How to get into a programming bootcamp

By Joe Averbukh

How to Get Into a Programming Bootcamp

Joe Averbukh

This book is for sale at http://leanpub.com/programmingbootcamp

This version was published on 2015-04-08

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

A Personal Letter

I didn't know what I wanted to do after college. Regardless, ten days after graduation I started working full-time as a management consultant. I was basically a glorified excel jockey – it sucked. However, I found pleasure in hacking together VBA scripts to automate my workbooks and make them dance. That was cool.

I dreamed about working as a software engineer. Over the course of the next two years I discovered that learning to code and working a full-time job is hard. After several false starts, I decided to go all-in summer of 2013. I moved to SF and attended App Academy[1]. A month after graduating I showcased my final project to a random person who just happened to work at a startup[2]. Another month later, I pushed my first commit to production. I was wide awake, the dream became reality.

Attending a programming bootcamp was one of the best decisions I made in my life. I learned more about code in my first two weeks than the two years combined. I got my dream job and was making over six figures for the first time. Thanks to that job I paid off my tuition and saved enough money to go traveling throughout Asia. It really did change my life.

I had many questions when I was researching bootcamps: What is the application process like? What is a typical day like? How do I know whether the bootcamp is legit? Where are the graduates today? Am I really going to get a job after this? The list goes on.

Aside from a couple of blog posts from former students, there wasn't much information or advice online. To my surprise this still seems to be the case two years later. Since graduating I've spent several

[1] http://www.appacademy.io/
[2] https://www.quixey.com/

hours guiding friends through the process and coaching them on how to get accepted. After several requests I made the time to write out all the things I wish I knew before starting my journey.

Good luck on your application.

Cheers,

– Joe

Intended Audience

I wrote this guide for people who are exploring and/or seriously considering attending a programming bootcamp. More specifically:

Perhaps you've completed a couple of tutorials and think programming may be right for you. Alas, you don't have a C.S. degree though. But you heard about these bootcamps that teach you enough to get a job. It sounds too good to be true so you want some more information.

Perhaps you've already decided to apply but want more information and tips on the application process.

Or perhaps you've already been accepted to bootcamp but want tips for ensuring success.

If one or more of these accurately describe you, I am confident you will find this guide useful.

What is a Programming Bootcamp?

Programming bootcamps are intense vocational programs. A good one will give you the relevant skills to work as a web or mobile developer after graduating. Programs are typically 9-12 weeks, with some even being as long as six months. Longer programs will cost more and push back your employment start date, but they will be less intense and give you an opportunity to learn more before starting full-time. Nonetheless, a good 9-12 week program will adequately prepare you for work.

Class formally lasts about 8 hours each day. Only 1-2 hours will be lecture. Most of the time will be spent writing code. Including out-of-class time, expect to spend around 8 hours a day coding. If this does not sound appealing I would reconsider working as a programmer.

Similar to learning an instrument, learning to code is largely a function of practice. Lectures are meant to introduce and clarify programming concepts. The bulk of the learning comes from solving actual problems. Each day you will be assigned challenges and projects to complete. You will break into pairs and spend the whole day coding. Several times my partner and I would continue to work after class ended. If we still didn't finish I would continue working alone at home and compare my solutions with my partner the next day. There will also be an aggressive amount of assigned reading. Expect 1-2 hours per day and a couple of whole books to read every other weekend.

As a result of all this work, you will gain significant knowledge and build a marketable portfolio of projects. These will set you apart during the interview process. Having real working code samples

speaks louder than a C.S. degree.

After bootcamp ends, if you worked hard and learned well, you will be ready to enter the industry as a professional developer. But make no mistake, if you are serious about programming, you still have a long way to go. It takes longer than 12 weeks to be a skilled developer. However, a programming bootcamp will make you dangerous enough to start working as a software engineer!

Why Attend a Programming Bootcamp?

If you don't have a 4-year C.S. degree, attending a bootcamp is the fastest way to learn the basics of modern web/mobile development. You can try to learn the basics on your own, but it will be difficult. Modern web development is a complicated mess these days. There are several technologies and concepts you need to learn. HTTP, Databases, Authentication, Authorization, MVC, SQL, Python/Ruby/JavaScript, XSS, SQL-Injection, Caching, Deployment… the list goes on.

It can be overwhelming to try to learn all these on your own. Where do you start? What are the best resources for learning? How do all these concepts fit together?

Having an organized curriculum removes theses burdens and allows you to start learning immediately. There is a lot to learn and it's pretty complicated, the less confusion and friction the better it is for you.

Another major benefit of attending a bootcamp is that you will be surrounded by a group of similarly-motivated peers. This will keep you disciplined and honest. Spending 8+ hours a day, 5 days a week, for 3 months straight, learning to code is not an easy feat. Try doing this by yourself for two weeks, you'll soon understand what I'm talking about.

Having other people nearby will help you during the hard times. Sometimes you feel exhausted. Your peers will inspire you to press forward. Sometimes you can't get your code to run. You peers will help you hunt bugs. Sometimes you just feel stupid and not cut out to be a programmer. Your peers will feel the same way, and together you will cheer each other on.

Having others supporting you makes a huge difference. When you're in that room, surrounded by all those determined people, the energy is amazing. Personally, I think this is the biggest asset of an in-person bootcamp.

Another major benefit is having knowledgeable instructions and TAs. Bugs in your code can be very frustrating. What should have been a straight-forward five minute task can unexpectedly balloon into a two hour head-scratching session.

It can be very demoralizing to get stuck and not know how to proceed. Having an experienced eye diagnose the problem is a huge boon. Instead of wasting hours, you may lose only a few minutes. Furthermore, if the bug is not a simple syntax error, the instructor may be able to clarify a concept you misunderstood. Watching a more experienced developer debug is a lesson in itself. Next time you encounter a similar issue you will know a few extra tricks to diagnose the problem. Knowing how to debug is an important skill for a developer. Watching an experienced developer is one of the best ways to learn this skill.

Finally, attending a bootcamp is certainly going to be a unique experience. You will learn so much so quickly. I learned more in my first two weeks than in two years of on and off self study (while working full-time). Furthermore, you will make great friends who will also be your colleagues in the industry.

I don't think it's an exaggeration to claim that going through a programming bootcamp will change your life. Here's an anecdote about a guy in my cohort, we'll call him Jeff (not his real name).

Jeff was a smart, hard-working and all-around great guy. He was a family man too – married and father to a toddler. He worked for a research lab before attending bootcamp. Despite his advanced degree and research experience, he was barely making above minimum wage. In any case, he saved up his money for our bootcamp. For three months he was away from his wife and son. Jeff was making a huge bet.

He was the first person in our class to get a job. $120k, benefits, relocation bonus, the works. We were all stoked for him. As of this writing he still works for the same company and still enjoys code. He moved his family up to San Francisco and they are living a much more stress-free life. There's no doubt in my mind that bootcamp changed Jeff's life.

Honestly, it changed mine too. There's a good chance it would change yours as well.

The Application Process

The application process is fairly straight-forward. It contains a few parts. First you respond to a questionnaire and maybe submit a video. These are meant to introduce yourself and indicate your interest in the bootcamp.

Next you will be invited to complete a timed programming assessment. This will gauge your understanding of basic programming concepts. All of which can be learned free of charge on the web.

If you pass the coding challenge, you will be invited for a live coding challenge over Skype. The problems will be harder so make sure you're prepared. You may have to do multiple interviews before getting accepted.

Finally, you may have to complete some pre course-work before being officially accepted. This means you're pretty much in. Finish the assigned work and feel free to celebrate – you've been accepted!

The turn-around from initial application to conditional acceptance (being assigned pre course-work) is typically 1-2 weeks. However, you can push this back if you feel like you need additional time between steps. This part of the guide focuses on each step in greater detail.

Send in the Application

So you've decided you want to apply but you haven't taken that first step yet. Don't worry, the initial application is easy. You will most likely have to fill out a simple questionnaire and maybe send in a video. The questions will be generic and ask things like what's your background, education, and interest in programming. The video will probably ask you to introduce yourself and talk about something you know or ask you to teach a skill. You really should not sweat this. There is nothing to game here. Your best bet is to be completely honest.

If your only interest in programming is to get a high-paying job you should say that. Although some may think this is the "wrong" reason to learn how to code, I strongly disagree. Working as a software engineer is a job like any other. At the end of the day you are exchanging your time and brain cycles for money. Sometimes this means you will have to do mundane brain-numbing crap. If you're only interest in programming is "to do interesting work" then you might have a hard time staying motivated when you're asked to maintain someone else's buggy code. If you understand that software engineering is still work at the end of the day you will have higher average productivity.

That being said, if you find code to be the coolest thing then that's awesome! Did you hack on computers when you were a kid? Write about it! Did you automate some repetitive process at work? Share the story! Working as a software engineer was a dream of mine. I did not hesitate to share that fact in my application. If this is true for you then you should not hesitate to share either.

Another common reason is because you want to do your own startup. This is a pretty good reason for learning how to code. Sure you could always hire someone else to do all your code, but it's

not easy to find a talented individual or team that can build your vision. Moreover, those people probably have ideas of their own. Why should they work with you? There are plenty of "business" or "idea" guys out there with impressive credentials.

If you know how to code you will gain a lot of respect from your engineers. Use programming as a tool to build the future you want. If your true aspiration is to be an entrepreneur feel free to share that in your application.

Again, the initial application is nothing to fret about. Be clear and honest, it will start your relationship with the bootcamp on the right foot.

If you're unsure whether you want to apply, just do it. I wasted a whole month staring at that apply link and being indecisive. I finally said screw it and applied. The whole process from application to acceptance took only one week.

If you're not sure about attending, why don't you get accepted first and then make that decision afterwards. Much better than wasting time hypothesizing.

So what are you waiting for? Send in that application!

Prepare for the Programming Challenge

Within a few days of sending the initial application you should hear back about next steps. I encourage you to reach out regarding the status of your application if a week has gone by and you still haven't heard back. It's possible your application got lost accidentally.

The next step will most likely be a timed exercise. You will have to solve a couple of straight-forward programming challenges. These are not supposed to be difficult. Rather, they will gauge whether you put in the minimum effort to learn basic programming syntax and control flow. Additionally, these questions will test your problem solving skills. Here are some example questions.

```
Given a list of integers, write a program that calculat\
es their sum.
Input: [4, 0, -1], Output: 3
Input: [1, 2, 3, 4], Output: 10
Input: [], Output: 0
```

```
Given a number n, write a program that calculates n!
Input: 3, Output: 6
Input: 5, Output: 120
Input: 1, Output: 1
Input: 0, Output: 1
```

```
Write a program that returns all the prime numbers less\
 than 100
Output: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41\
, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

To solve these you will need to know about variable assignment, for/while loops, and maybe recursion. These are not advanced concepts, you should not struggle.

Don't fret If you've never programmed before. I applied to a bootcamp that taught Ruby/Rails and had never used either (granted I did have some experience in other languages). There is a good chance your program will recommend some introductory resources if you haven't coded before. Complete them, all of them. If you do this you will certainly be prepared. I finished all the tutorials and the first 10 chapters of Chris Pine's "Learn to Program"[3] (highly recommended for ruby) in a couple of evenings after work. Thanks to these resources, I was able to easily pass the first challenge.

If you took the challenge and found it difficult don't give up yet. You may still be invited for the next interview. In any case keep practicing! There are a lot of great resources out there. Codeacademy, Ruby Monk, Learn Python The Hard Way, and learnyounode are all great for Ruby/Python/JavaScript. In the following sections, I go over some helpful tips to further increase your chances of passing this interview.

[3] https://pine.fm/LearnToProgram/

Programming Challenge Tip #1: Use the Terminal

Knowing how to run your code locally on your computer is extremely important. Many tutorials will have you run code through an online interface. This is good for getting your feet wet, but you will need to know how to run code from the command line. It truly amazes me how few people know how to do this when attempting the first challenge. This is the first thing I tell my friends to do when applying. It will significantly increase your chances of passing this challenge. With this you can know for sure whether your code will pass the example test cases.

Let's solve an earlier example problem.

```
Write a program that returns all the prime numbers less\
 than 100
Output: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41\
, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

Here's a Python solution.

```
#!usr/env/bin python

def primes_less_than_100():
    num = 2
    stop = 100
    primes = []

    while num < stop:
        num_is_prime = True
```

```
        for prime in primes:
            if num % prime == 0:
                num_is_prime = False
                break

        if num_is_prime:
            primes.append(num)

        num += 1

    return primes

if __name__ == '__main__':
    primes = primes_less_than_100()
    print primes
```

Copy and paste this into your favorite coding text editor. If you don't already have one, I suggest using Sublime[4].

Save this into a file and name it *primes.py* Next open up Terminal or Command Prompt if you are on a Mac/PC respectively. Navigate to the directory where you saved *primes.py* Now type `python primes.py` and hit enter. You should see the expected output printed below. if you got an error check that python is properly installed and configured on your machine. Make sure there was no copy/paste error either.

Note: If you've never used the command line before you should google it up. Knowing how to find programming help is an important skill as well. You will frequently need to look up how to do things as a developer. Might as well practice now.

I encourage you to solve this problem in your language of choice. Solve the other problems I mentioned earlier as well. Run your

[4] http://www.sublimetext.com

solutions locally and verify that you get the right outputs for the specified inputs.

When completing the online code challenge, write your solutions in your text editor and run them on your computer. Test your solutions with various inputs and make sure you get the correct output. Once you're confident that your program works, copy+paste your answers in the online form. Take a breath and hit submit!

Programming Challenge Tip #2: Write Tests

Tests are a superb way to guarantee the expected behavior of your program. It can be annoying to manually keep running your program for different inputs. Tests will automate this and increase our confidence in the correctness of the program.

Suppose we wanted to test `primes_less_than_100` from *primes.py*. Let's create a new file called *test_primes.py* and save it in the same location. Copy and paste the following into our new file:

```
#!usr/bin/env python

from primes import primes_less_than_100

expected_primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \
31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89,\
 97]

actual_primes = primes_less_than_100()
error_message = "\nExpected {}\n Actual {}".format(
    expected_primes, actual_primes)

assert actual_primes == expected_primes, error_message
print "All tests pass!"
```

Now fire up Terminal and type `python test_primes.py`. If you see "All tests pass" then we know everything worked as expected. For fun let's cause an error on purpose. Make the following change to the assignment of `actual_primes`.

```
actual_primes = []
```

Run *test_primes.py* again. You should now see an error like this:

```
Traceback (most recent call last):
  File "test_primes.py", line 11, in <module>
    assert actual_primes == expected_primes, error_mess\
age
AssertionError:
Expected [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 4\
1, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
 Actual []
```

This makes sense, we were expecting `actual_primes` to be a list of the first 100 primes but it's not since we set it equal to the empty list. Set `sctual_primes` back to what it was before and re-run `test_primes.py`. It should work as expected now.

Writing tests is a good practice for software development. There is even a discipline surrounding it known as TDD[5]. Not only does writing tests increase our confidence in our code, it gives us a safety net for making modifications. For example, let's improve `primes_less_than_100` from *primes.py* to make it slightly more pythonic.

Open *primes.py* and paste `primes_less_than_100_v2` so that the file looks like this

[5] http://en.wikipedia.org/wiki/Test-driven_development

```
def primes_less_than_100()
    # our code from earlier ...

def primes_less_than_100_v2():
    start = 2
    stop = 100
    primes = []

    for num in xrange(start, stop):
        num_is_prime = True
        for prime in primes:
            if num % prime == 0:
                num_is_prime = False
                break

        if num_is_prime:
            primes.append(num)

    return primes

if __name__ == '__main__':
    # our code from earlier ...
```

Instead of using a `while` loop we use `for in`. Doing so will automatically increment `num` by `1` at end of each iteration in the outer loop. In other words we don't need to include `num += 1` like we did in our while loop. Forgetting to increment in a while loop is a common programming error so using `for in` is a good python practice.

Let's update *test_primes.py* as well, it should look like this now:

```
#!usr/bin/env python

from primes import primes_less_than_100, primes_less_th\
an_100_v2

expected_primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \
31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89,\
 97]

actual_primes = primes_less_than_100()
error_message = "\nExpected {}\n Actual {}".format(
    expected_primes, actual_primes)

assert actual_primes == expected_primes, error_message

actual_primes_v2 = primes_less_than_100_v2()
assert actual_primes_v2 == expected_primes

print "All tests pass!"
```

Re-running *test_primes.py* should again return "All tests pass!" Now we know for sure that `primes_less_than_100_v2` works correctly. We don't need two functions that do them same thing so let's tidy up.

- Delete the `primes_less_than_100`
- Rename `primes_less_than_100_v2` to `primes_less_than_100`
- Delete the tests again, `primes_less_than_100_v2` since that no longer exists.

This may seem tedious, but this process of going from working-state to working-state is a fantastic way to improve your code quality without breaking things.

When doing the online code challenge, writing tests may be overkill. You should manually test your solutions first. Only if you have lots of spare time should you write automated tests. If you get the correct outputs then most likely have an acceptable solution.

Why "most likely?" If the tests pass doesn't that mean you got it right for sure?

Not necessarily. Consider the following solution to the primes problem:

```
def primes_less_than_100_v3():
    return [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,\
 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

Pretty silly huh? But this will pass the tests. Be aware your tests will never be fool-proof. For the most part though, as long as you're not doing something like the above, you can place faith in your tests.

Programming Challenge Tip #3: Practice!

I've mentioned this before and it really goes without saying. The more practice problems you do the better. But don't over-do this. You shouldn't need more than a week for this. One of my friends spent three months "practicing" before attempting the initial challenge. Although he did pass and get in I honestly thought it was unnecessary. Several people I know got in after practicing for a couple of evenings, myself included. A couple diligent sessions is better than months of leisure effort. Move fast if you're serious about attending a bootcamp.

Prepare for the Skype Interview

After passing the initial coding challenge you will likely be invited to do a Skype interview. Be wary of any bootcamp that does not include some sort of formal interview – they probably are not competitive and are lower-quality. The Skype interview will last between 30-60 minutes.

Most of the time will be spent coding over an online code-pad (like Stypi[6]) and any remaining time will be devoted to you for asking questions. The programming challenges will be significantly harder but still reasonable. If you thoroughly prepared you will be fine. This challenge will not test you on any advanced CS concepts like data structures or time complexity. I remember one of my friends discussing graphs, stacks, and queues to solve a problem that only needed an inner and outer while loop.

This challenge will further test your problem solving skills and knowledge of basic programming syntax and control flow. The most complicated techniques you will most likely use are double loops and/or recursion.

If you pass the interview then you will most likely be accepted. You may be asked to do a second interview as well, this is a good sign too. Anything that moves the process forward is a positive indicator for acceptance.

Read on for some tips to ace this Skype interview.

[6] https://code.stypi.com/

Skype Interview Tip #1: Ask for Clarifications

Make sure you fully understand the problem you are asked to solve. Believe it or not, people often answer a different question than the interviewer asked. This is because we can be quick to make assumptions. Do not assume anything. Verify you understand the problem and the constraints with your interview before starting. Always ask if anything is unclear. Also, feel free to ask additional questions as they arise. This communicates thoughtfulness and attention to detail. Both are important traits of a strong developer.

For example, consider the following question:

```
Given a number n, write a program that calculates n!
Input: 3, Output: 6
Input: 5, Output: 120
Input: 1, Output: 1
Input: 0, Output: 1
```

This solution would satisfy the test cases.

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

But We may want to ask the interviewer the following:

- Is `n` always an integer?

- Is `n` always non-negative?

Sure, that is what the interviewer most likely intended, but you should always ask if you're not sure. The interviewer may respond with something like

> Yes you can assume that `n` is an integer but actually, let's say it can be negative. In that case let's return `-1`.

So now we need to handle the negative case. This means we'll have to slightly adjust our code. A complete solution would look like this.

```
def factorial(n):
    if n < 0:
        return -1
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

It's okay if you didn't think of asking these questions. The earlier answer is acceptable. Solve the simpler case first. Just keep in mind the more clarification you get, the better.

Skype Interview Tip #2: Verbalize Your Thought Process

The goal of this interview is to demonstrate your problem-solving skills. You don't have to write a bug-free solution to pass this interview. In fact, you don't even have to completely get the question correct. If you can clearly communicate your thoughts and justify your thinking you will most likely pass.

It's no good to just write code and not explain why you did what you did. The interviewer won't know whether you actually understand the program. Most programming interview questions can be found online. It's certainly possible you heard the question before and that's why you can write that elegant one-line solution. But do you understand it? You must clearly communicate you do.

I cannot over-emphasize this point. Although it's preferable to correctly answer the given question, it's not always a requirement to pass the interview. This is the case for full-time interviews as well. What is required is clearly communicating your thought process.

Spend a few moments devising an approach. Once you have it, verbalize your thoughts. Don't panic if you realize your current approach won't work. Calmly explain your realization and take a few more moments to come up with a new plan. Let's consider the following problem:

```
Given a list of integers, write a program that calculat\
es their sum.
Input: [4, 0, -1], Output: 3
Input: [1, 2, 3, 4], Output: 10
Input: [], Output: 0
```

Here's a solution that passes all the test cases:

```
def sum_list(int_list):
    sum = 0
    for num in int_list:
        sum += num
    return sum
```

Verbalizing this would sound something like the following:

> Okay so I've written a function `sum_list` that takes one parameter, `int_list`. `int_list` is expected to be a list of integers. This function, `sum_list`, will return an integer equal to the sum of all the integers in `int_list`.
>
> I want a placeholder for the sum so I initialize a variable, `sum`, to `0`. Next I iterate over the integers in `int_list` and add them to `sum`. Once the loop completes, `sum` will equal the sum of all the integers in `int_list`. This is the output we want so I return `sum`.

You obviously don't have to be this verbose, but you can see this explicitly describes what's going on in `sum_list`. The key things to explain are:

- The function's name and it's parameters
- The function's return value(s) and type(s)

- The significance of each line. Focus on "why" you are doing what you're doing rather than just the what.

You should ask your interviewer whether he or she would like to step through the code together with a test case. This won't be necessary if your interviewer is already satisfied with your solution. But more often then not they will happily accept. Let's verbalize the test case `[4, 0, -1]`

> So let's try this with the test case `[4, 0, -1]`. We expect the final output to be 3. At the beginning we set `sum` equal to `0`. Next we begin iterating over `num_list`. The first integer is `4` so `num` is equal to `4` now. We increment `sum` by `num` so `sum` is `4` as well. Next `num` is equal to `0` so sum remains at `4`. Finally `num` equals `-1` and `sum` is updated to 3. We return `sum`, which equals 3, in the last line so our output is 3 – what we expected. The solution passes this test case.

Again, you don't need to use so many words. But by meticulously stepping through the code you can quickly reveal any bugs you may have missed.

When preparing for the Skype interview I highly encourage you to verbalize your solutions to any practice problems you are doing. Better yet, grab a friend and go to a white board. Ask them to give you example programming questions and solve it in front of them, explaining your thought process along the way. Emphasize what you're doing and especially why. Definitely step through a test case. If you do this you will feel much more comfortable during the Skype interview.

Skype Interview Tip #3: Listen to your Interviewer

If you've been verbalizing your thought process then you probably have a dialogue going with your interviewer. While stepping through your solution you may encounter a bug and discover you're on the wrong path. At this point the interviewer may offer some feedback and give gentle hints. You should listen.

Sometimes I had to interview candidates for our team. I usually ask an abstract logic problem that often requires clarification. It was really annoying dealing with candidates that acted like they were all-knowing. Perhaps they were afraid that otherwise we wouldn't hire them? I'm not sure.

In any case, these people would not take hints when they were on the wrong path. It was really disappointing. My favorite interview question has multiple parts and it's frustrating to see someone get stuck early and be too stubborn to listen to feedback. I personally wouldn't want to work with those developers. They may be great individual contributors but are probably difficult team players.

Being a good listener creates a favorable impression. Ultimately your interviewer is the one deciding your fate. Listen to them when they speak.

Skype Interview Tip #4: Never Do More Than Two Things

In my mind there are four things you would be doing during the interview: thinking, speaking, coding, and listening. You should never do more than two of them at a time. Listening is a special case. When you are listening you should not be doing anything else. It will be a mess if you try to think, talk, and code at the same time. You will most likely confuse yourself and the interviewer. I understand you may feel pressure, believe me I remember that feeling during my interviews. Don't let that pressure trip you up.

After getting clarification on the problem ask for a couple minutes to think. The interviewer will oblige.

Spend no more than three minutes thinking and coding up an approach. If you spend more time the silence may feel awkward. If you're not done after three minutes, **stop** coding. **Start** explaining your approach to the interviewer. You should also be thinking of next steps to verbalize afterwards.

Continue thinking and talking until you feel you have a complete solution. Verify whether your approach makes sense to the interviewer. If they affirm they will probably ask to see the rest of the code. At this point you can stop talking and continue coding and thinking. Once your solution is complete start walking through it with your interviewer.

Don't panic if in the middle of thinking and coding you realize that your current approach won't work. Don't scrape your work yet either. Stop coding and begin explaining to your interviewer what you were trying to do. Explain why you understand it won't

work as well. Use this time to think of another approach and segue the conversation into a discussion of your new idea. Doing this communicates more thoughtfulness than just muttering to yourself and deleting your code.

Notice how at any point we are either thinking and coding, thinking and speaking, or attentively listening. Never do more than two things at once.

Skype Interview Tip #5: Solve Specific Cases First

Sometimes a solution will not be obvious to you. Let's take a look at the factorial problem:

```
Given a number n, write a program that calculates n!
Input: 3, Output: 6
Input: 5, Output: 120
Input: 1, Output: 1
Input: 0, Output: 1
```

Ultimately we want a function that does something like this:

```
def factorial(n):
    return n * (n - 1) * (n - 2)  ... * 1
```

We obviously can't use this notation in our code. So how do we do it? Well let's solve specific cases first. We know the following:

- factorial(0) –> 1
- factorial(1) –> 1
- factorial(2) –> 2 * 1 = 2
- factorial(3) –> 3 * 2 * 1 = 6
- factorial(4) –> 4 * 3 * 2 * 1 = 24

Do you see the pattern here? The generic solution is in the form of `factorial(n) = n * factorial(n - 1)`. Recursion makes sense here.

```
def factorial(n):
    return n * factorial(n - 1)
```

But wait, what happens when `n = 0`? This code won't terminate. We need to add the base cases. Here's an updated solution.

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

Now our code terminates and passes all the test cases. By finding the outputs of specific cases first, we may find a pattern that reveals the general solution. I personally used this trick during my Skype interview and during my job interviews. This technique is especially useful for solving recursive and double-loop problems.

Skype Interview Tip #6: Show Interest

There will be time towards the end of the interview for you to ask questions. This is a great opportunity for you to show your enthusiasm for the program. One of the best ways to do this is to come up with at least 2-3 genuine questions about the program. Here are some examples:

- What kind of people get in?
- What are the best ways to prepare for bootcamp success?
- What is the hardest thing about bootcamp?

These questions show genuine interest and also give you an opportunity to demonstrate why you are a good fit. A friend once told me that one of the best ways to "hack" an interview is to flip it around. Interview the interviewer. Figure out who he or she is looking for and then communicate how you are that person. It's a neat trick and works well if you are sincere and genuine.

Complete Pre-Coursework

You finished all the tutorials, passed all the interviews, and got accepted – major congratulations! Ah, but you may find out that your acceptance is conditionally based on completing some coursework and passing an additional test. Don't worry, you've come this far, you're almost there!

If you can spare the time before your cohort begins, take a few days off to relax. You've earned it. Otherwise start working asap.

The pre-coursework may take anywhere between 20-80 hours depending on your current programming ability. It is important you thoroughly understand the concepts in the material. You want to hit the ground running from day one. If you put in the effort you will pass any additional tests or interviews.

The pre-coursework will test how serious you are about studying and putting in the effort to learn. Take it seriously and come prepared for day one of bootcamp!

Additional Information

This part of the guide covers additional questions you may have about bootcamps. These are questions I had myself and a few others that were asked by friends. I hope you find this information useful.

Why Most Bootcamps Teach Web Development

Most bootcamps teach web development because there is an abundance of jobs available. The startup scene has heated up the market. The demand for engineers significantly outweighs the supply.

As a result, the barrier to entry is lower for a junior developer. This is good news for both you and your bootcamp. Your school wants you to get a great job, it gives them another success story under their belt. It's also genuinely a nice feeling when your students succeed.

More web developers in today's market is a win-win-win for students, schools, and employers. Hence the reason web development bootcamps have become popular.

What Kind of People Attend Bootcamp

All kinds. It's not just CS, engineering, or math majors. Sure there are plenty of those. But lots of people come from life sciences, humanities, and the arts. I remember one of our best students was a sociology major. She did incredible despite having zero coding or engineering background. She was extremely bright and motivated. She frequently stayed late after class to methodically finish assignments. She got multiple offers and was able to leverage them to get the job she wanted with an incredible six-figure package.

Although everyone's background was unique, there were a few common denominators. People were generally extremely motivated, bright, and enjoyed problem-solving. The last one is especially important. If you don't enjoy problem-solving you will detest programming. You don't need to already be an engineer to be successful in the bootcamp. But you do need to be a problem-solver. And even then, you will have a hard time leaning if you lack the motivation.

One last thing. I personally think there is something eccentric about the students who attend bootcamp. You've got to be a little crazy to pay all this money and put yourself through this coding hell.

But it's worth it – and the cool people that attend are a big part of that as well.

How Much to Budget for Bootcamp

Aside from tuition, I recommend having at least $10k to cover living expenses during bootcamp and for job searching afterwards. This should cover the essentials. It is certainly possible to get by with less cash, and I personally know of people who have done this. However, you may be cutting it close. Let's break down this number.

Rent: This will most likely be your biggest expense. Especially if your bootcamp is located in California or New York. If you have the opportunity to live at your school you should definitely take it. Not only will you save money, you will learn more and make better friends. Don't expect to have your own place unless you're willing to shell out at $2k+/mo. I was able to score my own room in downtown for $750/mo. It was a 10x10 dorm with just a bed, chair, and desk. Honestly, that's all you need during bootcamp. $750/mo is on the cheaper end. Expect to pay around $1000/mo for room and transportation to class.

Food: This all depends on the individual. You probably won't have time to cook so expect to eat out. I usually ate two full meals a day for $10 a piece. Roughly $600/mo.

Gym: Buy yourself a membership if you don't have one. This is especially true if you live at the school and it doesn't have a shower. Most of our class had a membership. It's a great way to relieve stress. Much cheaper than going to the bar too. Roughly $40/mo.

You want to budget for 3 months of bootcamp and another 3 months afterwards to find a job/get your first pay check. Rent + Transportation + Food + Gym comes out to $1640/mo or $9840 for 6 months. Let's round this up to $10k to be ultra-conservative.

If you can cut down on rent that will save you the most. But don't forget you will also need a couple thousand for first month's rent and deposit if you need to relocate. Hopefully by then you have received your signing bonus, that should help.

$10k should provide you enough cushion for those six months. Oh and if $10k + tuition seems like a big investment to you, I assure you, it's well worth it. Especially if you consider what you will be making afterwards.

What a Typical Day is Like

Here's what a typical day for me was like:

- **9:00am**: Morning lecture begins, lasts about one hour. Discusses assigned reading from previous night and introduces concepts for today's challenges
- **10:00am**: Break into pairs and begin working on programs
- **12:15pm**: 75 minute lunch break
- **1:30pm**: Regroup with my partner and continue coding
- **3:15pm**: 15 minute break to stretch our legs and get coffee. God knows we need it
- **3:30pm**: Continue coding
- **5:00pm**: End-of-day lecture. Lasts about an hour. Reviews most challenging assignments of the day and discusses solutions.
- **6:00pm**: Continue working on problems if I didn't finish everything
- **7:00pm**: Grab dinner
- **9:00pm**: Spend an additional 1-3 hours finishing assignments and doing reading.

Depending on the load, each day consisted of 6-9 hours of code. When you account for sleep, eating, breaks, lecture, and reading, you'll see you spend all your remaining time coding. Welcome to programming bootcamp.

Best Time to Attend Bootcamp

The best time to attend bootcamp is the time that will be the most convenient and distraction free for you. With that aside, if you are flexible on your start date and want to get a job afterwards you should pick a time that graduates you during the busy hiring season. Hiring season is busiest in spring and fall. Thus, attending bootcamp during summer or winter is ideal.

Luckily, hiring is busy for tech year-round. Nonetheless, companies close books at the end of the year. There is significant slow-down around Thanksgiving until the end of the year. As a result, you may have a harder time finding a job if your 9-12 week bootcamp starts in fall.

It certainly won't be impossible to find a job if you attend a fall session. I know several people who have and are working at great companies today. However, it did take them a more time on average to find jobs. Regardless, most were settled by the end of January.

What Kind of People Bootcamps Want

Bootcamps want problem solvers and effective communicators. You don't already need to know how to code to be successful for bootcamp. Those skills can be taught over the course of the class. But strong problem solving and clear communication skills cannot be taught in such short a time. These are things you must have developed on your own.

Do you like solving puzzles or answering brain-teasers? Do you refuse to give up when you're stuck on a tough challenge? If this sounds like you then a bootcamp probably wants you.

Are you personable? Can you clearly communicate with others? Do you generally get along with other people? If yes then you're also desirable. People don't like working with robots. It will be harder to find a job if you're unable to present yourself in a friendly and professional manner. Bootcamps know this. They want employable students. Not only that, but they prefer students who have the potential to rise in the ranks. If you're communication skills are good they may make up for weaker technical knowledge. That being said, no amount of charm will work if you're totally clueless.

You have a better chance to get accepted if you have good problem-solving and poor communication skills then the other way around. But you don't have to be a genius either to get accepted.

How to Choose the Right Bootcamp

Not all bootcamps are created equal. If you're going to throw down $20k for tuition and devote three months of your time you should choose your bootcamp carefully. How do you know which bootcamps are legit? Here are a couple of indicators.

Curriculum: A solid bootcamp will clearly lay out their curriculum. Reach out to the program if you can't find it online. Red flags should raise in your head if they can't provide a well organized week-by-week plan. There should also be a focus on projects, with time allocated for completing a large project. An even better program will also allocate time for interview prep. Keep in mind these are bare minimums. The best programs will go over several topics and cover advanced concepts and popular libraries.

Admission Process: A good bootcamp will have a non-trivial admission process. I know of some schools that don't even do a coding challenge. You should seriously consider what kind of students are likely to attend these programs. You want a program that accepts candidates who know the fundamentals. Those programs will have more motivated students. You want to be around those people. The coursework will be more intense and you will be pushed harder. Ultimately, this will increase your chances of getting a job afterwards.

Reviews: Find out what alums think of the program! Scour Quora and Yelp. The best programs will have students genuinely gushing about their experience. I wrote my first Quora post after I graduated. I had an amazing experience and was happy to write a review. I was not alone either, several students offered their feedback – both good and bad.

Find where alums work: Go to LinkedIn and search for people who attended the bootcamp you are interested in. Look when they graduated and see where they are working now. Alums of good programs will usually start working within three months of their graduation date. Red flags should be raised if this isn't the case for at least 3 out of 4 alums you find.

Reach out to alums: You'll be surprised how open people are to sharing their experiences. I emailed five people I found on LinkedIn and got immediate responses from three. They answered several of my questions and made me feel more assured with my decision to attend. This may help tip the scales if you find yourself in the fortunate position of being able to choose between programs.

You don't have to do all or even any of these things. But choosing a good program will significantly increase your chances of getting a job.

In-Person vs Remote

Nowadays you have the option of attending an in-person or remote bootcamp. Both have their pros and cons. Full disclosure, I attended App Academy[7], an in-person bootcamp, so my opinion is biased. Nonetheless, I emphatically believe if you're going to do a bootcamp you should only do an in-person one. Here are some pros and cons of both:

Pros of In-Person:

- There will physically be other students and instructors in the same room as you. This is huge. It will keep you disciplined, honest, and motivated. It will allow you to have ad-hoc discussions with your peers and instructors. It will allow you to watch how your instructors write and debug code. These are important and cannot be adequately replicated in a remote program. There may be other pros but in my mind this far outweighs anything else. Including all the "cons" of a in-person program.

Pros of Remote:

- It's cheaper. This can be a big deal if you can't afford attending in-person.
- You don't have to relocate. Again, for those who can't afford relocating a remote program is sounds attractive.

Cons of In-Person:

[7] http://www.appacademy.io/

- It's more expensive. $20k + living expenses for 3-6 months is no joke.
- You have to re-locate. Although depending on your outlook, you may consider this a benefit.

Cons of Remote:

- You won't have peers that are easily accessible. Sure you can chat and maybe even screen share but that's not the same as collaborating in real-life. Professional developers have developed tools and practices to make remote pairing work. It's still a hassle. You don't have time to learn that workflow. You need to focus on learning material.
- Limited instructor access. Similar with the above, Skype and chat is not the same as having an instructor present. It will be harder to share your code and discuss your bugs remotely. Furthermore, not being able to watch your instructors debug your code is a major loss. This is an invaluable skill that you want to develop as a junior engineer.

So which program should you choose? It should be obvious I'm inclined to say in-person. The cheaper price tag of a remote program may make it look attractive. But as the saying goes, you get what you pay for. The one scenario in which remote may be better is if your only desire it to learn programming for learning's sake and not get a job afterwards. But in that case I strongly urge you to consider free online bootcamp alternatives like Odin Project[8].

An in-person bootcamp is the way to go if you're serious about working as a software developer or gaining technical chops to prototype your start-up.

[8] http://www.theodinproject.com/

Dealing with Rejection

Rejection sucks. Especially after you've studied so hard. But don't be discouraged. Rejection shouldn't stop you from coding. In fact, one guy who was rejected from App Academy went on to build a popular website[9] comparing bootcamps.

Ask yourself why you were rejected. Reach out to the bootcamp if you honestly don't know. Otherwise improve on those weak spots and consider applying to another bootcamp. There is more than good program out there.

Find out how to re-apply if you're determined to attend after getting rejected. One of my friends was initially rejected by Hack Reactor. He contacted the founder and re-applied after a few months. He got in.

As a software engineer you will repeatedly experience failure. There will be bugs in your code. You will fail to ship on time. It happens. Don't let it get you down. You won't progress if you give up. But if you push forward and continue to improve, you will have a better chance of getting accepted next time.

[9] http://www.bootcamps.in

Closing Remarks

I hope you found this guide useful. At a minimum I hope you are encouraged to apply to a bootcamp. They are a great way to fast track into the tech industry.

Getting accepted to a top-tier bootcamp has become more competitive. Regardless, the admission process is still straight-forward. I've summarized the steps below:

The initial application: This requires no prep. Just fill out the application and hit submit! Be honest and clear about your intentions for attending.

The coding challenege: This will test your basic programming and problem solving skills. Before attempting the coding challenge make sure you:

- Complete practice problems!
- Use the resources recommend by the bootcamp.
- If none are provided, see the *Additional Resources* chapter at the end of the guide
- Learn to use the terminal and know how to test your code

After completing and testing several practice problems, go ahead and take the programming challenge!

The Skype interview: This step also tests your programming/problem solving skills as well as your ability to communicate your thought process

- Practice harder problems (see *Additional Resources*)
- Practice verbalizing your thought process
- If you can, grab a friend and do problems on a white board!

I encourage you to immediately schedule the Skype interview after being invited. In the meantime keep practicing. During the interview never do more than two things at once (see my earlier chapter for reference).

Complete pre-coursework: You may receive a conditional acceptance contingent on completing some additional coursework. If you're serious about attending finish this early. You want to hit the ground running.

Acceptance: Hooray, this is what you wanted! Look up reviews and reach out to alums before sending that deposit or tuition. Bootcamp is expensive and you're investing a lot of personal time – make sure it's worth it!

Rejection: It sucks, but don't give up! Identify your weaknesses and apply again. Keep working hard and you will do better next time.

Finally, make sure to take some time off and recharge before starting class. Once bootcamp begins you won't have as much time to take it easy :)

One More Story

I want to share a few words about my thoughts on software engineering. At my job, I was fortunate enough to be mentored by one our best engineers. He taught me much. Most importantly, he showed me what it means to be a great software engineer.

He was great because he cared. Not just about the code, but about everything. He made sure he knew why something had to be done. He read the documentation for every library we used. He wrote an in-depth README for each code-base he maintained. He created detailed tickets explaining the work to be done. He put forth more effort (not necessarily more time) than any other engineer in our company.

A good engineer views their work as a craft. They are willing to invest time in learning and improving the trade. That's what you want to do. Attending a bootcamp is an early step in your programming journey. The real adventure begins after graduation. If you want to be one of the best, keep working hard and never stop learning. Good luck!

Additional Resources

Looking for a textbook or additional practice problems to help you prepare? Here's a short list of resources that will prepare you during the application process. All are available online completely free!

Textbooks: These introductory textbooks are good for absolute beginners who are looking to learn basic programming concepts and syntax. I've provided one for Ruby, Python, and JavaScript.

- Ruby: Learn to Program (Chris Pine)[10]
- Python: Learn Python The Hard Way (Zed Shaw)[11]
- Ruby: Eloquent JavaScript (Marijn Haverbeke)[12]

Practice Problems: These exercises will test your knowledge of basic programming concepts and test your problem solving skills:

- RubyMonk[13]: For easy problems. The site is ruby-focused but you can solve the questions in any language on your local machine.
- Coderbyte[14]: For interview-style questions, these range from easy to difficult
- Reddit[15]: This thread contains easy, medium, and difficult questions. The easy questions are similar to what you may get in the initial challenge. The Skype interview will be near the medium level. The hard questions are very difficult, you won't be asked anything like that.

[10] https://pine.fm/LearnToProgram/
[11] http://www.learnpythonthehardway.org/
[12] http://eloquentjavascript.net/
[13] https://rubymonk.com/
[14] http://www.coderbyte.com/
[15] http://www.reddit.com/r/learnprogramming

Feedback

Thank you for purchasing this guide. I truly hope you found it useful. This guide is still in an early phase so I actively encourage my readers to provide feedback. Please feel free to reach out for any of the following reasons:

- You have a question or concern that I did not address
- You have a suggestion for improving the guide
- You noticed a typo or grammatical error
- You want to say thanks (I appreciate these)

You can contact me at joe@codecampguide.com

Cheers!

– Joe