is intended for use at 9-points with anti-aliasing turned off.

6 Monofur

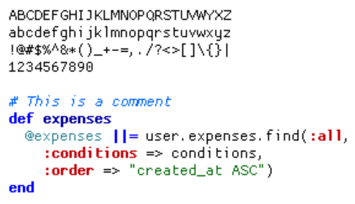


```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Monofur

Monofur is a unique monospace font that looks great anti-aliased at all sizes. It's a fun font with a distinct look that is vaguely reminiscent of Sun's OPEN LOOK window manager, which ran Solaris (aka SunOS) systems back in the late 80's. If you're looking for something a bit different, try this font, but make sure you have anti-aliasing turned on, even at small sizes.

5 Proggy

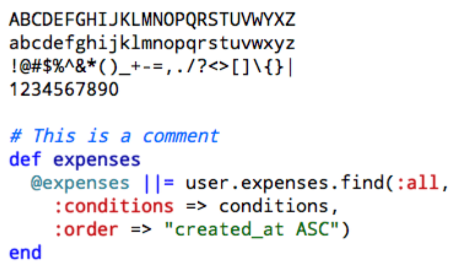


```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Proggy Clean at 15-point , without anti-aliasing

Proggy is a clean monospace font that seems to be favored by Windows users, although it works fine on a Mac. It's a clean font intended to be used only at smaller points, and without anti-aliasing.

4 Droid Sans Mono



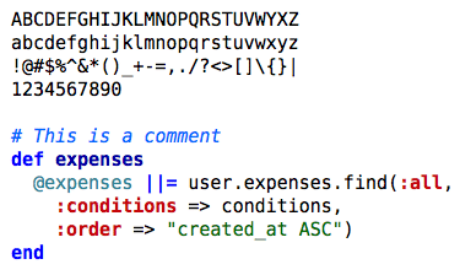
```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Droid Sans Mono

The Droid font family is a nice font family designed for use on the small screens of mobile handsets, like Android, and licensed under the Apache license.

Droid Sans Mono makes for a great programming font. It's got a bit of flair, and stands out among the other monospace fonts I've listed, and its only real flaw is the lack of a slashed zero.

❸ Deja Vu Sans Mono



```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

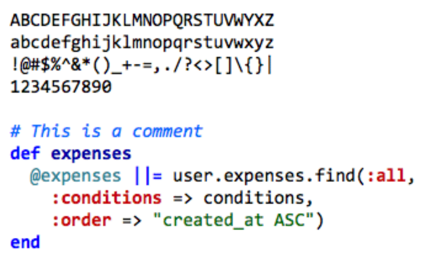
Deja Vu Sans Mono

The Deja Vu family of fonts are one of my favorite free font families, based on the excellent Vera Font family. The Deja Vu fonts have been updated with a wider range of characters while maintaining a similar look and feel to that of Vera.

This was my go-to font family for many years. It looks great at any size with anti-aliasing turned on.

Panic ships a font with it's Coda application called "Panic Sans" which is based on this font. Gruber says via email that when he compared Panic Sans against Vera, he noted that "Panic had noticeably crisper punctuation chars" and that it seemed like they had improved the hinting on some characters as well.

❷ Consolas



```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

Consolas

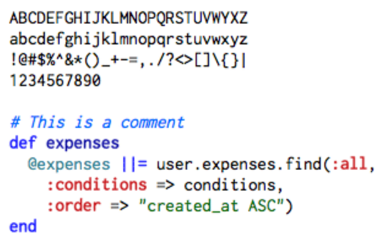
Consolas suddenly appeared on my Mac after I installed Microsoft Office, along with a handful of other new fonts from Microsoft.

This font was designed by Luc(as) de Groot for Microsoft's ClearType font family (there's a nice write-up with samples of each of the new Microsoft fonts here). Consolas is a commercial font, but is bundled with many Microsoft products, so there's a good chance you might already have it on your system.

You'll absolutely want to have anti-aliasing turned on if you're using Consolas, because it'll look terrible without it.

Too bad it's not free … if it was, it would be #1 on this list.

❶ Inconsolata



```
ABCDEFGHIJKLMNOPQRSTUVWYXZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+-=,./?<>[]\{}|
1234567890

# This is a comment
def expenses
  @expenses ||= user.expenses.find(:all,
    :conditions => conditions,
    :order => "created_at ASC")
end
```

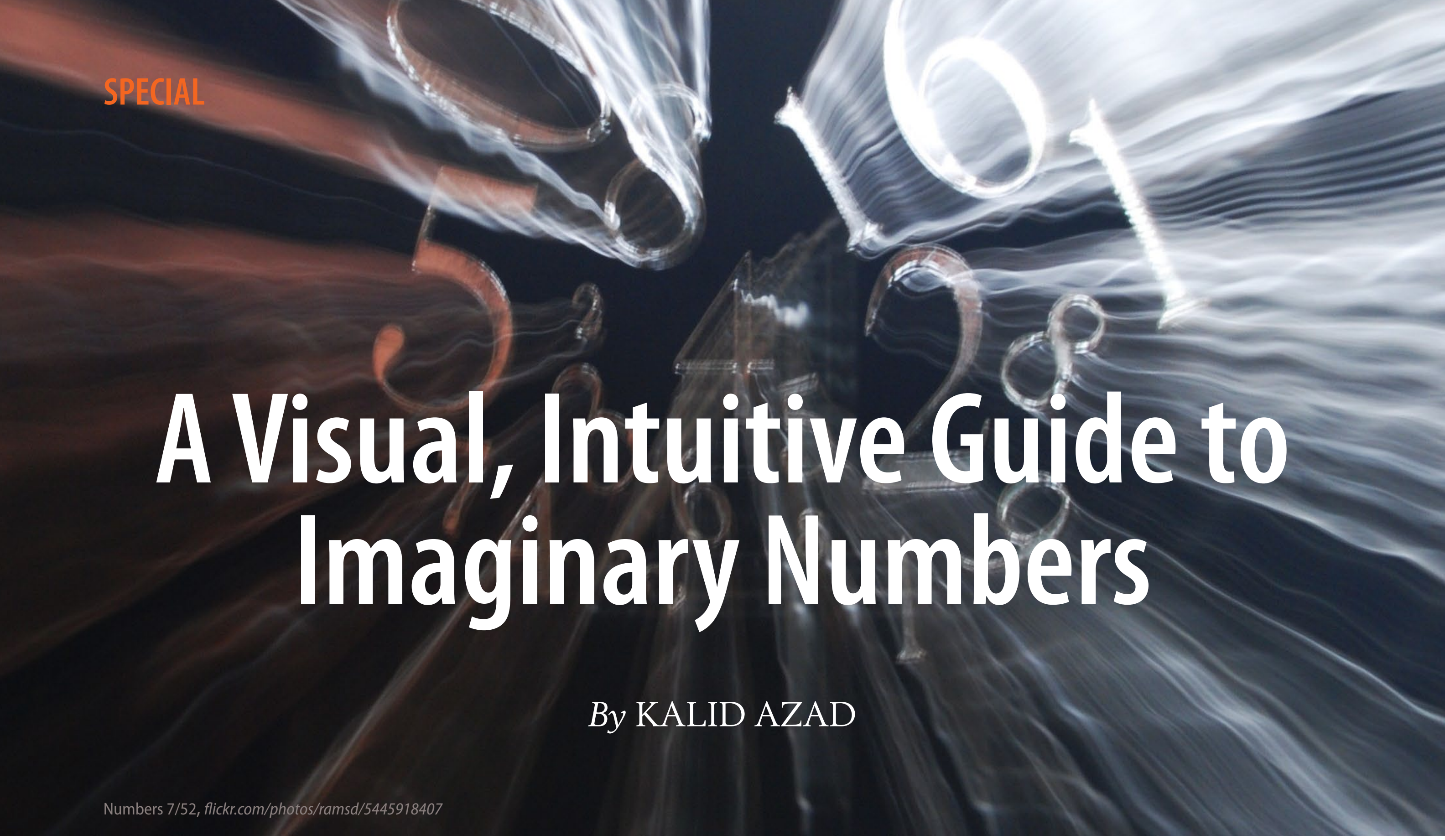
Inconsolata

Inconsolata is my favorite monospaced font, and it's free. Shortly after discovering it, it quickly supplanted Deja Vu Sans Mono as my go-to programming font. I use it everywhere, from Terminal windows to code editors. It has a certain sublime style that's unique without being over the top, and it looks fantastic at both large and small sizes. I use this font when I show code samples in a presentation, and it's the font we use in Terminal and TextMate windows when filming PeepCode screencasts.

Inconsolata is designed to be used with anti-aliasing enabled, but it's surprisingly legible even at very small sizes. A big thanks to Raph Levien for creating this font, and for making it free. ■

Dan is the founder of 5by5 Studios and author of Hivelogic. He also created the Email Address Enkoder, co-founded Cork'd and founded Playgrounder.

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



A Visual, Intuitive Guide to Imaginary Numbers

By KALID AZAD

Numbers 7/52, *flickr.com/photos/ramsd/5445918407*

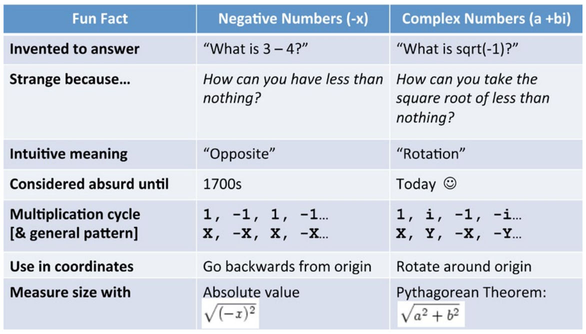
IMAGINARY NUMBERS ALWAYS confused me. Like understanding e [hn.my/e], most explanations fell into one of two categories:

- It's a mathematical abstraction, and the equations work out. Deal with it.
- It's used in advanced physics, trust us. Just wait until college.

Gee, what a great way to encourage math in kids! Today we'll assault this topic with our favorite tools:

- Focusing on relationships, not mechanical formulas.
- Seeing complex numbers as an upgrade to our number system, just like zero, decimals, and negatives were.
- Using visual diagrams, not just text, to understand the idea.

And our secret weapon: **learning by analogy.** We'll approach imaginary numbers by observing its ancestor, the negatives. Here's your guidebook:



Fun Fact	Negative Numbers (-x)	Complex Numbers (a +bi)
Invented to answer	"What is 3 – 4?"	"What is sqrt(-1)?"
Strange because…	*How can you have less than nothing?*	*How can you take the square root of less than nothing?*
Intuitive meaning	"Opposite"	"Rotation"
Considered absurd until	1700s	Today ☺

Multiplication cycle [& general pattern]	`1, -1, 1, -1…`
	`X, -X, X, -X…`
		`1, i, -1, -i…`
		`X, Y, -X, -Y…`

Use in coordinates	Go backwards from origin	Rotate around origin
Measure size with	Absolute value $\sqrt{(-x)^2}$	Pythagorean Theorem: $\sqrt{a^2+b^2}$

It doesn't make sense yet, but hang in there. By the end we'll hunt down i and put it in a headlock, instead of the reverse.

Really Understanding Negative Numbers

Negative numbers aren't easy. Imagine you're a European mathematician in the 1700s. You have 3 and 4, and know you can write 4 – 3 = 1. Simple.

But what about 3-4? What, exactly, does that mean? How can you take 4 cows from 3? How could you have less than nothing?

Negatives were considered absurd, something that "darkened the very whole doctrines of the equations" (Francis Maseres, 1759). Yet today, it'd be absurd to think negatives aren't logical or useful. Try asking your teacher whether negatives corrupt the very foundations of math.

What happened? We invented a *theoretical number that had useful properties*. Negatives aren't something we can touch or hold, but they describe certain relationships well (like debt). It was a useful fiction.

Rather than saying "I owe you 30" and reading words to see if I'm up or down, I can write "-30" and know it means I'm in the hole. If I earn money and pay my debts (-30 + 100 = 70), I can record the transaction easily. I have +70 afterwards, which means I'm in the clear.

The positive and negative signs automatically keep track of the direction — you don't need a sentence to describe the impact of each transaction. Math became easier, more elegant. It didn't matter if negatives were "tangible" — they had useful properties, and we used them until they became everyday items. Today you'd call someone obscene names if they didn't "get" negatives.

But let's not be smug about the struggle: negative numbers were a huge mental shift. Even Euler, the genius who discovered e and much more, didn't understand negatives as we do today. They were considered "meaningless" results (he later made up for this in style).

It's a testament to our mental potential that today's children are expected to understand ideas that once confounded ancient mathematicians.

Enter Imaginary Numbers

Imaginary numbers have a similar story. We can solve equations like this all day long:

$$x^2 = 9$$

The answers are 3 and -3. But suppose some wise guy puts in a teensy, tiny minus sign:

$$x^2 = -9$$

Uh oh. This question makes most people cringe the first time they see it. You want the square root of a number less than zero? That's absurd!

It seems crazy, just like negatives, zero, and irrationals (non-repeating numbers) must have seemed crazy at first. There's no "real" meaning to this question, right?

Wrong. So-called "imaginary numbers" are as normal as every other number (or just as fake): they're a tool to describe the world. In the same spirit of assuming -1, .3, and 0 "exist", let's assume some number i exists where:

$$i^2 = -1$$

That is, you multiply i by itself to get -1. What happens now?

Well, first we get a headache. But playing the "Let's pretend i exists" game actually makes math easier and more elegant. New relationships emerge that we can describe with ease.

You may not believe in i, just like those fuddy old mathematicians didn't believe in -1. New, brain-twisting concepts are hard and they don't make sense immediately, even for Euler. But as the negatives showed us, strange concepts can still be useful.

I dislike the term "imaginary number" — it was considered an insult, a slur, designed to hurt i's feelings. The number i is just as normal as other numbers, but the name "imaginary" stuck, so we'll use it.

Visual Understanding of Negative and Complex Numbers

As we saw last time [hn.my/arithmetic], the equation x^2 = 9 really means:

$$1 \cdot x^2 = 9$$

What transformation x, when applied twice, turns 1 to 9?

The two answers are "x = 3" and "x = -3": That is, you can "scale by" 3 or "scale by 3 and flip" (flipping or taking the opposite is one interpretation of multiplying by a negative).

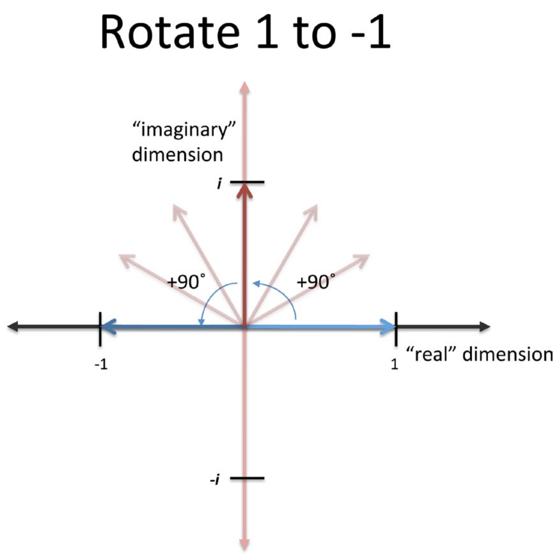
Now let's think about x^2 = -1, which is really

$$1 \cdot x^2 = -1$$

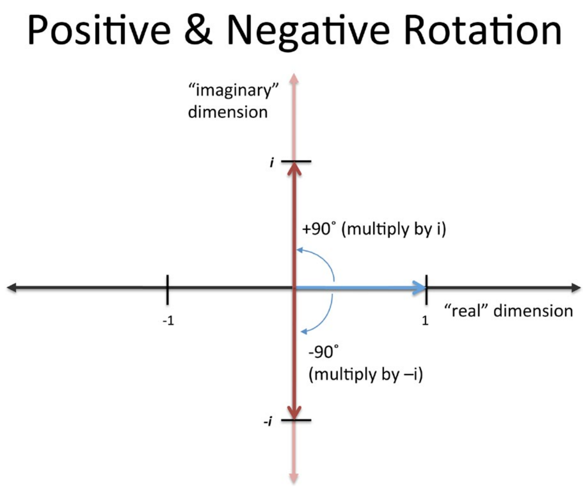
What transformation x, when applied twice, turns 1 into -1? Hrm.

- We can't multiply by a positive twice, because the result stays positive
- We can't multiply by a negative twice, because the result will flip back to positive on the second multiplication

But what about…a rotation! It sounds crazy, but if we imagine x being a "rotation of 90 degrees," then applying x twice will be a 180 degree rotation, or a flip from 1 to -1!



Yowza! And if we think about it more, we could rotate twice in the other direction (clockwise) to turn 1 into -1. This is "negative" rotation or a multiplication by -i:



If we multiply by -i twice, we turn 1 into -i, and -i into -1. So there are really two square roots of -1: i and -i.

This is pretty cool. We have some sort of answer, but what does it mean?

- i is a "new imaginary dimension" to measure a number
- i (or -i) is what numbers "become" when rotated
- Multiplying i is a rotation by 90 degrees counter-clockwise
- Multiplying by -i is a rotation of 90 degrees clockwise
- Two rotations in either direction is -1: it brings us back into the "regular" dimensions of positive and negative numbers.

Numbers are 2-dimensional. Yes, it's mind bending, just like decimals or long division would be mind-bending to an ancient Roman. (What do you mean there's a number between 1 and 2?). It's a strange, new way to think about math.

We asked "How do we turn 1 into -1 in two steps?" and found an answer: rotate it 90 degrees. It's a strange, new way to think about math. But it's useful. (By the way, this geometric interpretation of complex numbers didn't arrive until decades after i was discovered).

Also, keep in mind that having counter-clockwise be positive is a human convention — it easily could have been the other way.

Finding Patterns

Let's dive into the details a bit. When multiplying negative numbers (like -1), you get a pattern:

- 1, -1, 1, -1, 1, -1, 1, -1

Since -1 doesn't change the size of a number, just the sign, you flip back and forth. For some number "x", you'd get:

- x, -x, x, -x, x, -x…

This idea is useful. The number "x" can represent a good or bad hair week. Suppose weeks alternate between good and bad; this is a good week; what will it be like in 47 weeks?

$$x \cdot -1^{47} = x \cdot -1 = -x$$

So -x means a bad hair week. Notice how negative numbers "keep track of the sign" — we can throw -1^47 into a calculator without having to count ("Week 1 is good, week 2 is bad… week 3 is good…"). Things that flip back and forth can be modeled well with negative numbers.

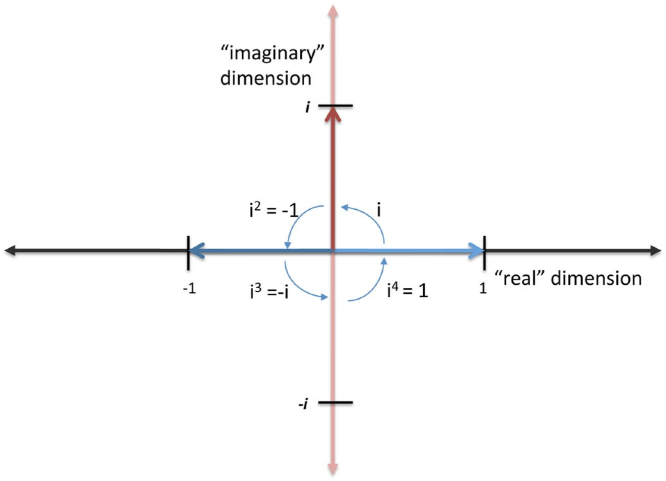
Ok. Now what happens if we keep multiplying by i?

$$1, i, i^2, i^3, i^4, i^5 \ldots$$

Very funny. Let's reduce this a bit:

- $1 = 1$ (No questions here)
- $i = i$ (Can't do much)
- $i^2 = -1$ (That's what i is all about)
- $i^3 = (i \cdot i) \cdot i = -1 \cdot i = -i$ (Ah, 3 rotations counter-clockwise = 1 rotation clockwise. Neat.)
- $i^4 = (i \cdot i) \cdot (i \cdot i) = -1 \cdot -1 = 1$ (4 rotations bring us "full circle")
- $i^5 = i^4 \cdot i = 1 \cdot i = i$ (Here we go again…)

Represented visually:



We cycle every 4th rotation. This makes sense, right? Any kid can tell you that 4 left turns is the same as no turns at all. Now rather than focusing on imaginary numbers (i, i^2), look at the general pattern:

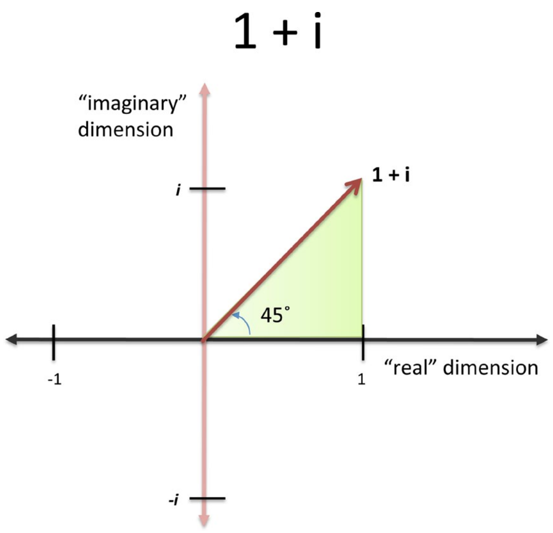
- X, Y, -X, -Y, X, Y, -X, -Y…

Like negative numbers modeling flipping, imaginary numbers can model anything that rotates between two dimensions "X" and "Y". Or anything with a cyclic, circular relationship — have anything in mind?

Understanding Complex Numbers

There's another detail to cover: can a number be both "real" and "imaginary"?

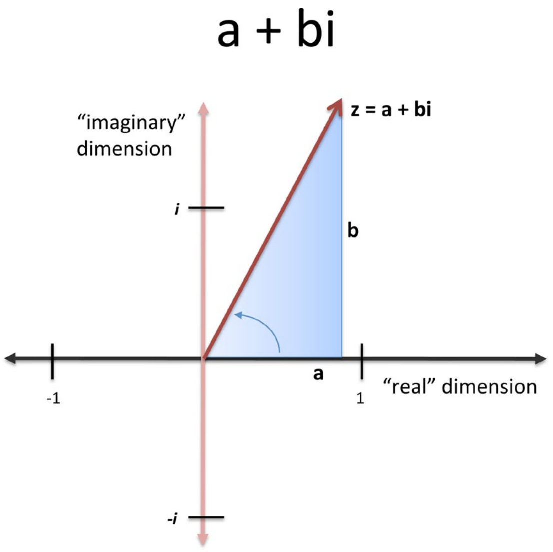
You bet. Who says we have to rotate the entire 90 degrees? If we keep 1 foot in the "real" dimension and another in the imaginary one, it looks like this:



We're at a 45 degree angle, with equal parts in the real and imaginary (1 + i). It's like a hotdog with both mustard and ketchup — who says you need to choose?

In fact, we can pick any combination of real and imaginary numbers and make a triangle. The angle becomes the "angle of rotation." A complex number is the fancy name for numbers with both real and imaginary parts. They're written a + bi, where

- a is the real part
- b is the imaginary part



Not too bad. But there's one last question: how "big" is a complex number? We can't measure the real part or imaginary parts in isolation, because that would miss the big picture.

Let's step back. The size of a negative number is not whether you can count it — it's the distance from zero. In the case of negatives this is:

$$Size of\,(-x) = \sqrt{(-x)^2} = |x|$$

Which is another way to find the absolute value. But for complex numbers, how do we measure two components at 90 degree angles?

It's a bird… it's a plane… it's Pythagoras!

Geez, his theorem shows up everywhere, even in numbers invented 2000 years after his time. Yes, we are making a triangle of sorts, and the hypotenuse is the distance from zero:

$$Size of\, a + bi = \sqrt{a^2 + b^2}$$

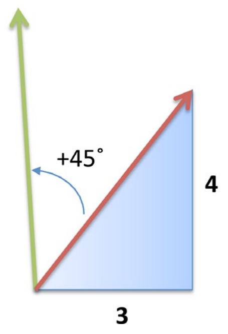
Neat. While measuring the size isn't as easy as "dropping the negative sign," complex numbers do have their uses. Let's take a look.

A Real Example: Rotations

We're not going to wait until college physics to use imaginary numbers. Let's try them out today. There's much more to say about complex multiplication, but keep this in mind:

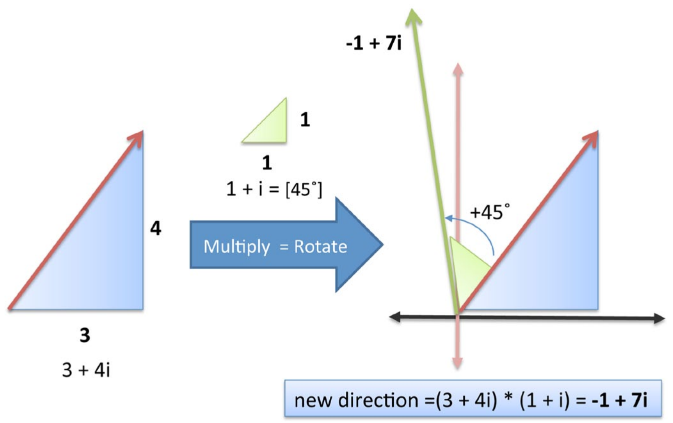
- Multiplying by a complex number rotates by its angle

Let's take a look. Suppose I'm on a boat, with a heading of 3 units East for every 4 units North. I want to change my heading 45 degrees counter-clockwise. What's the new heading?



Some hotshot will say "That's simple! Just take the sine, cosine, gobbledegook by the tangent… fluxsom the foobar… and…" Crack. Sorry, did I break your calculator? Care to answer that question again?

Let's try a simpler approach: we're on a heading of 3 + 4i (whatever that angle is; we don't really care), and want to rotate by 45 degrees. Well, 45 degrees is 1 + i, so we can multiply by that amount!



Here's the idea:

- Original heading: 3 units East, 4 units North = 3 + 4i
- Rotate counter-clockwise by 45 degrees = multiply by 1 + i

If we multiply them together we get:

$$(3+4i)\cdot(1+i) = 3+4i+3i+4i^2$$
$$= 3-4+7i = -1+7i$$

So our new orientation is 1 unit West (-1 East), and 7 units North, which you could draw out and follow.

But yowza! We found that out in 10 seconds, without touching sine or cosine. There were no vectors, matrices, or keeping track what quadrant we are in. It was just arithmetic with a touch of algebra to cross-multiply. Imaginary numbers have the rotation rules baked in: it just works.

Even better, the result is useful. We have a heading (-1, 7) instead of an angle (atan(7/-1) = 98.13, keeping in mind we're in quadrant 2). How, exactly, were you planning on drawing and following that angle? With the protractor you keep around?

No, you'd convert it into cosine and sine (-.14 and .99), find a reasonable ratio between them (about 1 to 7), and sketch out the triangle. Complex numbers beat you to it, instantly, accurately, and without a calculator.

If you're like me, you'll find this use mind-blowing. And if you don't, well, I'm afraid math doesn't toot your horn. Sorry.

Trigonometry is great, but complex numbers can make ugly calculations simple (like calculating cosine(a+b)). This is just a preview; later articles will give you the full meal.

Complex Numbers Aren't

That was a whirlwind tour of my basic insights. Take a look at the first chart — it should make sense now.

There's so much more to these beautiful, zany numbers, but my brain is tired. My goals were simple:

- Convince you that complex numbers were considered "crazy" but can be useful (just like negative numbers were)
- Show how complex numbers can make certain problems easier, like rotations

If I seem hot and bothered about this topic, there's a reason. Imaginary numbers have been a bee in my bonnet for years — the lack of an intuitive insight frustrated me.

Now that I've finally had insights, I'm bursting to share them. But it frustrates me that you're reading this on the blog of a wild-eyed lunatic, and not in a classroom. We suffocate our questions and "chug through" — because we don't search for and share clean, intuitive insights. Egad.

But better to light a candle than curse the darkness: here are my thoughts, and one of you will shine a spotlight. Thinking we've "figured out" a topic like numbers is what keeps us in Roman Numeral land.

There's much more complex numbers: check out the details of complex arithmetic [hn.my/complex]. Happy math.

Epilogue: But they're still strange!

I know, they're still strange to me, too. I try to put myself in the mind of the first person to discover zero.

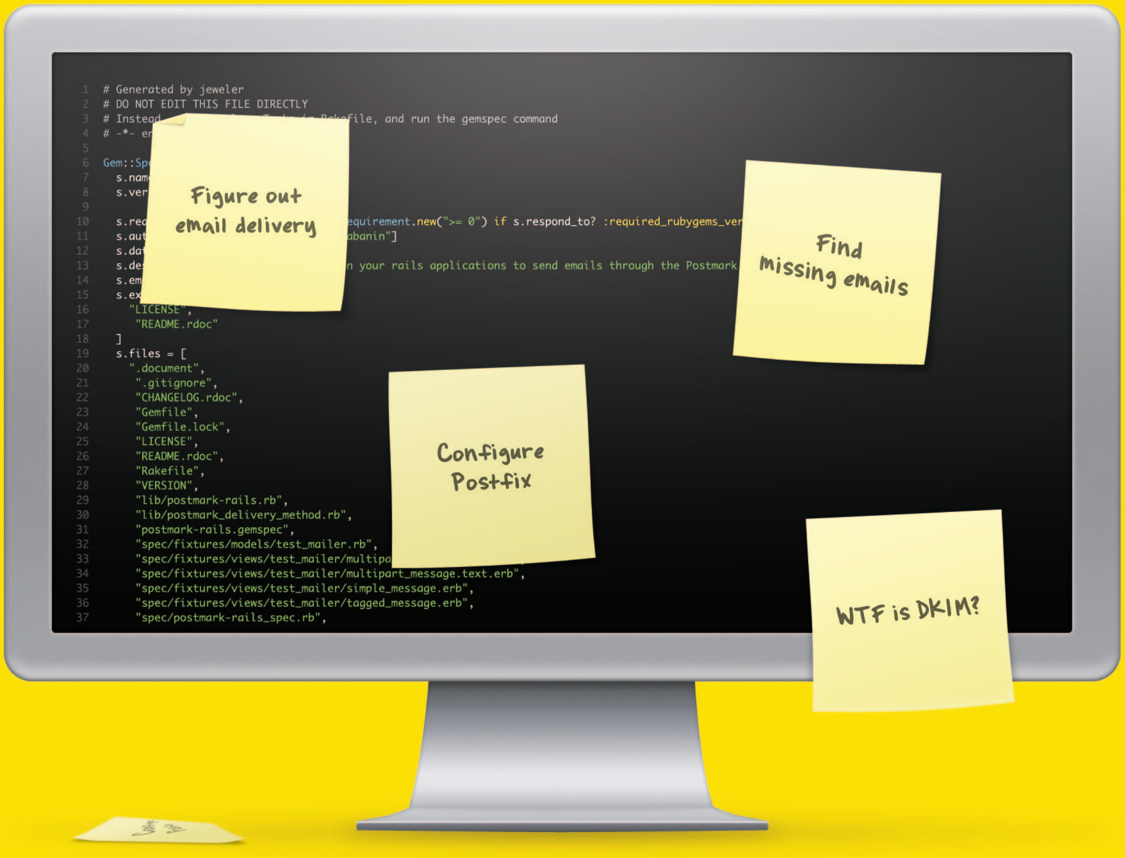
Zero is such a weird idea, having "something" represent "nothing," and it eluded the Romans. Complex numbers are similar — it's a new way of thinking. But both zero and complex numbers make math much easier. If we never adopted strange, new number systems, we'd still be counting on our fingers.

I repeat this analogy because it's so easy to start thinking that complex numbers aren't "normal." Let's keep our mind open: in the future they'll chuckle that complex numbers were once distrusted, even until the 2000s. ■

Kalid is a YC alum living in Seattle. He loves to simplify complex ideas, blog aha! moments at BetterExplained [betterexplained.com], and do math with *instacalc.com*

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

If you enjoy configuring
Postfix and Sendmail, ignore this.



```
# Generated by jeweler
# DO NOT EDIT THIS FILE DIRECTLY
# Instead                      kefile, and run the gemspec command
# -*- en

Gem::Sp
  s.nam
  s.ver

  s.re                            equirement.new(">= 0") if s.respond_to? :required_rubygems_ver
  s.aut                           abanin"]
  s.da
  s.de                            n your rails applications to send emails through the Postmark
  s.em
  s.ex
    "LICENSE",
     "README.rdoc"
  ]
  s.files = [
    ".document",
     ".gitignore",
     "CHANGELOG.rdoc",
     "Gemfile",
     "Gemfile.lock",
     "LICENSE",
     "README.rdoc",
     "Rakefile",
     "VERSION",
     "lib/postmark-rails.rb",
     "lib/postmark_delivery_method.rb",
     "postmark-rails.gemspec",
     "spec/fixtures/models/test_mailer.rb",
     "spec/fixtures/views/test_mailer/multipa
     "spec/fixtures/views/test_mailer/multipart_message.text.erb",
     "spec/fixtures/views/test_mailer/simple_message.erb",
     "spec/fixtures/views/test_mailer/tagged_message.erb",
     "spec/postmark-rails_spec.rb",
```

We'll deal with that
so you can get back to coding.

Postmark

hacker.postmarkapp.com
Transactional email delivery for web apps.

Dream. Design. Print.

MagCloud, the revolutionary new self-publishing web service by HP, is changing the way ideas, stories, and images find their way into peoples' hands in a printed magazine format.

HP MagCloud capitalizes on the digital revolution, creating a web-based marketplace where traditional media companies, upstart magazine publishers, students, photographers, designers, and businesses can affordably turn their targeted content into print and digital magazine formats.

Simply upload a PDF of your content, set your selling price, and HP MagCloud takes care of the rest—processing payments, printing magazines on demand, and shipping orders to locations around the world. All magazine formatted publications are printed to order using HP Indigo technology, so they not only look fantastic but there's no waste or overruns, reducing the impact on the environment.

Become part of the future of magazine publishing today at www.magcloud.com.

25% Off the First Issue You Publish

Enter promo code **HACKER** when you set your magazine price during the publishing process.

Please contact promo@magcloud.com with any questions.

MAGCLOUD