Poptip Summer Internship
‘Take Home Assignment’

The following set of problems is intended to give you an opportunity to show off how you think, your programming skills, and what kind of things you’re passionate about. We’ve tried to line up an assortment of tasks and problems and hope this ‘assignment’ can be an opportunity to **do something different and fun**.

Do not attempt to complete everything. This is NOT a problem set. Most important is that you show off how you think and code - beyond that, feel free to keep coding and come up with something awesome or tackle other challenges if you’re having fun. We’ll give feedback on whatever you send our way.

We suggest you write up some sort of plan for how you would attack two challenges, and write code for one. This is the real world, not a closed book exam, so you may use any resources at your disposal (open source packages, stack overflow, textbooks) but avoid try to avoid googling the exact problem or using code you’ve already written for another project. Additionally, any code you write should be your own, and if you brainstorm with anyone else please document it.

Please **document how long you have spent working**. If you get busy and aren’t able to make as much progress as you’d like, don’t worry about it, we’ll assess based on time spent and what kind of creativity and understanding you demonstrate. Feel free to work at your leisure - don’t do it all in one sitting.

For all of the programming questions, we ask that you create a **git repository** (we’d recommend bitbucket so that the repo stays private, but feel free to use github) and commit early and often as you work on the solution. If you choose bitbucket, invite john@poptip.com to your repo. Don’t worry, we won’t check on things until you give us the go ahead.

Additionally, try to **use whatever coding practices you’re familiar with**. Write a readme and make it useful: if for you that means it is very formal, write it that way. If you prefer informal readmes and more detailed comments in code, write that way. Either way, please try to document the thought process behind the code.

Finally, **anything you create for this assignment is yours to keep**. We ask that you don’t show off anything you’ve built until mid April, as others may be working on the same challenge. After that, if you make something you’re proud of and want to put in your portfolio or open source, feel free.

Please turn in your assignment to john@poptip.com by 11:59pm EST a week after the date you receive it. If you have any questions or feedback, feel free to email me any time.

1. (Real-time) API Visualization

Choose an API and hack together a dynamic visualization of your choice. This could be display of all the data, some sort of aggregation of the data, or really anything else that you think would look cool or be useful.

If you just want to show off your design chops, feel free to do make something that has next to no functionality but looks really cool. If you want to make something super functional that allows you to glean cool insights from data, feel free to make it more functional than pretty.

Ideally, you'd work with a realtime API, but we know they're few and far between. There are a few in this list (twitter, instagram, foursquare, and some google feeds might be easiest), and a few data.gov APIs are available as realtime feeds as well. Alternatively, you can continue polling any API for new data at a regular rate - be sure to pick something that we won't have to sit and watch for a few hours in order to have your visualization display anything.

We recommend using javascript and keeping data ephemeral - we're not asking you to build a full stack application. But if you're enjoying things, feel free to incorporate a database, make it interactive, or host whatever you build on something like dotcloud, heroku, or appengine.

Alternatively, if you've got experience with Python and OpenCV, or want to make a Java Applet with Processing… use whatever language you'd most like to build something in.

Some recommended resources:

Visualization:
http://d3js.org/
http://www.chartjs.org/
http://www.highcharts.com/
http://www.creativebloq.com/design-tools/data-visualization-712402

API discovery:
https://apigee.com/console
https://www.mashape.com/explore/All?page=1&filter=popularity

2. Document Similarity

Using a programming language of your choice, build a program that computes the similarity between two or more ‘documents’.

Pick any type of document - it could be words, webpages, books, images, facebook friend graphs, songs. Use any level of rigour you want when comparing the documents - your algorithm could compute a similarity score or group documents into topics or categories, or even determine whether a small document is a component of a large document (think Shazam). You can make the algorithm super simple and efficient and index a large number of documents, or just compare two but do a very in depth analysis.

In a readme, describe your design decisions and why you think the decisions you made make your method a useful assessment of similarity.

For this problem, please leverage existing algorithms and packages: a good start is to check out the wikipedia page for document classification.

Some jumping off points:

Text:

- http://nlp.stanford.edu/software/
- http://www.nltk.org/

General IR and ML:

- http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html
- http://scikit-learn.org/stable/
- http://www.cs.waikato.ac.nz/ml/weka/

Images:

- http://en.wikipedia.org/wiki/List_of_CBIR_engines

Speech:

- http://en.wikipedia.org/wiki/Cepstrum

Audio:

- http://en.wikipedia.org/wiki/Acoustic_fingerprint

Golang

The following two problems give you the opportunity to program in Go (http://golang.org/). Most of Poptip's stack is written in Go, with JS for the web interface, so these should give you a taste of what the summer could be like.

Go is a statically typed language developed by Google in 2007 that has syntax loosely derived from C. Code is extremely readable due to clean syntax and the the small language footprint - the only loop type is a for loop. You should be able to pick up the basics of the language in a few hours through the official language tour.

We like go for several reasons:

- Concurrency primitives are built in, and proper use of these primitives abstracts away the complication that come with lock contention and other aspects of concurrent programming.
- The language does not include redundant and potentially confusing features like method and operator overloading, circular dependencies, pointer arithmetic, assertions, generics, and type inheritance.
- Compilation is fast, remote dependencies are managed, and documentation is automatic.
- The standard packages are awesome and include everything you really need to make a full stack web application - no need to rely on MVC, frameworks, or other things that make code confusing and inefficient.
- Even though the community is in its infancy, awesome things are being built. Check out the top repositories and all the cool companies using it.

In getting acquainted with go, we recommend that you check out the language tour, the language specification, and effective go, and feel free to reach out and ask questions.

3. Dining Philosophers (Concurrency in Go)

Utilize Go to build a solution to the Dining Philosophers Problem, the famous problem presented by Dijkstra and Hoare as an exam exercise in concurrent algorithm design. The problem is formulated as follows:

> There are *N* philosophers sitting around a circular table eating (an infinite supply of) rice and discussing philosophy. However, there are only *N* chopsticks, one between each philosopher. A philosopher needs two chopsticks in order to begin eating the rice, and a chopstick may only be held by one philosopher at a time. Assume that every philosopher will stop eating some time *X* after they have begun eating, where *X* is finite.
>
> Design an algorithm that the philosophers can follow such that no philosopher will starve that does not assume that any philosopher knows when the others want to think or eat.

If you have no experience with concurrency, feel free to look at solutions to the problem. If you want to attempt to solve the problem yourself, document your problem solving process: **coming up with a creative solution or a proof could be a viable alternative to actually implementing it.**

In implementing a solution, take advantage of Go's built-in concurrency primitives and include unit tests that check for any edge cases you can come up with.

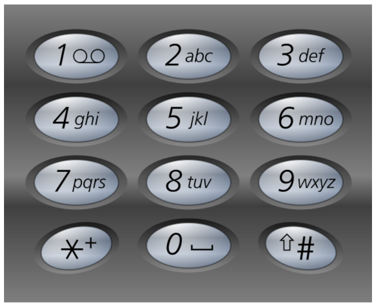
Resources for go concurrency:

http://talks.golang.org/2012/concurrency.slide#1
http://www.nada.kth.se/~snilsson/concurrency/

4. T9 in Go

This blurb from Andy, our head of product:

> In my day, before the age of smartphones, we had to type text messages using our numeric keypads like animals. T-9 was a predictive text algorithm to do this efficiently. The idea was to map the possible words that could be formed by typing a series of touch tone numbers (as you can see, each number is mapped to 3 or 4 characters), so to type “hello” you would type 43550.



Implement a version of T9 that takes in a number as a command line argument and returns all the possible words that can be formed using those digits.

You can use http://cl.ly/1L3d0A2W0T3h as the dictionary for your solution, but feel free to use a small subset of words. Just english is fine. Let us know what subset of the dictionary you used.

Some examples:

43550 -> ["hello"]
5263 -> ["lame", "lane"]

If you’re feeling ambitious, feel free to implement prefixing: return all words that start with this sequence of numbers. Alternatively, you could make T9 as a service - build an API that returns the possible results as JSON.

5. Natural Language Calculator

In a language of your choice create a command line program that has the functionality of a calculator. However, unlike a typical calculator, this calculator will take english strings as an input.

For example, all of the following should output 3:

- “one plus two”
- “1 + 2”
- “1 plus 2”

You may assume that each token is delimited by a space.

At a minimum, please implement (+,-,/,*) for digits 0-9. But the sky's the limit - we’d love to see solutions that can interpret “seven hundred and 33 point 2 six” correctly - even Wolfram|Alpha gets that wrong, or calculators that can support more complex operations, whether they be unit conversions, integrals, or whatever else you’d find useful in the command line. As another idea, figure out how you’d get around having space delimiting: how could you interpret thirtyfivepoint3?

The following two challenges are a bit different and potentially even more open ended than the preceding challenges.

6. Site critique

Do a critique of the UI and UX of any site from http://www.alexa.com/topsites. Make at least three concrete suggestions.

If you're interested in design we'd love to see you mock up a design or a full user flow. Alternately, build a widget or some portion of the site functionality as you think it should be built.

7. Kaggle Challenge

Kaggle is a platform for predictive modeling and analytics competitions in which companies will post a problem with a dataset and data scientists compete to produce the best model.

Pick any competition from http://www.kaggle.com/ and build a solution or describe (in detail) how you would tackle the problem. **If you build a solution, we'd accept this as your only challenge completed.**

Keep in mind the limitations placed by the Kaggle ToS - if you submit a solution on the Kaggle platform and compete for the prize money, you may lose the intellectual property you have created. If you're confident about things, go ahead, but if not feel free to just share the source with us.