Chapter 1: Finding and Validating Ideas

The first step is knowing what you want to write about. If you picked up this book with a burning idea in mind, that is great. Write it down with as much detail as you can, and then follow the steps in this chapter to come up with ideas two through twenty. If you are starting from scratch, do not worry. Coming up with validated, well-scoped article ideas is not a daunting task when approached methodically. This chapter describes the exact process I have formulated through trial and error and repeated experience to generate well-scoped articles with market demand. My ideas rely on my existing skills and often prompt me to explore concepts that further my own learning.

> Every engineer has at least one good guide in them. — *Angel Guarisma*

I used to struggle to generate article ideas. I also used to rely solely on moments of inspiration to keep my content pipeline full. When I would pitch editors, it would take half an hour to write the email because I would have to scramble to find a decent topic and formulate a reasonable pitch. I used to dread the phrase "send me a couple of ideas." For the first ten or so articles that I wrote, every well-scoped topic with an outline that I actually stuck to was a miracle. More than once I had to cover my mistakes with an apologetic downward revision of either the article's scope or its delivery date.

This was a bad system and led to me working through some needlessly difficult rewriting to try to recover from insufficiently thought-out pitches. To fix this issue, I created a more rigorous system of idea generation and evaluation. Now, I maintain a list of dozens of un-pitched article ideas waiting to take spots in my publication schedule as repeat clients ask for new work and I find new publications to pitch to.

There are several angles to consider when determining what to write about. Your existing skills, experience, and interests; your prospective client's needs and wants; and your potential readership's knowledge, domain, and attention span must all align to create a great outcome for all parties. While prudent writers do consider their audience, both the publisher and the reader, in the planning process, you should start with yourself. If you are not writing about topics that you find intrinsically interesting and that are within your skillset (or ability to learn), it will not matter what the other parties think because there will never be a finished article for them to read.

For the rest of this chapter, we will mostly assume that there exist publishers and readers with your same interests. While we will consider existing clients' interests and the potential readership, future chapters will explore these constituencies in greater depth.

1.1 The 9 Questions Process

Set aside fifteen minutes with your favorite thinking tool (pen and paper, notes app, sidewalk chalk) to generate answers to these nine questions. Each question is designed to expand the list of potential topics, not constrain it. Write down everything you can possibly think of for each question, even if it does not feel possible right away. The next step will narrow your focus.

Run this exercise before you start writing or pitching clients and run it again if your backlog of ideas ever gets uncomfortably low. Ideally, after working through this process a few times, the questions will become second nature to the point that fully formed ideas are occurring to you just as often as you can execute them.

1.1.1 Questions

The questions are broken into groups of three by category. As an example, I completed the exercise myself; my answers are included at the end of this chapter. You may wish to use the worksheet included in Appendix B to complete the exercise.

Language Questions:

1. What programming languages do you know well enough to write about?
2. What frameworks and libraries within those languages do you enjoy using?
3. What company-specific technologies do you know well?

These three questions are very straightforward. Remember that you do not need to be an expert on a topic to teach it. If you have substantial experience, that is great, but as an intermediate practitioner of a language or framework, you remember what you struggled with as a beginner and can help others reach your level.

> We often overestimate the expertise or the knowledge of the general public in terms of something that we're working on. Maybe you've been working on...OAuth authentication in Rails for the last three weeks. Well, you are now at the top 0.01 percent of programmers around the world at OAuth authentication with Rails. It might feel obvious now because you spent three weeks working on it, but approximately everyone else has no idea how it works.... You might be pleasantly surprised if you write something that seems obvious to you. You might get a lot of people who say, "Wow I didn't know how that worked and that is super helpful." Don't be discouraged

> from writing about topics that seem obvious. — *Mark McGranaghan*

As a writer, you are introducing your readers to more than just technologies; you are introducing them to new ways of thinking and solving problems. Consider how technologies you know fit together and think about meta aspects like their community, support, and popularity.

> When you're learning web app development with Python, it's really useful to start with Django.... I love the Django community. I think that out of all of the programming communities, it is the most friendly to beginners. There are lots of beginner events—including Django Girls—running workshops all over the US and Europe. The Django conferences are beginner-friendly and encourage beginners. I like Django not only for the fact that it has all of the bits included, but also because I know that if I'm going to bring a new programmer into a community, this community is going to support them. — *Tracy Osborn*

Topic Questions:

4. Which topic domains are you interested in?
5. Who are your clients and what topic domains are they interested in? (It is okay to skip this question until you have read Chapter 2.)
6. Who would you like to land as clients and what topic domains are they interested in? (More about this in Chapter 2.)

Now that you have figured out what you can do, it is time for the fun part: figuring out what you want to do. Think of topic domains by developer-centered divisions, like data science and front-end development, but also by real-world application, like finance or medicine. If you do not have clients yet, skip number five, and if you are not looking for new clients, skip number six. However, never let your perceptions of your clients' interests convince you to skip number four. I have written about web scraping for an AI blog and AWS for a front-end magazine; editors tend to respond well to enthusiastic pitches, even for topics where the connection is tangential.

For question four, do not forget to list domains outside of programming that you are familiar with.

> Domain-specific knowledge is very, very useful in writing. Anyone can go and learn a technology and say, "Alright, here's how to write your first program in Python," but have no real background in it. However, if someone is an astrophysicist or a biochemist or a marketing growth expert and they learn how to use Python, it becomes, "How to write your own thing in Python that is relevant to being an astrophysicist." That magnifies the writing's value by a hundred-fold, not merely by being more specific in the topic but also being specific in the problem space. There is domain-specific knowledge that you know is going to go into an article like that. The writer is going to write from the perspective of, say, a biochemist and there're very few people doing that, so it adds a ton of value straight away. — *Peter Cooper*

Experience Questions:

7. What relevant prior projects have you done?
8. What real-world systems have you studied and could reverse engineer?
9. What environments have you programmed in?

Question seven is of vital importance. Having a prior project, even if it is not in great shape, that you can reuse all or part of as the foundation of an article makes the pitching, researching, outlining, and writing processes all substantially easier. Most of the first articles I wrote were based on personal or school projects that I had completed and could adapt to teach a concept. For example, the first article I wrote for FloydHub was about genetic algorithms. Though the final product was a series of examples of basic genetic algorithms in Python, the inspiration was a school project from over a year prior intended to teach object-oriented programming in Java.

> A lot of [what I write] is influenced by what I'm doing in my personal projects, and a lot of it is influenced by a certain timeliness to articles that help them gain traffic. If Svelte has just come out then you write about Svelte for a little bit. I think that what a lot of people do is start by trying to figure out what projects would make a good article topic, but I try to break it down a little further than that. Ask, "What is a good project?" and then, "What are the components that make up that good project?" Then, I try to break down every component into its own article so that people can try to piece together one, two, or even three articles. — *Chris on Code*

Good answers to question eight make great articles. The logic of modeling or reverse engineering a real-world system is easy for your readers to follow and thus provides a good structure to the article. You can add and remove details where appropriate to bring the article to the desired length and knowledge level.

Question nine is independent of the previous two. By environment, I do not mean IDEs or frameworks, but real-world situations in which you have written code. Academic backgrounds outside of computer science, the industry you work in and its domain-specific problems, or targeting a particular group of users are all interesting filters. Adding any one of those perspectives to an article can make the content uniquely engaging to a narrower market, which, unless taken to the extreme, is a good thing. Think about problems that you faced working in those environments. Do other people struggle with the same issues? If they do, solving a problem from your chosen environment is a great basis for an article.

> You have to be thinking not selfishly but selflessly; explain this problem and that helps not just me but everyone else that has this problem. Now to be fair, there could be some super narrow problem that only one person will ever have, but in reality most problems people run into a fair number of times, and it's best to carry that concept around in your mind as you're starting. —*Jeff Atwood*

1.1.2 Merging Answers

Reading over your answers will help article ideas form in your head. Write them down. A complete map would combine each answer to each question with every combination of every answer from every other question. Fortunately, we do not need to run that O(N^9) algorithm to find compelling article ideas. Instead, focus on taking one element from each category and seeing how they work together. Start by performing the obvious combinations, but do not neglect the surprising ones. What would it mean to write about using R to explore front-end development for startups looking to build chat apps? At first, it seems like these topics have little to do with each other, but a bit of creative brainstorming could lead to an article on building a web dashboard for analytics on real-time data.

For each combination that makes any amount of sense, write a couple of sentences describing the contents of the article. This is not a formal outline, just a brief description of the idea, enough so that you will be able to come back later and pick up on your thought process.

Also, write titles. Titles are a great way of narrowing down your audience. These are working titles; they do not have to make it to publication. The formulas "{Topic} in {Language} for {Environment}" or "Creating {System} using {Framework(s)}" should get the job done in most cases. Keep your titles precise and relevant; eschew clickbait.

> A perennial issue is focusing on titles. We did see a trend three to four years ago back where people were trying to be very clickbaity with titles. There were all kinds of publishing companies and sites that would take feel-good stories from the media and write these really long titles for them with "and you won't believe what happened next!" For example, "Someone found ten cute kittens in a bag and you won't believe what happened next." They would put that up, but it would just be a video like a YouTube embed or something and all they were doing was siphoning all of the traffic with these clickbaity headlines for things.... People are becoming wise to it, and they're becoming fatigued, in the same way they became fatigued by banner ads and things like that.
> — *Peter Cooper*

Throw away any article ideas that do not appeal to you or seem blatantly impossible, but try to keep an open mind about what you want to write throughout this nine questions process. Once you have fully concluded your brainstorming, you can move on to winnowing the list down to sure winners.

1.2 Idea Validation

At this point, you want to get a rough sense of the scope of and market for each idea before committing to pitching or writing any one of them. If you have ever performed market research for a product idea, it is a similar process, but it is also much easier and scaled-down. You do not have to perform nearly as thorough a validation for three reasons:

1. Articles have a low opportunity cost to create.
2. Articles are very flexible in scope.
3. Articles can coexist with very similar work while still providing value.

The opportunity cost of unsellable articles is fairly low as they only take a few hours to write and can often be repurposed or published on your personal site. Thus, you do not need to do as much market validation for an article as you would for a minimum viable product, which might take a hundred hours or more to develop. However, it is still important to perform these validation steps; it is better to sink five minutes into thinking up an idea that you cannot execute than five hours writing an article that you cannot publish. For a book or larger work, winnowing down ideas is a much more intensive process.

> The main filter that I wanted to apply was that I only wanted to write about things that I was very intimately familiar with. AWS is a vast topic meaning there are hundreds of services including lots of things I've never used myself. Basically, I didn't want to do any research

> or experimentation. I wanted something that could pretty much write itself, something that was basically a brain dump of things that I knew intimately, where I knew what I wanted to say. In the introduction we mentioned that we only talk about things that we have significant first-hand experience with. That was the first cut.
>
> Then there were a few other things that didn't make it. Ironically one of the products that my co-author and I worked on ourselves when at AWS, we didn't include in the book. It was part of the time-boxing process. We tried to group things that were related. The second thing is that we left out things that didn't relate together. The theme was infrastructure-related topics; there were other things like application monitoring and things like that which we could have included—we have first-hand experience with these things—but they would have made the book significantly bigger and would have taken longer to do. To be honest, we thought that it might be a good candidate for a sequel, a part two on a different topic set.
> *— Daniel Vassallo*

You can get a lot done in two thousand words, a hundred lines of code, and maybe a chart or two. The key issue, which we will explore in Section 4.1, is appropriately scoping your sample application or topic outline to include only relevant information and assume the right level of background knowledge. Right now, you mostly want to make sure that you are within the right order of magnitude. Some publications offer quick tips, writing about the semantics of a single line of code or short snippet might need expanding. On the opposite end of the spectrum, a topic like "Making a Website in Django" would need to be broken down substantially to extract achievable topics for single articles.

> We started Scotch with a lot of these simpler articles: how to do form validation in Angular, how to do it in React, how to handle event inputs in Vue. Then, we got to a point where we said, "Ok, let's do these really cool long-form articles, where we would do, 'How to Build XYZ with Technology 1, Technology 2, 3, and 4.'" What we found was that a few people read and followed through on those articles, but far fewer than the broader-term articles. — *Chris on Code*

When you have an idea, make sure there is a demand for it. Search for key terms for your title as if you were a reader trying to find your article. Skim the top few hits. If your idea turns out to be wholly original, then that is great, but you will need to do extra research to make sure there is an interested readership for it. Finding other articles on your topic is not a bad thing; on the contrary, it means that you have identified something interesting enough for other people to write about. However, if there are hundreds of articles, books, videos, and other resources on the topic (for example, say you want to write a general introduction to programming in Python), you will have to carefully consider your angle to stand out from the crowd. This angle could come from an answer to question four or question nine. Also, check the dates on the articles you find. If you are writing about a specific implementation and the best competing article is, say, five years old, for many topics that means your fresh take might be marketable. After this validation step, note any similar articles you find, as they will make good sources in the research phase.

Now is also a great time to audit the previous projects that you are using for inspiration. Make sure that you have the appropriate rights to repurpose the code and are not bound by any employment restrictions or other contractual obligations not to write about a given subject. For showstoppers like that, it is best to get ahead of the issue early.

1.3 Article Lifespan

In the validation phase, we considered how competing articles have aged to see if there was a market need for your idea. Now, we will think about what will happen to what you write over time.

> I'm trying to write about things that aren't, "Here's my technology of this particular year." You're tying yourself too much to very specific bits of technology and specific bits of time. You're not looking at the more timeless aspects of the underlying principles, the underlying human factors. If you look at my early blog, I did write about some very technical stuff that now seems completely pointless. It might have mattered for like a year, but nobody's going to care about that post anymore. That's fine. If you look at the ones that really matter, it's about the deeper human issues of how we get better at doing this as people. How do we become better versions of ourselves?... If you want to write stuff that's going to last, that's going to have a long shelf life, you want to avoid really meticulous, detailed, in-depth technical stuff because it has such a short shelf life in the big picture of things. —*Jeff Atwood*

While it can be valuable to focus on long-term trends and evergreen ideas, there is also a lot of value in up-to-date technical content. Publishers prove this value with their demand for technical tutorials.

> If it's helpful and reaches a problem people are having with current technology, it's not wrong to write that. You

> have to mix it up, though. You have to write a variety of content. Some percent of your content can be current technology and another percent should be relatively more timeless things about how to run projects, how to deal with people, how to deal with emotions—things that are more about sustainability in your career. For every post you write about stuff that's happening this year, write a post that's going to be relevant ten years from now. Try to mix it up, and keep a good balance going. I'm not saying don't write about current technology—I'm basically the "why not both girl." Why not both? *—Jeff Atwood*

Taking this idea a step further, you should write one-year articles for your clients and publish ten-year articles for yourself. This way, you get immediate payment for the shorter-lived work while the client bears the depreciation, and you accumulate the long-term benefits of having published a sustained resource. Publishing articles with a longer shelf life for yourself will help you build a strong presence with fewer pieces than if your writing goes out of date every year.

> I've been writing professionally since at least 2006. I spent a lot of cycles early in my writing career writing things that depreciated quickly, in the economic sense. For example, writing about the current best practice on Rails, especially back when Rails was moving at rapid speed, did not set me up for long-term professional success. I would probably not write on that topic today, particularly not if I was writing in a concerted fashion. The reason for that is that, as you mentioned, there are some posts that maintain their value after ten years and then there are some posts that are, like the Rails article, probably good for about eighteen months. Eighteen

months might sound like a long time. There are many written artifacts that lose more than ninety percent of their value within twenty-four hours. I like to say that *The New York Times* throws out more excellent writing than you will produce in your entire career every day because the value of the ninety-fifth percentile *New York Times* piece after forty-eight hours approaches zero. Given that you are vastly less resourced than *The New York Times* and you probably have a longer scale with regards to your decision-making horizon, you should intentionally try to get more of your pieces into the bucket where they will continue to be valuable for years ahead.

In 2012 I wrote a piece on salary negotiation advice for engineers, and because I was writing on blog software and blog software generally includes the date prominently all over the piece, I routinely get questions about that piece asking "Is this advice still good?" That's a frustrating question for me to be asked: of course the world has not advanced so much between 2012 and 2020 such that not negotiating your salary is suddenly a good idea. The thesis of that piece is that you should always negotiate your salary. Just the fact that I made the date so prominent in the piece caused people to believe that as soon as the date changed, which happens literally every day, then the piece starts losing value. I am more intentional now in writing to position my pieces as essays rather than blog posts, because essays have an anchor around timelessness and blog posts have an anchor around being up-to-the-moment. I'm intentionally writing most of my essays on subjects that will continue to be relevant to my readers in ten years. — *Patrick McKenzie*

This is also a good deal for your client. Their economics are substantially different from yours such that they can create a viable business model on content with a primary relevance that lasts for a few years at best.

> There exist publishers of technical content that either have much better monetization options with respect to one reader than an independent writer of technical content would, or they are selling something that is not itself content and so their investment in content is essentially a marketing expense for the core thing. For example, I work at a large software company. The core economics of large software companies are extremely healthy. The market characteristics of my employer are a bit different than the typical enterprise software company, but the typical enterprise software company might have margins approaching one hundred percent with deal sizes in the five or six figures. They can justify an awful lot of spending on a daily basis to produce content if it successfully gets decision makers for their industry to get in touch with the sales rep. That doesn't work out quite so well if you're an individual engineer yourself and you have some allocation to make of doing core billable work versus investing in your own marketing; you would generally want to write artifacts that would continue having value even in weeks or quarters where you were primarily doing billable work. — *Patrick McKenzie*

Not only are many publishers able to better monetize depreciating work, but some extract long-term value from it. If a publisher invests in keeping their old tutorials up to date, they can get fresh technical content for a smaller investment than commissioning original work.

> With a library as large as the one that we have...we rely on our open-source community to flag things that need to be updated for us. Or, because our library is open-source, community members might just make the update themselves and we'll review it. We also have somebody on staff who is dedicated to updating things—one person on the team who basically takes feedback from the community in the form of comments, GitHub issues, etc., and updates documentation accordingly. When a new version comes out of something, we're able to get to it as fast as possible.
>
> When someone arrives at a piece of documentation, it's important that you're not actually sending them off on the wrong track. Even if you write it really well, people still just skim down to the thing that they need, so they could miss that what they're looking at is not about the right version or something. We put deprecated notifications on guides, and we do this very purposefully because sometimes people use legacy systems and they need a Debian 5 version of a piece of documentation because that's what they're using. But, somebody will arrive there and miss the giant banner that reads, "Don't look at this guide if you're not using Debian 5."
>
> The reason that we write documentation here at Linode is to be genuinely helpful and to inspire confidence in people. We have to make that investment to hit both of those things—you can't be genuinely helpful if you're not updating your documentation and you can't inspire

> confidence if somebody breaks their system after following your advice. — *Angel Guarisma*

Tech moves fast, but many people still work with legacy systems every day and need high quality resources to help them solve their problems.

> We rarely delete versions because of the edge case where people use legacy systems and come to those old versions directly. — *Angel Guarisma*

The good news is that even if there is a competing technical tutorial of high quality, it might be old enough that your fresher take on the topic is still needed. The bad news is that it is really hard to write evergreen technical content beyond basic introductions because systems change so quickly. It is up to publishers to worry about keeping their content up to date. As a writer, I focus on creating my best work in the moment while making an educated guess regarding my work's longevity.

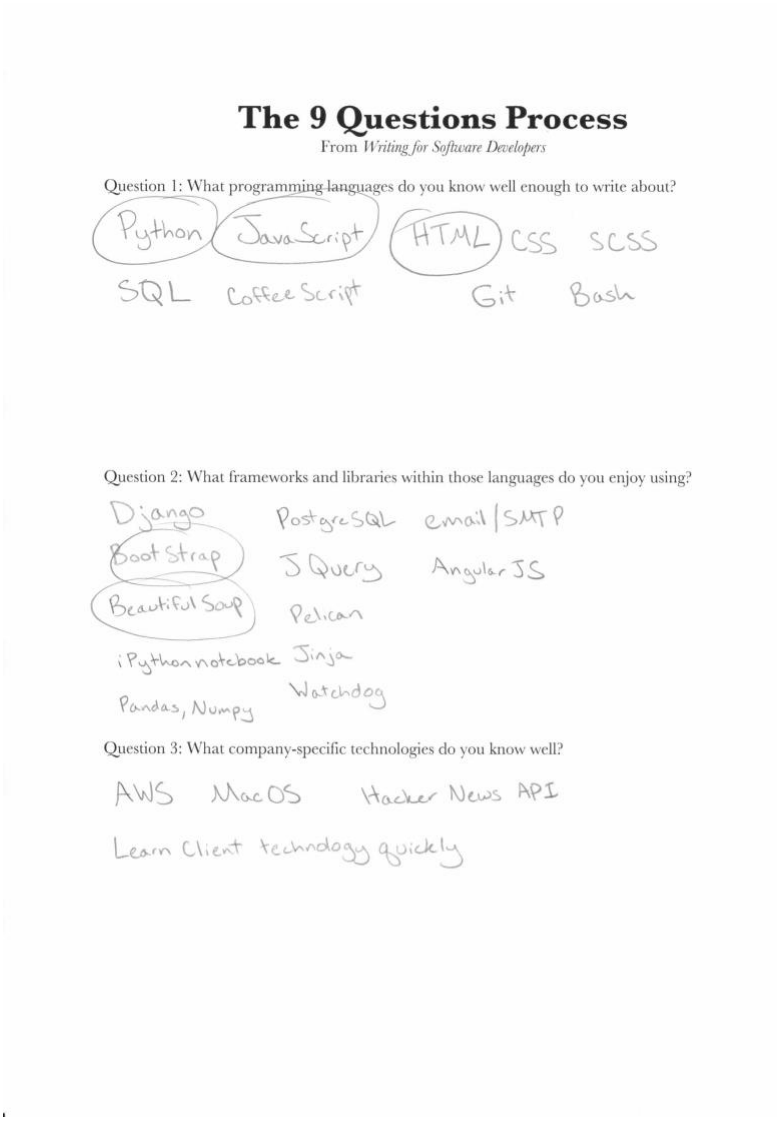
//TODO

1. Complete the 9 Questions Process Worksheet from Appendix B.
2. Combine your answers in as many ways as you can think of to generate article ideas. Try to come up with at least ten titles.
3. Pick the three titles that you are the most interested in writing. For each title, search keywords related to the topic to see if there is room on the market for your article.

Here are my answers to the worksheet. (In between answering these questions and publishing this book, I actually wrote and published an article with CSS Tricks based on my answers to this exercise, which I had not planned to do.)

Some answers from the worksheet are circled. These answers inform three titles we will be considering throughout this book. The tutorials *Scraping Shakespearean Sonnets* and *Shakespeare-Style Sonnet Generator* along with the technical article *Computational Poetry* are based on the circled ideas. These articles are presented and discussed in full in Act 2.

The 9 Questions Process Example



The 9 Questions Process

From *Writing for Software Developers*

Question 1: What programming languages do you know well enough to write about?

[✓ Python] [✓ JavaScript] [✓ HTML] CSS SCSS

SQL CoffeeScript Git Bash

Question 2: What frameworks and libraries within those languages do you enjoy using?

Django PostgreSQL email | SMTP

[✓ Boot Strap] JQuery AngularJS

[✓ Beautiful Soup] Pelican

iPython notebook Jinja

Pandas, Numpy Watchdog

Question 3: What company-specific technologies do you know well?

AWS MacOS Hacker News API

Learn Client technology quickly

Question 4: Which topic domains are you interested in?

Backend web dev

API creation and use

[✓ front end web dev] and design

[✓ Web Scraping]

Security

data analysis

database design

[✓ textualdatasets]

[✓ text generation]

System design

[✓ literature]

Finance

Startups

Question 5: Who are your clients and what topic domains are they interested in?

Smashing Magazine : Web development & front end design

Floyd hub: AI & Data Science

Wonder proxy: Internationalization

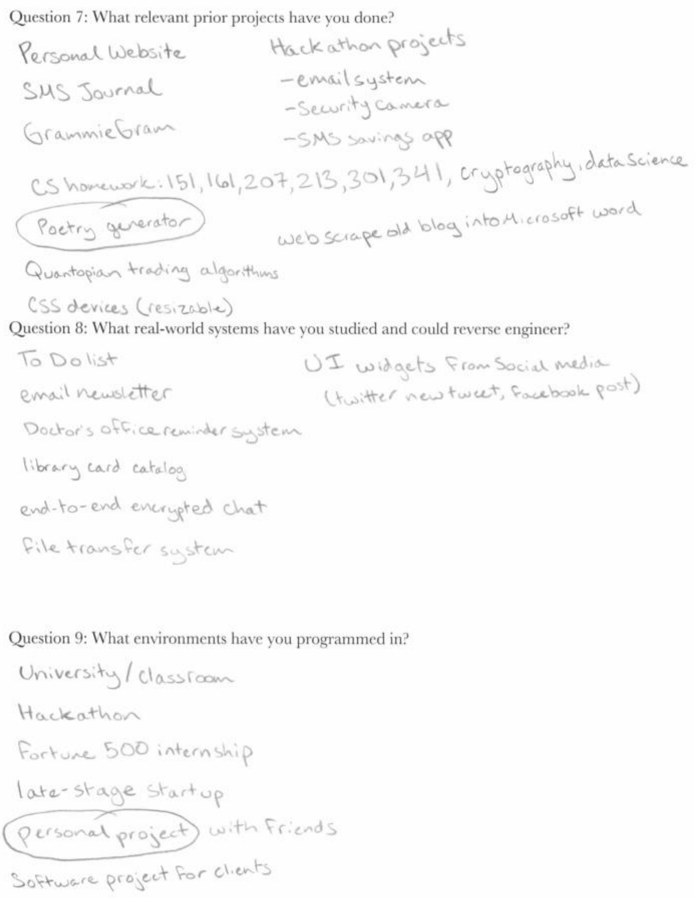
Twilio: Applications using Twilio services

Question 6: Who would you like to land as clients and what topic domains are they interested in?

CSS tricks: CSS

Stripe Increment : long form on listed topics on their website

No Starch Press: Full-length technical books



Question 7: What relevant prior projects have you done?

Personal Website

SMS Journal

GrammieGram

Hackathon projects
- email system
- Security camera
- SMS savings app

CS homework: 151, 161, 207, 213, 301, 341, cryptography, data science

[✓ Poetry generator]

web scrape old blog into Microsoft word

Quantopian trading algorithms

CSS devices (resizable)

Question 8: What real-world systems have you studied and could reverse engineer?

To Do list

email newsletter

Doctor's office reminder system

library card catalog

end-to-end encrypted chat

File transfer system

UI widgets from Social media (twitter new tweet, facebook post)

Question 9: What environments have you programmed in?

University / classroom

Hackathon

Fortune 500 internship

late-stage startup

[✓ Personal project] with Friends

Software project for clients