Google Forms

Welcome

This is a very long questionnaire so pease set aside at least 30 minutes to complete it. For each question, please select the option that best describes you. You can only select one.

General Experience

How many years of experience do you have? *

- 0
- [✓ Less than 1 year]
- 1 - 3
- 4 - 6
- 7 - 8
- 9 years or more

Select the option that best describes your ownership of tasks and projects *

- No ownership responsibility yet: you are learning and being actively developed by others.
- [✓ You own your own tasks with guidance]
- You own your own tasks without guidance
- You have a track record of task and small project ownership
- You have fully owned a project
- You have a consistent record of very strong ownership and have been a team lead

Rate how comfortable you are speaking in front of a group inside the company *

	1	2	3	4	5	
Extremely uncomfortable					✓	Extremely comfortable

Rate how comfortable you are speaking in front of a group OUTSIDE the company *

	1	2	3	4	5	
Extremely uncomfortable					✓	Extremely comfortable

How do you communicate? *

- You communicate status poorly, don't share anything with anyone, consider or internalize feedback, nor participate in any user groups, forums, or even twitterparticipate in any user groups, forums, or even twitter

- ☐ You can write a basic bug report and attend the occasional conference or meetup
- ☐ You give constructive feedback to peers
- ☐ You are comfortable offering constructive feedback to all levels, even clients, and write down processes and procedures to follow for other developers
- ☑ You have an excellent ability to communicate status to all interested parties, including when the news is bad. You are active in user groups and meetups, regularly peer review others' code, accept feedback and modify behavior based on the feedback, and give feedback
- ☐ You write an occasional blog post or other research summary, document code examples and communicate with non-technical team members to give technical advice
- ☐ You regularly summarize research and share, maybe via blog or gists, give talks at conferences and in Secret Source, and successfully communicate technical topics to a wide variety of audiences

Describe your approach to learning *

- ☐ You make no effort to learn anything
- ☐ You follow some technology leaders on Twitter, blogs, etc.
- ☑ You read books, subscribe to tech blogs, have done some online courses and know when to ask for help when you are becoming stuck; do not go down rabbit holes
- ☐ You publish work (code) seeking input / feedback from others
- ☐ You test new ideas by sharing code or ideas inside company, experiment (and document) results, write the occassional blog post, present regularly at local meetups and conferences
- ☐ You regularly write blog posts / participate in open source projects, actively mentor / teach others, and are committed to a self-study program

How much of a leader are you? *

- ☐ You aren't a leader, not even for yourself
- ☐ You are a good follower (implement the lead's ideas in practical ways)
- ☐ You document processes and procedures for the rest of the staff
- ☐ You are able to identify which developers are able to execute given tasks effectively and you seek input prior to making a decision
- ☐ You break things down into small steps and you are able to clarify expectations to find common ground
- ☐ You set reasonable and effective standards for how something should be done
- ☑ You pave the way for the next leaders, provide leadership opportunities for others, take risks making decisions, fail and get up and keep going, succeed and share the success with the rest of the team, lead by actions and not just by words, you have a track record of building successful teams

Technical Breadth and Depth

In the following sections, please select the option that most closely describes you. If you feel there is more than one, pick the one nearest the top of the list. For example, if there are 4 options listed from 1 - 4 and you feel you are between a 2 and a 3, select the 2.

This section is VERY long. Please take the time required to answer these questions thoughtfully.

Also, please remember you are answering these questions about yourself in spite of how they are expressed (third person).

Project Planning and Estimating

Estimating / Tracking *

Unable to guess or highly innaccurate guesses. No real estimating methodology. Unable to track time spent to time planned.

Can estimate very small, concrete tasks, but may (often) fail to recognize uncertainty inherent in some tasks.

[✓ Can estimate based on prior experience and might have an estimating methodology. Can track progress but will fail to track training, research, and email / slack as part of the project.]

Estimates based on a repeatable formula such as number of pages / widgets. Tracks time using a time tracking tool, however rudimentary. Estimates are normally within 20% of what was estimated.

Project Planning *

Unaware that planning is necessary.

Plans in head (not in writing) and often only plans the main part of a single file (very small plans).

Organizes activity in writing prior to starting a new project or series of tasks. Can specify the criteria for doing things in a specific order. Will often do what's easiest first, not what the client needs.

[✓ Same as intermediate and plans work in view, controller, model order but modifies order depending on system requirements and client needs.]

Development Methodologies *

Unaware they exist

Has heard of Agile but doesn't know what it is. Hasn't heard of waterfall.

Has heard of Agile and maybe used it. Has heard of waterfall and maybe used it.

[✓ Has experience working in an Agile team and can self-organize.]

DevOps / Sysadmin

Version Control *

Never used

Knows what it is, might have used, but never on a team

Has used on a team, is comfortable working on branches, has resolved a conflict

[✓ Uses advanced features of version control, including branches and rebasing, totally comfortable resolving conflicts, uses pull requests regularly]

Development Environment *

Doesn't use one

[✓ Uses MAMP et al]

Uses Homestead or similar

Uses homestead or similar and has customized to optimize development activity. May also use Docker.

SSH, Bash, command line *

- [✓ Knows a few commands]
- Knows basic commands and can log in via ssh
- Regularly works on the command line and has written some supporting scripts. Can manage keys without support.
- Regularly works on the command line and often writes basic to intermediate bash scripts to support all activity. Uses advanced bash features such as set -e (Bash Strict Mode) and logging to improve script resiliance.

Server Configuration *

- [✓ No experience]
- Has used cPanel or similar to create an email account or install a software package.
- Knows how to start and stop services from the command line. Has configured a virtual host in apache or nginx.
- Has created user accounts with custom skeletons. Has compiled PHP and/or installed PHP extensions using PECL, apt install, etc. Understands file permissions, can configure PHP to run as a CGI and as a specific user. Understands firewalls, proxies (forward and reverse), can configure one or both from the command line.

Common Web Stacks *

- [✓ Only knows about Apache or nginx but not both. Only knows about node.js but not apache, etc.]
- Same as Trainee but has heard of both apache and nginx and has modified .htaccess files.
- Same as other but has also used PHP's built-in server for development.
- Can configure virtual hosts and pre-processors for a variety of languages on apache, nginx and node.js.

HTTP *

- Does not know any of the codes by heart but does know the conecpt that a command with params is sent to a server and the server responds
- Knows what a 200 and 404 code mean. Has seen 500 codes but is unaware of the meaning.
- [✓ Knows the meaning of the main codes but has never written an API. Knows some of the common headers sent and received.]
- Knows the meaning of the main response codes by heart. Has written an API using response codes to communicate meaning. Knows the common headers sent and received by heart. Is a power-user of Postman / cURL.

DNS *

- Has heard the term but is unaware of how it works.
- Is aware of how it works but has had little or no experience configuring DNS
- [✓ Has configured DNS for a domain using a domain hosting control panel. Knows what the main DNS records are used for. Has used dig or similar to help set up DNS.]
- Has configured an internal DNS server, including defining a forwarder. Knows what TXT records are and what they are commonly used for. Can efficiently debug DNS using dig or similar.

Web Development

HTML *

○ Knows the basic elements but doesn't know much about semantic HTML or the meaning of most elements. Doesn't understand the DOCTYPE or its purpose. Doesn't know which elements can contain other elements nor what type. Doesn't know where to get the definitive documentation (and if they do, they don't read it).

○ Knows how to validate HTML using one of the popular validators. Knows what a minimum HTML5 document must consist of (and why). Makes overuse of data-attributes to compensate for lack of knowledge of other element attributes. Knows where the standard is defined and might look things up from time to time. Sometimes writes invalid HTML.

○ Writes mostly valid HTML. Actively reads the documentation to keep up to speed on changes to HTML. Uses some of the new features in HTML5 but still relies on JavaScript to polyfill for most features. Knows exactly what the DOCTYPE is and what it is used for. Knows what semantic HTML is and makes an effort to write it but may still make overuse of wrapper DIVs.

[✓ Is a master of HTML (HTML5 in particular). Fully understands the DOCTYPE declaration and its purpose. Knows how to write semantic HTML and what elements can contain other elements. Knows tricks for implementing new HTML5 elements in browsers that don't yet support them. Does not use "data-" attributes where exisitng attributes would suffice (does not make overuse of data- attributes). Writes only valid HTML.]

CSS *

○ Knows the basics of CSS (selectors, scope, inheritance) but does not know the purpose of media or feature queries. Does not use advanced features of the LINK element (adding the media query as LINK attribute). Makes overuse of classes to compensate for a lack of understanding of the cascade.

○ Knows how to use some advanced CSS features to progressively enhance a design (calc, for example). May rely heavily on CSS libraries for advanced layouts. Makes some use of Flexbox but does not fully understand topics such as axes. Still does not make full use of the cascade (as evidenced by overuse of font declarations, for example).

[✓ Plans how CSS will be written prior to writing it. Has a full understanding of most of the features that can be used by modern browsers. Knows where to look for CSS documentation. Writes mobile first CSS.]

○ Is a master of CSS. Fully understands the cascade and takes full advantage of it. Has used and is knowledgeable in CSS architectures such as BEM. Writes all solutions as mobile first. Knows what render blocking is and how to avoid it. Uses advanced CSS features such as feature queries. Knows when to use Flexbox vs. Grid. Understands the CSS working group release structure (Level 1, Level 1, etc.).

JavaScript *

○ Uses jQuery for nearly everything. Does not have an in-depth understanding of the DOM or prototypes, or the JavaScript language in general.

○ Relies less on jQuery but still does not have a strong command of plain JavaScript. Writes synchonous code by default. May rely on JavaScript rather than using progressive enhancement. Does not know where to look for the definitive source of JavaScript documentation.

[✓ Uses jQuery only when necessary. Writes mostly asynchornous code but is aware of promises. Has a good understanding of the DOM, event binding, and has an awareness of the bubble (but may not be able to articulate it).]

○ Is a JavaScript master. Fully understands the DOM, the bubble, how to bind to events, scope, etc. Can write ES6 (or whatever the most recent version is) from memory. Uses asynchronous code whenever possible to minimize render blocking.

Server-side (PHP, node, C#, etc.) *

[✓ Can write basic procedural-style code. Has a vague understanding of how server-side code operates, of how servers operate. Could not set up a server-side processor from scratch.]

Can write basic OO code. Has a clear understanding of what a web server is and how servers work (binding to ports, handing requests off to preprocessors, etc.). Knows what htacces and nginx equivalent is. Understands basic server permissions in the context of a web server.

Everything prior plus has had some experience writing out of process code and working with jobs (cron) in conjunction with activity on the server.

Has a full understanding of how server-side code is executed. Knows about the most common HTTP servers, roughly how they work, and has had experience configuring them (setting up virtual hosts, writing htaccess files, etc.). Writes all server-side code in Object Oriented fashion (even JavaScript). Is concious of the potential for running out of memory and writes code out of process when necessary to minimize blocking.

General Language Knowledge

Variables *

Knows of only one way to instantiate. Has a vague understanding of scope. Does not understand the implications of type nor the difference between loosely typed and strictly typed languages.

[✓ Understands and can plan for differences in scope or type (but not both).]

Knows more than one way to instantiate variables. Understands the subtle difference in meaning between emtpy() and isset() in PHP (for example). Can plan for differences in scope and type.

As a standard coding practice, initializes all variables with default values prior to use. Uses scope and type to her advantage.

Constants *

Is unable to differentiate between a variable and a constant.

Knows how to set a constant and generally how they are used.

Knows what types of values can be stored in a constant.

[✓ Understands the memory and speed implications of using constants and knows when to use them.]

Classes *

Has heard of OOP

Has written a basic class with methods and properties.

[✓ Has written a suite of classes that are in production and completely modular (can be included in any project).]

Has defined a complete API and implemented it via a class-based solution. Understands what getters and setters are and when to use them. Knows the different levels of visibility and when to use them (private, protected, public). Is familiar with and implements common OOP patterns such as the singleton and factory patterns (and others). Has written both interfaces and abstract classes for implementation by others. Can explain what the constructur and destructor are in PHP and at least one more language and when to use both. Knows what PSR-4 is in PHP and related standards in at least 2 other languages

Functions *

- Knows what functions are and how to write one.
- Familiar with defining parameters with default values.
- Can write a recursive function that doesn't end in an infinite loop.
- [✓ Writes functions with accurate names that makes sense in a variety of contexts.]

Namespaces *

- [✓ No idea what namespaces are]
- Has seen namespaces but no idea what they are
- Knows what they are and can figure out how to use them.
- Doesn't write code without them.

Expressions *

- Unaware of the consequences of type in expressions. Unaware of the peculiarities of type juggling in dynamic languages (PHP) and order of precedence in dynamic languages.
- [✓ Knows some of the common pitfalls of working with expressions in a dynamically typed language (unexpected errors and exceptions due to type mismatch) but not necessarilly how to avoid them.]
- Employs strategies for dealing with potential type mismatches in expressions (such as type hinting and casting). Fequently uses expression shorthand in code where it makes sense.
- Can recite how dynamically typed languages will handle type juggling from memory, including order of preference when mixing types in a single expression. Knows all the common and some uncommon expression shorthands.

Operators *

- Knows the most basic operators (math, simple cancatenation)
- Uses operator shorthand whenever it makes sense.
- [✓ Knows the difference between "logical" and bitwise operators e.g.: the difference between 'and' and && and in a related fashion, between == and ===]
- Knows the advanced operators, including those that operate on objects and other unusual types, how dynamic languages handle precedence in all situations (including when different types are being operated on) and shorthand versions of all operators.

Types *

- Knows, at most, basic types (integer, string, array)
- Knows some advanced types, e.g.: float, object, exception
- Understand the subtleties of types and precedence in a dynamically typed language (PHP, Ruby) and how to work around some of the peculiarities (math fails in unexpected ways when one of the parameters is a string).
- [✓ Knows all types of their primary languages by memory and what happens when one type is cast to another (how types are transformed when cast from an object to an integer, for example)]

Control structures *

- Knows the most basic control structures (if/else, for loops, switch) and only one form (if() vs. if:)
- Knows additional control structures, a few ways of looping, for example, and has seen an alternative syntax (where it exists).

[✓ Knows the majority of control structures and their variant forms should they have some. Might not be able to defend the choice of for vs. foreach.]

Knows a vast array of control structures, including seldom used constrol structures and their shorthand versions from memory. Knows the common pitfalls of potentially problematic control structures of their main languages, eg. in PHP a foreach loop copies memory thus requiring more whereas a standard for loop simply inspects values on each iteration.

Scope *

Vaguely aware of the concept but completely unaware of the implications

Knows why a local variable is not available in the global scope and won't try to access it.

[✓ Makes a habit of defining global variables and constants in code initialization routines.]

Has total control of scope and rarely needs to check the documentation. Understands the risks and drawbacks of putting everything in the global scope. Actively manages scope to control resource usage.

Error Handling *

[✓ Knows try/catch and perhaps one other method of handling errors and exceptions]

Knows how to hide errors from the browser via configuration.

Has some experience enabling error and exception handlers but does not make a habit of it.

Uses advanced error handling techniques including custom error and exception handlers that log all the necessary information to be able to debug any issue. Is able to implement site-wide "pretty" exception handler so the site doesn't ever appear to go down.

Documenting Code *

[✓ Comments the obvious]

Comments everything

Comments when necessary but doesn't use a commenting system and is inconsistent. Doesn't document apis, for example.

Writes detailed docblock or similar comments with every parameter and return value fully defined.

Object Oriented Programming

Fundamentals (what is an object…) *

Can write a basic object and read object oriented code

[✓ Writes OO code that uses inheritance, albeit incorrectly.]

Writes OO code that uses inheritance correctly, knows how to write abstract classes and interfaces and how (and when) to use them. Knows what dependency injection is but may not always use it. Understands fundamentals of writing decoupled code but struggles with doing it.

Only uses OO Design patterns, duck typing, dependency injection, and other OO design techniques (decorators, mutators, form validation dependencies, etc.)

Common Patterns (singleton, factory, and null in particular)