Contents

[***Professional Communications*** 2](#_Toc29204268)

[Personal and Business Life (V1.0.0) 4](#_Toc29204269)

[Inappropriate Language (V1.0.0) 9](#_Toc29204270)

[***Coding Principles in JavaScript*** 14](#_Toc29204271)

[Syntax (V1.1.1) 15](#_Toc29204272)

[Variables (V1.1.0) 16](#_Toc29204273)

[Datatypes (V1.1.0) 20](#_Toc29204274)

[Conditional Statements (V1.0.0) 23](#_Toc29204275)

[Functions (V1.0.0) 27](#_Toc29204276)

[***Interactive/Online Media*** 30](#_Toc29204277)

[REST API (pt. 1) 31](#_Toc29204278)

[***Computer Engineering/Science*** 33](#_Toc29204279)

[***Appendix A: Vocabulary*** 34](#_Toc29204280)

[String 39](#_Toc29204281)

[Input/Output 40](#_Toc29204282)

# Professional Communications

#### Company Culture

#### Time and Energy

The only truly finite resources humans possess are time and energy. Anything else can be found or created using time and energy. Indeed, having a job is simply selling your time and energy to create everything else humans need!

This is something you must always be cognizant of in both the workplace and your life. Not only is efficient use of your time and energy vital to your success but being aware of the cost your actions have on others’ time and energy will help your entire team succeed.

Your teammates can not accomplish work if they are spending all their time reading your long, pointless emails. If you are pessimistic, non-constructively critical, hold grudges, spread rumors, and are generally negative, you will find your team will start using their energy handling their work environment rather than working.

##### Speed vs Substance

#### Personal and Business Life

#### Interpersonal Boundaries

#### Resolving Conflicts

## Personal and Business Life (V1.0.0)

##### Relationships

The word “**relationship**” is loaded with implications in our culture. It typically implies that someone is romantically involved with another. The literal definition, however, is simply the way that two or more things affect each other. Specifically, how they *relate* to each other. Although there are infinite ways to categorize relationships, we will focus on only two concepts. Learning how to mix and separate these will lead you to success in a career.

##### Personal and Business Relationships

Personal relationships come in quite a variety. They can be family, friends, fellow hobbyists, and romantic partners. In a personal relationship, two or more people are typically giving and receiving emotional effects. Be it support through tough times, sharing in the enjoyment of a mutual hobby, or just companionship.

Business relationships, in contrast, involve exchanging goods and labor. An important distinction between personal and business relationships is that *all* business relationships are either *negotiating* or *fulfilling* **the contract**. Sometimes, the **contract** isn’t a “real” thing. Sometimes, it’s just an agreement.

A job interview is simply a *negotiation* between a worker and a company to create a **contract**. The company will give you an agreed upon amount of money in return for your time, labor and skills. Having a job is simply the *fulfillment* of that **contract**. For as long as you provide your work according to your **contract**, the company will continue to give you money. Most job **contracts** come with many implied expectations in our culture, but almost every company has an “Employee Handbook” in case your ever have any questions.

##### Mixing Personal and Business Life

The answer to “Should I mix personal with business?” is almost always “No”. Of course, there are exceptions to every rule, and humans and culture are nuanced. Many business **contracts** have been negotiated over a meal or a game of golf. Developing ways to identify when personal and business are being mixed, and evaluating whether they should or not, is a great tool to have in your skillset.

##### Personal/Business Time

Most job “**contracts**” expect you to be working and focusing on work during your agreed upon work hours. Mr. Smith’s job **contract** is to provide education to students for a few hours a day. Your **contract** as a student is to give a few hours a day to the company (school) in exchange for education and marketable skills. Both **contracts** involve using time, but neither can be fulfilled if the time is spent on YouTube or cell phones.

##### Personal/Business Property

Legally, most companies can “own” physical things. Although things where you work may not belong to “anyone”, they are needed by the people in the company to make money and pay you. You should not take, modify or abuse things that belong to your company. They belong to the company.

In our culture, companies are expected to provide the tools and safety equipment for your labor. If they do not, you have *leverage* in your **contract** negotiations if you can provide your own. Companies have no right to use your things. They belong to you.

##### Personal/Business Relationships

* It’s not abnormal to develop friendships with the people you work with. Remember, nobody *has* to be your friend. If you do make friends, make sure you don’t get distracted by personal stuff at work.
* Trying to date your coworkers is usually a very bad idea for a multitude of reasons. Someone could feel very uncomfortable if they have to reject you and harassed if they have to do it more than once. If your work with your partner, a romantic relationship going good or bad can be distracting at work.
* A “superior” (manager) dating a “subordinate” (employee they manage) is very taboo in our culture. Almost all companies require you to report to Human Resources if you are romantically involved with a superior, and many prohibit it outright.

##### Theft

Taking physical objects from the company is an obvious way to steal. Less obvious is when someone doesn’t hold up their end of a **contract**. If you feel that you are not being paid enough at your job, end the **contract** or renegotiate your pay. Do not do less work or steal capital to “get even”. “Time-clock theft” is counted by most companies’ Loss Prevention department. To them, taking care of personal matters on company time (being on your cell phone at work), or manipulating the time clock is considered stealing. The law tends to agree.

### Personal and Business Life: Comprehension

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Describe what you think your “**contract**” (business relationship) with the school is:   |  | | --- | |  | |  | |  | |  | |  | |  |  |  | | --- | |  | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Can you think of a time Mr. Smith didn’t respect *or isn’t respecting* company property?   |  | | --- | |  | |  | |  | |  | |  | |  |  |  | | --- | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| What is something your or classmates do that uses up school time on non-school things?   |  | | --- | |  | |  | |  | |  | |  | |  | |

### Personal and Business Life: Analyze

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| If you work in a store, and you bring in things from home that the store happens to sell, what would you bosses think if they saw it?   |  | | --- | |  | |  | |  | |  | |  | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Could streaming entertainment on a school/company computer or network be considered time-theft, theft, or misuse of school/company property?   |  | | --- | |  | |  | |  | |  | |  | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| What do you think customers think of you when they see you on your phone at work? What do you think they think of the company?   |  | | --- | |  | |  | |  | |  | |  | |  | |

### Personal and Business Life: Evaluate

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| If your mouse at school is poor quality and uncomfortable to use, but you have a nice gamer mouse at home. Should you bring it in to use at school? Why or why not?   |  | | --- | |  | |  | |  | |  | |  | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| You work with your boyfriend/girlfriend. They haven’t been responding to your texts for two days. You will see them when they come to work four hours into your eight-hour shift. When and how should you confront them, and why?   |  | | --- | |  | |  | |  | |  | |  | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| You see someone working at their retail job that you’re interested in. You want to talk to them and maybe ask them out. Should you do it now? Why or why not?   |  | | --- | |  | |  | |  | |  | |  | |  | |

### Personal and Business Life: Anticipation Guide (V1.0.0)

**Before Class**: In the *left column*, put a ✔️ if you think the sentence is **true**. Otherwise, leave it blank.  
**After Class**: In the *right column*, put a ✔️ if you think the sentence is **true**. Otherwise, leave it blank.

|  |  |  |
| --- | --- | --- |
|  | I can check my phone at work if I’m not doing anything else anyways. |  |
|  | The best people to work with are my friends. |  |
|  | It’s hard to work with people who wouldn’t talk to you outside of work. |  |
|  | Nobody would have a problem if Mr. Smith brought in their VR from home. |  |
|  | It is ok to take things home if the company never uses them. |  |
|  | It is ok to take things out of the trash from work. |  |
|  | I can take something home from work if I bring it back tomorrow. |  |
|  | The company is using my personal equipment, so I can take something of equal value home. |  |
|  | It is ok to use a VPN to access my computer at home on my company’s network. |  |
|  | It’s ok to plug my phone into my work computer to charge or access photos. |  |
|  | I should just use my company phone for Pokemon Go if it runs better. |  |
|  | If I need more energy, I should take a bit more time than my break to leave and get coffee. |  |
|  | I can nibble on food if I work in a kitchen. |  |
|  | I should use company tools roughly if it gets my job done faster. |  |
|  | I can take things from other company departments if mine doesn’t have what I need. |  |
|  | My coffee is fresher than what the company buys, so I can brew my coffee for everyone in the company coffee machine. |  |
|  | It’s ok to leave my things in the classroom overnight. I’ll be back first block anyways. |  |
|  | A customer’s was order wrong at work and their food was replaced. I can eat the old food. |  |
|  | If I don’t have a personal email address, or don’t want to give out mine, I can use my company email address. |  |
|  | I don’t have my work email set up on my phone. I can just give people I’m doing business with my personal email instead so I can get back to them faster. |  |

## Inappropriate Language (V1.0.0)

Simply speaking, no words are inherently inappropriate. Any word can be inappropriate if used in the wrong way. As a rule, you are taught to *never* use “curse words”, but human language is more complex than that. “Swear” words tend to amplify or embellish ideas that are already complete. Amplifying an idea that is poorly developed or disruptive can make a situation become much more negative than it needs to be.

##### Disruptive Language

The primary goal of a work setting is for individuals to cooperate in a team to complete large, complex, and challenging tasks. Communication is pivotal to this. Using any language that disrupts this communication or accomplishing the primary goal is considered inappropriate. It can be as concrete as saying a “bad” word or using generally offensive or coarse language. It can also be as ambiguous as creating discussions on divisive and irrelevant topics – such as politics – or using sarcasm, gestures, and negative tones.

##### Choosing Language for the Situation

There are an infinite number of scenarios you could be in, but here are a few you may want to consider now. When you find yourself in these situations, it’s best to focus on your words and tone carefully. Speaking in a clam, serious, and respectful manner using clear and concise language is always safe, and you should master it early.

* **Familiarity:** As you work with your coworkers, you start to learn their unique flavor of communication. It is also not uncommon to become personal friends with some coworkers. In these situations, language that is normally inappropriate ceases to be disruptive, and can sometimes even become conducive. However, this **familiarity** is a process that must be built over time and can be very foreign to outsiders. It is not safe to assume new colleagues and customers will not “take you joke the wrong way” just because your friend coworker did not. It is also not safe to assume that your language isn’t upsetting to others because they are not part of the conversation.
* **Criticism:** It’s very difficult to hear that you are wrong, and it’s equally difficult to communicate to someone else that they are wrong with good results. **Criticism** is vital to fixing problems in a project, so finding the right way to tell someone is important to master. Using passive aggressive or sarcastic tones can be highly inappropriate, even if you don’t curse or your criticism in constructive. “Sugarcoating” criticism too much, on the other hand, may cause the receiver to think the problem they are being presented with is not important.
* **Confrontation: Confrontation** can be even more difficult to handle than **criticism**. Regular **criticism** may address a problem created in the work, but **confrontation** seeks to address problems created in the *workplace*. Most people do not want to upset others and are often simply unaware they may be doing it. **Confronting** someone when a **conflict** is still small in a calm, serious tone gives your coworker an opportunity to negotiate and come to a solution you’re both happy with. Waiting until the problem becomes too frustrating may result in you using abrasive or upsetting language, which may leave your coworker feeling ambushed and defensive. It is also important to set **boundaries**. If you find someone’s language or behavior towards you disruptive or offensive, it’s important that they know you do not want them to continue. A calm, serious tone is important to make the coworker understand that are not “the bad guy”, but that they can not continue the behavior towards you. *If someone’s behavior or language towards you ever makes you feel too unsafe to confront them directly, always seek help from your coworkers*.
* **Bologna-Sandwiching:** It’s a common occurrence to find yourself collectively getting distracted with your coworkers and talking about things that are not in any way productive. These are the times when it’s very easy to get comfortable saying things inappropriate for the workplace. Be careful with your idle words.
* **Empathizing:** It can sometimes be very difficult to see things from your coworker’s perspectives, especially if you are already in a confrontation with them or presenting criticism. Sometimes, you may want to ease tension by using foul or facetious (joking) language, but this can come across as cavalier, or not taking them seriously.
* **Gossiping: Gossiping** or “spreading rumors” is how humans learn about each other as social creatures. It is very easy and natural for humans to do but can be highly disruptive in the workplace. Details of a story can lose their fidelity each time they are told (think telephone game), resulting in spreading misinformation about people. It can also result in a lot of resentment of coworkers in the company. What’s worse is the person being gossiped about has no chance to defend themselves or change their behavior.

### Inappropriate Language: Anticipation Guide (V1.0.0)

**Before Class**: In the *left column*, put a ✔️ if you think the sentence is **true**. Otherwise, leave it blank.  
**After Class**: In the *right column*, put a ✔️ if you think the sentence is **true**. Otherwise, leave it blank.

*[Tricky Tricky!] There are no right and wrong answers here, but these questions are more complex than they appear. Please take your time to analyze each one.*

|  |  |  |
| --- | --- | --- |
|  | Swearing is *never* OK in the workplace. |  |
|  | It is *always* OK to swear around your friends. |  |
|  | You shouldn’t tell your coworkers about little frustrations they cause you. |  |
|  | Swearing is an effective way to let others know you are serious. |  |
|  | Mr. Smith always uses appropriate language in the classroom. |  |
|  | You can be more disruptive and upsetting to others without swearing. |  |
|  | People will like you less if you swear. |  |
|  | Swearing is an effective way to let others know you are not serious. |  |
|  | Being serious or unserious at the wrong time is more offensive than swearing. |  |
|  | It isn’t inappropriate to talk a lot about a hobby at work if all your coworkers enjoy that hobby. |  |
|  | If a coworker is doing something that annoys you, you should vent to other coworkers. |  |
|  | Sometimes, coworkers just need to be told how awful their work is. |  |
|  | It’s ok for coworkers to tell you about their personal problems at work. |  |
|  | If your coworker says something that upsets you, you should tell your manager right away. |  |
|  | Other people will know it’s ok to curse at your coworker because you are friends. |  |
|  | Using sarcasm will lighten the mood when you tell your coworker their work can improve. |  |
|  | It’s ok to curse loudly in frustrating situations. People around you know it’s not their fault. |  |
|  | Most people are too uptight about curse words. |  |
|  | People find it difficult to take criticism and will even defend bad decisions. |  |
|  | You should confront your coworkers about their habits once they are too much to tolerate. |  |

### Inappropriate Language: Comprehension

|  |
| --- |
| What is the difference between **criticism** and **confrontation**, in your own words?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|  |
| --- |
| Describe a time you’ve seen **gossip** disrupt work or school:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|  |
| --- |
| Describe a time you’ve seen **gossip** spread good news or keep people safe:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|  |
| --- |
| Who is someone you think it’s ok to swear in front of or directly at? Why do you think it’s ok?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

### Inappropriate Language: Analysis (V1.0.0)

*[There are no wrong answers. Do your best!]*

|  |
| --- |
| Your coworker is very jovial and loves joking around with everyone. One day, they sneak up behind you and tickle your waist. You like this coworker, but you *really* did not like being tickled. For each of the following, write if you think the response is appropriate or not, and why you think that.   * *Giggle and say, “Stop it!”*   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   * *Simply say, “Please don’t do that.”*   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   * *Tell them to “Get the @#$% off me!”*   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|  |
| --- |
| You are working the cash register at a store and there are a few customers in line. The other register is being run by your friend and coworker, and you notice their customer is being short with them. The last time you saw this customer, they were rude to you, also. As soon as the customer leaves, you want to tell your coworker all the “choice words” you have to describe that customer. You know your friend would appreciate the empathy. Do you tell them? Why or why not?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

### Inappropriate Language: Evaluate (V1.0.0)

|  |
| --- |
| “You can avoid **conflicts** by never avoiding **confrontation**.”  What do you think this means?  Does it only apply to ‘arguments’ or ‘fights’?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|  |
| --- |
| How does empathy amongst coworkers make the company as a whole run more efficiently?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

# Coding Principles in JavaScript

## Syntax (V1.1.1)

JS is an interpreted language. All code is handled at runtime. This means JS executes each line one by one as it reads through the list of code.

##### Statement

A single instruction in Javascript.

##### Expression

An expression is any valid unit of code that evaluated and resolves to a value. They are most commonly simple Algebra equations, such as ‘(3 \* 2) + 7 ‘, which resolves to 13. Other times they are true/false conditions, such as ‘3 > 2’, which the computer resolves as ‘true’.

##### Assignment Operators

Symbols used to assign a value or expression on the right side of the operator to a variable or object on the left side of the operator. The most common operator is ‘=’, but you can combine it with **arithmetic operators** to manipulate a value already in a variable on the left side.

|  |  |  |  |
| --- | --- | --- | --- |
| = | Assign variable/object to this value. | | |
| += | Add this value to the current value | -= | Subtract this value by the current value |
| \*= | Multiply current value by this value | \*\*= | Multiply current value to this value’s power |
| /= | Divide current value by this value | %= | Remainder of the current value divided by this value |
| ++ | Add one to the current value (increment) | -- | Subtract one from the current value (decrement) |

##### Arithmetic Operators

Symbols used in **expressions** to perform mathematical operations. They function just like Algebra. The order of operations is even the same.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | Addition | - | Subtraction | \*\* | Exponents |
| \* | Multiplication | / | Division | % | Modulus (Remainder) |
| () | Parenthesis (expressions inside of parenthesis get evaluated and resolve first) | | | | |

##### Comparison Operators

Used in **expressions** and **conditions** to compare one value or **expression** to another.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| == | Equal to | <= | Less than or equal to | < | Less than |
| != | Not equal to | >= | Greater than or equal to | > | Greater than |

#### String Operators

Used to manipulation strings.

**+** , **+=**

##### Keywords

A predefined word in Javascript that has a specific function and cannot be used for a variable, function, object, or class name.

Some keywords include break, continue, debugger, do, while, for, function, if, else, return, switch, try, catch, var, and let.

## Variables (V1.1.0)

*An object that stores a single value in memory that can be changed.*

##### Explanation

Variables function just like variables in Algebra. They are symbols (or in this case, words) that represent either an unknown value, or a value that could change depending on the program’s state.

##### Syntax

First, use the ‘let’ keyword to declare a variable. This gets done once in the program. You cannot use a variable that has not been declared. When you declare a variable, you don’t have to give it a value right away. You do have to give it a good name. One that explains what the variable’s purpose is in as few letters as possible. You should also always use **camelCase**.

|  |
| --- |
| const and var You can also declare variables with var and const. const creates a variable whose value cannot be changed. |

After a variable has been declared, you can change its value by ‘assigning’ it a value. This is done with an **assignment operator**. The most basic and frequently used assignment operator is ‘=’. The statement looks like this:

keyword variableName assignment operator expression or value semicolon

Which looks like this in practice:

let applesInBasket = 13;

##### Substitution

As your program encounters variables, it will substitute the variable name with its value that is stored in memory at that moment. This is exactly like substituting the value in Algebra!

##### Manipulation

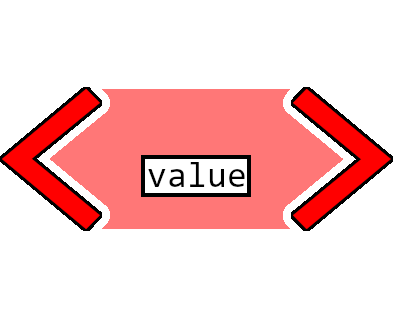
Changing a variable looks a lot like Algebra. Just like ‘x = (y \* 2) + 7’ is a valid equation in Algebra, so too is it valid in Javascript (albeit for slightly different reasons). In addition to **assignment operators**, you can create **expressions** with **arithmetic operators** to create new values for the variable to store. **Expressions** even follow the same order of operations as Algebra. PEMDAS!

Other **assignment operators** combine an **arithmetic operator** with the ‘=’ operator to provide fast and easy to read way of changing variable values. For example, while we could type out this statement:

applesInBasket = applesInBasket + 3; // Add three apples to the basket

It’s much faster to use the ‘+=’ **assignment operator**:

applesInBasket += 3; // Add three apples to the basket

Some variables can be ‘strings’ (see **data types**). Adding them together is called **concatenation**. This is done by using the **string operator** ‘+’ or the **assignment operator** ‘+=’.

#### Shapes

In this class, we will identify all variables (and indeed, object properties) with   
“ < > ”, as per our ***Shapes Definitions*** guide.

### Variables: Knowledge (V1.1.1)

Go through these code blocks and identify each variable by putting the “ < > ” shape markers around it. You don’t have to understand the program. Just identify the variables. For example, change:

let sayHelloWorld = sayHello + sayWorld ; // sayHelloWorld now equals ‘Hello World’

…into:

let < sayHelloWorld > = < sayHello > + < sayWorld >; // sayHelloWorld now equals ‘Hello World’

… to identify ‘sayHelloWorld’, ‘sayHello’, and ‘sayWorld’ as variables.

function sayHelloWorld ( iterant ) {  
  
 let phrase = ‘Hello World! ‘ ;

let loopCounter = 0 ;  
  
 while ( loopCounter < iterant ) {  
  
 console.log ( phrase ) ;  
  
 loopCounter += 1;  
  
 }  
}

function updateBar ( player , hpsp , min , max ) {

let calculated = ( ( min / max ) \* 100);

if ( calculated > 100 ) {

calculated = 100;

} else if (calculated < 0) {

calculated = 0;

}

}

for ( let i = 0 ; i < 10 ; i += 1 ; ) {  
   
 console.log ( ‘We have looped ‘ + i + ‘ times!’ ) ;  
   
}

### Variables: Comprehension (V1.1.1)

For each of these code blocks, replace the variables with the value they hold at the place in the code where they appear. You do not have to replace variables that are being declared or are on the left side of an **assignment operator**:

let applesPerBasket = 12;

let baskets = 3;

let totalApples;

console.log('You have ' + baskets + ' baskets');

\_\_\_\_\_\_\_\_\_\_\_

console.log('They have ' + applesPerBasket + ' each');

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

totalApples = baskets \* applesPerBasket;

\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

console.log('You have ' + totalApples + ' in all');

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

let roaches = 16;  
let maurauders = 12;  
if ( mauraders > roaches ) {

\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_

roaches -= ( mauraders / 2 ); // Two hits to kill a roach  
  
  
 \_\_\_\_\_\_\_\_\_

console.log( roaches + ' roaches remain!');  
  
  
 \_\_\_\_\_\_\_\_

} else {  
 rageQuit( 'terran' );  
}

let dingdong = 7;  
let wingding = 5.5;  
let herpderp = ((dingdong \* 3) + wingding) / (dingdong - wingding);

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_  
console.log( herpderp );  
  
  
 \_\_\_\_\_\_\_\_

### Variables: Application (V1.1.1)

On the lines below, write a program that does the following:

* Declare a variable called *name* and assign the value ‘Bob’ to it.
* Declare another variable called *bobsPigeons* and assign the value 3 to it.
* Make *bobsPigeons* equal its current value multiplied by 3.
* Make *name* equal its current value plus the string ‘s Pigeons’ at the end.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Declare a variable called *soon*. Do not give it a value.
* Declare a variable called *thirdNum*. Do not give it a value.
* Declare a variable called *firstNum* and assign it the value 4.
* Declare a variable called *secondNum* and assign it the value 7.
* Change *thirdNum* to equal the expression ‘*firstNum* multiplied by 3, then subtract *secondNum’*
* Change the value of *soon* to ‘I will have my revenge’.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Datatypes (V1.1.0)

##### **Types** **of** **Values**

There are two types of values. Variable values (can change), and fixed values (never change). We already know variable values as simply **variables**. Fixed values are known as **literals**. Their names and their values are the same, making them literally what they say they are. This means typing 10 in code is the number ten, and typing ‘Hello, World!’ in code is literally the human phrase ‘Hello, World!’ and not any variable named *Hello* or *World*.

##### **Data** **Types**

Every value has a data type. The five data types of Javascript are numbers, strings, booleans, arrays, and objects. A variable with no value has an undefined value. A value that doesn’t exist is null, but Javascript counts null as an object.

Javascript stores data in formats that cannot be directly compared. For example, human-readable words are not numbers, so you cannot multiply or divide a number by a letter or word. Sometimes, Javascript can resolve these mismatches on its own. For example, if you attempt to concatenate a number onto a string, it will convert the number to a string before concatenating.

Unlike other languages, variables do not need to stay the data type they were declared as. Assigning a new data type to them overwrites the old data type. You can check what data type a variable is by using the typeof keyword. It will return the data type like a function.

##### Numbers

The most obvious data type is a number. Unlike other programming languages, Javascript stores all real numbers as one type, whether positive, negative, with or without a decimal place.

##### Strings

A string is a list of human-readable letters that are not code. Strings are surrounded by **single quotes** ( ‘ ‘ ), **double quotes** ( “ “ ), or **backticks** ( ` ` ). Strings can be **concatenated**, which adds one string onto the end of another. Concatenating another value onto a string will turn the result into a string, also.

##### Booleans

true or false. Also resolves as 1 and 0. It is more accurate to say that it is ‘not false’ and ‘false’, as any value other than false or 0 is counted as true.

##### Arrays

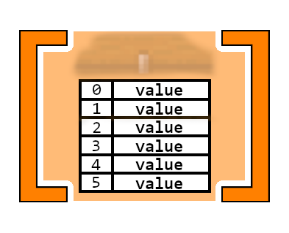
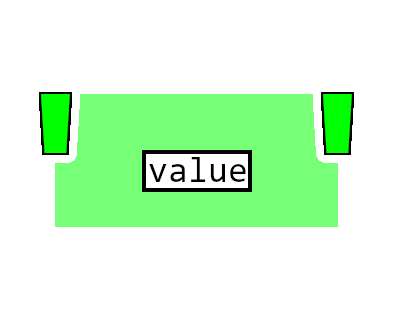
Arrays are a list of values, separated by commas, contained in square brackets ( [ ] ). Unlike other languages, each value in an array in Javascript can be a different datatype. Individual values can be accessed by their **index**. Think of it like a page number, or address.

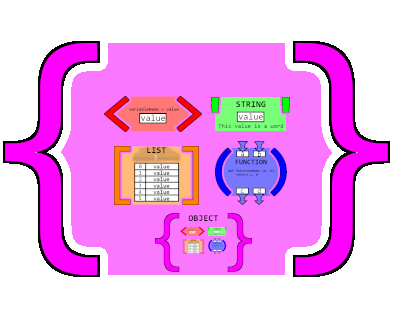
##### Objects

Technically, everything in Javascript is an object. Ultimately, an object is a variable whose value is more variables and functions (properties and methods). They can be treated as individual items, widgets, or indeed – objects in your code that have all kinds of features.

##### Shapes

We will need three more shapes from our **Shapes Definitions** guide, though we will only start with one extensively. We can mark **literal strings** by wrapping them in single quotes and underlining them “ ‘ \_\_ ' “. *We are only marking literal strings, because they are frequently confused with variables and* functions. Later we will use **arrays** (otherwise known as ‘List’), and **objects**.





### Datatypes: Knowledge (V1.1.1)

For this exercise, we will focus on **variables**, **literal strings**, and **literal numbers**.

* Wrap all **variables** in “ < > ”
* Wrap all **literal strings** in “ ‘ \_\_ ' “
* Circle all **literal numbers**.

let roaches = 16;  
  
let maurauders = 12;  
  
if ( mauraders > roaches ) {

roaches -= ( mauraders / 2 ); // Two hits to kill a roach

console.log( roaches + ' roaches remain!' );

} else {

rageQuit( 'terran' );

}

let applesPerBasket = 12;

let baskets = 3;

let totalApples;

console.log('You have ' + baskets + ' baskets');

console.log('They have ' + applesPerBasket + ' each');

totalApples = baskets \* applesPerBasket;

console.log('You have ' + totalApples + ' in all');

function sayHelloWorld ( iterant ) {  
  
 let phrase = ‘Hello World!‘ ;

let loopCounter = 0 ;  
  
 while ( loopCounter < iterant ) {  
  
 console.log ( phrase ) ;  
  
 loopCounter += 1 ;  
  
 }  
}

### Datatypes: Comprehension (V1.1.1)

At the end of these programs, write the console output:

let apples = 10;  
let urmumSize = ‘large’;  
let customerList = [‘Bob’, ‘Earl’, ‘Frank’];  
let sizeApples = urmumSize + apples;  
let bananas = ‘12’;  
console.log(typeof apples);  
console.log(typeof urmumSize);  
console.log(typeof customerList);  
console.log(typeof sizeApples);  
console.log(typeof customerList[1]);  
console.log(typeof bananas);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Datatypes: Application (V1.1.1)

On the lines below, write a program that does the following:

* Declare a variable that is assigned a literal string as the value
* Declare a variable that is assigned a literal number as the value
* Declare a variable that is assigned an **expression** that resolves as a number
* Declare a variable that is assigned an **expression** that resolves as a string

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Conditional Statements (V1.0.0)

An important principle of programming is “flow control”. A program is a lot like a movie script that can listen to the audience and change the outcome of the movie accordingly. Conditional statements change the “flow” of the program according to changes in information, like user input.

##### If/Then/Else

The “classic” conditional statement is the **If**/**Then**/**Else** statement. The syntax is different in every language, but the function is always the same:

if this expression resolves as true, then do this; else do that.

The important parts of this are the **keywords** “**if**” and “**else**”, the **condition**, and code to be run if the **condition** is true or false. In Javascript, it looks like this:

if (this experession is true) {  
 then do this;  
} else {  
 do that;  
}

Using an “**if**/**else**” statement, we can “fork” our code into a “**true**” path and “**false**” path. We can create a “side quest” by only using “**if**” and no “**else**”.

do this;  
if (this experession is true) {  
 then also do this;  
}

do that;

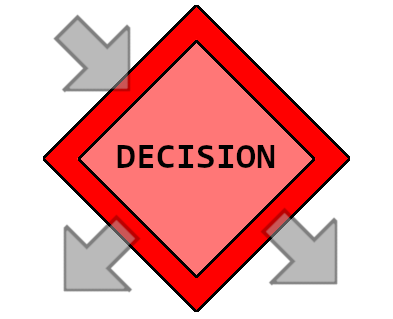
We can also create more than two paths using an “**else if**” keyword. If the first **condition** is not true, then it will check the second **condition**. If none of the provided conditions are true, it will run the **Else** statement. There can be as many “**else if**” statements as you want.

if (this experession is true) {  
 then do this;  
} else if (this other expression is true) {  
 then do this other thing;  
} else {  
 do that;  
}

##### Flow Control and Shapes

In **flow charts**, a **decision step** is represented by a diamond. Like all other types of flow chart steps, it can take as many inputs as you want. Unlike all other steps, however, it is the only step that that can have any number of outputs.

While **decision steps** are just a vague description of a general “decision”, you can always break decision steps down into smaller series of steps. You can choose to closely reflect your code in the flow chart, which is what online tools such as **code2flow** seek to do. This will create patterns of shapes that you will frequently see while writing flow charts, creating a visual link to the abstract thought of forks in code.



### Conditional Statements: Knowledge

For each of these code blocks:

* Circle each keyword.
* Wrap each **condition** in ‘ **< >** ‘.
* Underline any code that will run if the condition is **true** or **false**.

socks.owner = ‘Jimmy’;

socks.putOnUser(‘Fred’);  
  
if (socks.user != socks.owner) {

socks.userMood = ‘Not great’;

} else {

shoes.insert(socks);

}

if (temperature > 60 && weather != ‘rain’) {

vehicle = ‘Wrangler’;

} else if (weather == ‘snow’) {

vehicle = ‘Cherokee’;

} else {

vehicle = ‘tC’;

}

if (student.arrivalTime > class.startTime) {

student.tardy++;

if (student.tardy > 3) {

student.tardy = 0;

student.detention++;

student.detention();

} else {

student.tardyWarning();

}

}

### Conditional Statements: Comprehension

At the end of each of these programs, write the console output that would be displayed if the programs were run.

trainer.gymBadges = 7;

if (trainer.gymBadges < 8) {

console.log(‘Get out of here, scrub!’);

} else {

console.log(‘Welcome to the Indigo Platuea!’);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

vegeta.scouter.horizontalFlip = true;

kakarot.powerLevel = 9001;

console.log(‘Vegeta, what does the Scouter say about his power level?’);  
  
if (kakarot.powerLevel > 9000) {

if (vegeta.scouter.horizontalFlip == true) {

console.log(‘It\’s 1006. Kick his butt!’);

} else {

console.log(‘It\’s over nine-thousaaand!’);  
 vegeta.destroy(this.scouter);

}

} else {

console.log(‘Hardly worth our time!’);

vegeta.destroy(kakarot);  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

userNumber = 71;  
randomNumber = 54;  
  
if (randomNumber == userNumber) {

console.log(‘You guessed correctly!’);

} else if (randomNumber > userNumber) {

console.log(‘You guessed too low’);  
} else {

console.log(‘You guessed too high!’);  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Conditional Statements: Application (V1.0.1)

For each of the following word problems, write a program below it that solves the problem. You will need to pick good variable names to describe variables in the word problems.

/\* If the user’s age is less than 21, then check to see if the the user’s age is less than 12. If it is, tell the user they’re too young for Slurp Juice. Otherwise, tell the user they’re too

young for Krunk Juice.

\*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\* If the user’s name if ‘Frank’, tell the user “Welcome back, Frank!”. Otherwise, if the user’s name is ‘Bill’, ask the user “Where’s my money, Bill?”. Otherwise, tell the user “I don’t talk to strangers.”.

\*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\* If the user’s wool color is blue, tell the user “Blue Wool”. Otherwise, if the user’s wool color is red, tell the user “Red Wool”. Otherwise, if the user’s wool color is green, tell the user “Green Wool”. Otherwise, tell they user “I can’t see that color”

\*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Functions (V1.0.0)

When you take a step, your brain sends commands to a list of muscles in your body in a precise order. Which muscles it sends commands to depends on the direction of stepping. We can tell our brain to “take a step forward”, and our brain will take care of all the commands needed to accomplish that task for you. We can describe this is code like this:

takeStep( ‘forward’ );

‘takeStep’ is the action we want to perform, and ‘forward’ is a modifier or input to that action. This are what we call the **function name** and **function arguments**.

**Functions** are repeatable lists of statements that a program can jump to. They are also known as subroutines (considering that you program is a ‘routine’). Code that contains the list of statements a **function** performs is called a **function declaration**. Code that tells the program to jump to a function is called a **function call**. **Functions** take inputs, called **arguments**. All **function declaration** and **calls** are immediately followed by “ **( )** “, which contain the **function arguments**, if any.

// Function Declaration Example

function functionName ( arguments, separated, by, commas ) {

console.log(‘The first argument\’s value was: ‘ + arguments);

return value;

}

**Functions** have four components to consider when **declaring** one.

|  |  |  |
| --- | --- | --- |
| Function Name There are many ways to give a function a name.  The classic way is to use the **function** keyword:  function takeStep() {};  Because variables and functions are also objects, we can create an object that is a function:  let takeStep = function() {};  In Javascript, we can also use **arrow functions**: (We will discuss **arrow** **functions** more later, as they are usually used to create **functions** without a **function** **name**)  let takeStep = () => {}; Function Arguments Function arguments are temporary objects that are created when the **function** is **called** and given values in the **function** **call** itself.  function takeStep( direction ) {  console.log(‘Taking a step ‘ + direction);  } Function Statements Simply put, this is where your repeatable code goes. | |  | | --- | | Shapes Using our Shapes Definitions Guide, we can mark function declarations and calls using parenthesis “ **( )** “. | |
| |  | | --- | | Callback Functions Functions can be passed as arguments in Javascript. Many functions have an argument for **callback** functions, which are code to be execute once the rest of the function has finished. | |

##### Return Statement

When a function reaches a **return statement**, it ends the function. It then *returns* to the point in the code where the function was **called** and replaces the **function call** with the **return value**. The return value is an **expression** after the **return** keyword.

function sayHelloName ( name ) { // Function is named ‘sayHelloName’, and it takes a ‘name’ input

return ‘Hello, ‘ + name; // Sends the value of <‘Hello, ‘ + name> back to the function call

}

console.log(sayHelloName(‘Bob’)); // This is the function call for ‘sayHelloName()’

// When it returns, it will replace sayHelloName(‘Bob’)

// with the return value (‘Hello, Bob’)

### Functions: Knowledge

For each of the code blocks below:

* Put each **function** **name** and the **arguments** that follow inside of “ **( )** “
  + Remember, function names always have (arguments) after them
* Circle **function declarations**
* Underline **function calls**
  + Some things might be underlined multiple times

function sayHello ( name ) {

console.log ( ‘Hello, ‘ + name ) ;

}

sayHello ( ‘Bob’ ) ;

let addFunc = function ( a , b ) {

return a + b;

}

console.log ( ‘Five plus three is ‘ + addFunc(5, 3) ) ;

let firstNum = 5.5 ;

let secondNum = 3.2 ;

let newNumber = ( 1stn, 2ndn ) => {

return Math.round ( addFunc ( 1stn, 2ndn ) ) ;

}

console.log ( ‘Your rounded answer was ‘ + newNumber ( firstNum , secondNum ) ) ;

function sayThenDo ( callback ) {

console.log( “I’m going to do the callback function at the end of this function” );

callback();

}

sayThenDo ( () => { console.log ( “We declared this function as an argument!” ) );

### Functions: Comprehension

For each of the code blocks below, write the console output:

function sayHello (name) {

console.log (‘Hello, ‘ + name);

}

sayHello (‘Bob’) ;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

let addFunc = function (a , b) {

return a + b;

}

console.log (‘Five plus three is ‘ + addFunc(5, 3)) ;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

let firstNum = 5.5 ;

let secondNum = 3.2 ;

let newNumber = (1stn, 2ndn) => {

return Math.round(addFunc(1stn, 2ndn));

}

console.log (‘Your rounded answer was ‘ + newNumber(firstNum, secondNum));

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

function sayThenDo(callback) {

console.log(“I’m going to do the callback function at the end of this function”);

callback();

}

sayThenDo (() => {console.log(“We declared this function as an argument!”));

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Functions: Application

Write a program in the lines beneath each prompt:

/\* Write a function that takes one argument. Make the function log the argument to the console.

Then call the function and pass it an argument.

\*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\* Write a function that takes two numbers as arguments. If the first number is greater than the second number, return true. Otherwise, return false. Then call the function with two numbers as the argument for console.log()

\*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\* Write a function that logs “Returning true” to the console and returns true. Then call the function in the condition of an if statement. Inside of the if statement, log “Function returned true” in the console.

\*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Arrays/Lists (V1.0.0)

Functions (V1.0.0)

Loops (V1.0.0)

Objects (V1.0.0)

Scopes and Hierarchies (V1.0.0)

Callbacks (V1.0.0)

# Interactive/Online Media

## REST API (pt. 1)

An **API** is an *Application Programming Interface*. They are designed by a developer to allow other developers to interact easily with their application, without ever exposing the source code of the application.

A **REST API** is a *Representational State Transfer*, which is a set of rules a developer follows as they make their API. Once you’ve learned how to use a **REST API** for one application, it’s easy to learn how to use them for other applications, too!

##### HTTP Methods

**REST APIs** rely heavily on **HTTP** (hyper-text transfer protocol). Typically, the user will send a **request**, and the application will send a **response**. How the application handles the **request** depends on the **HTTP method**. There are many types of **methods**, but we will focus on **GET** and **POST**. When you type a URL in your browser, it always sends a **GET request**. If you click a ‘Submit’ button on the page, it typically sends a **POST request**. This is because **GET** is used to *get* a **resource** from the application, whereas as **POST** is used to *send* information to the application.

By convention, most developers follow the **CRUD methodology**. This means while developing your API, you should allow your users to ***C****reate,* ***R****ead,* ***U****pdate, and* ***D****elete* **resources**.

##### Endpoints

An **endpoint** is a URL that when navigated to, will activate a command in the application depending on the **method** used. For example, if you type this into a browser:

https://api.github.com/users/csmith1188

… it will return information about the user ‘csmith1188’ from GitHub in **JSON** format. A **root endpoint** it is the point where all other **endpoints** build from. In this case, the **root endpoint** would be this:

https://api.github.com/

The structure of the URL that makes up the **endpoint** is known as the **path**. In other words, what “**path**” through the **REST API** of the application do you have to take to access specific **resources**? A **path :variable** is a part of the URL that can be changed by the user to access a specific **resource**. The root-endpoint/path/:path-variable of the previous example is as follows:

https://api.github.com/users/csmith1188

‘users/’ is the **path**, letting the application know we want to select from a list of users. ‘csmith1188’ is the specific user we want to look up. The application then finds the relevant information and sends it back in the **response**.

##### Query Parameters

Another way we can request information from an application is through **query parameters**. **HTTP** allows you to put extra variables into a URL to be read by the application you are connecting to. For example

https://api.github.com/repos/csmith1188/JSFighter-class-/issues?state=open&label=bug

Not only are we using the **path** to tell the application we want to access the issues in repos, and using **path variables** to tell the application which user and repo we want to access, but we are using **query parameters** to tell the application that we only want to look at issues whose state is open and are labelled as bugs.

The ‘**?**’ tells the application **query parameters** are coming. The syntax looks like ‘**parameterName=value**’, and each parameter is separated by ‘**&**’.

##### JSON Responses

After the client **requests** information from the server, the server will send a **response**. The response will contain the requested resource, which can be a file, an error code, or some data. Data plain text typically sent in **JSON** (JavaScript Object Notation) format, which looks like this:

{“userName”: “Bob”, “icon”: “icons/bob.png”}

This data uses syntax rules that make it easy for a computer to parse and convert to code.

### Concept Check & Anticipation Guide

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_ | REST API  Resource  REST Endpoint  Root Endpoint  HTTP Method  REST Path  Path :variable  Query parameters  HTTP Request  HTTP Response  CRUD | \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_  \_\_\_\_\_\_ | | Before Class: In the *left column*, mark each word with a symbol from the list below.   * ➕ if you know enough to teach someone else about this * ⚪ if you know a little bit about this, but could learn more * ➖ if you do not know anything about this  After Class: In the *right column*, mark each word with a symbol and see what has changed! |

#### Before Class:

In the *left column*, put a ✔️ if you think the sentence is **true**. Otherwise, leave it blank.

#### After Class:

In the *right column*, put a ✔️ if you think the sentence is **true**. Otherwise, leave it blank.

|  |  |  |
| --- | --- | --- |
|  | Path :variables are another way of saying query parameters |  |
|  | REST APIs are vital for making chat bots |  |
|  | CRUD is a method of cleaning out excess data from a request |  |
|  | An endpoint is last line of code run before the server sends a response |  |
|  | A GET request is used to retrieve data from a server over HTTP. |  |
|  | A POST request is to send data to a server over HTTP. |  |
|  | HTTP is how computers send web data over the internet. |  |
|  | The root path of a REST API is the top level of hierarchal categorization of API endpoints. |  |
|  | A ‘client’ is a computer requesting information. A ‘server’ is a computer that provides it. |  |
|  | JSON format is Javascript code sent over the internet. |  |
|  | A ‘resource’ is a piece of data or file stored on a server, which clients request access to. |  |
|  | Your company project uses a REST API. |  |

# Computer Engineering/Science

# Appendix A: Vocabulary

Code Color Guide

Keyword Object Function Number String Comments

### Variable

*An object that stores a single value that can be changed.*

This is used to store data to be manipulated later. It works like a variable in algebra.

Use the ‘let’ keyword to declare a variable. This gets done once in the program. You cannot use a variable that has not been declared.

let value = 10; // Declare value  
console.log(value); // Log the variable’s value to console

### Constant

*A variable whose value cannot be changed during runtime.*

They are useful for tweaking configurations during development.

Use the ‘const’ keyword to declare a constant in the same way as a variable.

const MAXLIMIT = 3; // Declares MAXLIMIT as a constant  
let counter = 4; // Declares a counter variable  
if (counter > MAXLIMIT) {  
 // Checks to see if counter is greater than MAXLIMIT  
 console.log(‘counter is greater than’ + MAXLIMIT);  
}

### Array

*A single variable that stores multiple values.*

Arrays are declared like a variable except all values are enclosed in square brackets, separated by commas. Elements (values) of an array are accessed by their index number. Index numbers start at 0.

let classmates = [‘Josh’, ‘Noah’, ‘Harry’]; // Declare array as a variable  
console.log(classmates[0]); // Prints the first value in the array  
console.log(classmates[2]); //Prints the third value in the array

### Statement

*An executable line of code.*

It is represented by a rectangle in a flow chart.

### Conditional Statement

*A line of code that executes when a given condition is NOT FALSE.*

This is used for “Program Control Flow”. It is represented by a diamond in a flow chart.

### “If” Statement

*A conditional statement that forks the program flow.*

Called by the keyword ‘if’, followed by a condition in parenthesis **()**, followed by a list of statements in curly braces **{}**.

Can also contain an ‘else if’ and ‘else’ branch. ‘else if’ requires a different condition. ‘else’ executes if no prior conditions were met.

Formerly known as an “if/then” statement and is verbalized as “if/then/else if/then/else”.

let name = ‘Isiah’;  
if (name == ‘Noah’) { // If name is Noah, execute statements in these curly braces  
 console.log(‘Hello, Noah! My neighbor.’);  
} else if (name == ‘Harry’) { // If name is not Noah, but IS Harry…  
 // Then say hello to Harry instead  
 console.log(‘Hello, Harry, who sits three rows away’);  
} else { //Else if no other previous conditions were met  
 // Then execute these statements  
 console.log(‘I have no idea who you are’);  
}

### while loop

*A conditional statement that loops through a series of statements if the condition is NOT FALSE.*

It is good for an unknown number of loops.

Called by the keyword ‘while’, followed by a condition in parenthesis **()**, followed by a list of statements in curly braces **{}**.

let looper = true;  
while (looper) {  
 console.log(‘Condition was NOT FALSE, so I did the loop.’);  
 looper = false; // Make looper false, so we won’t repeat loop  
}

### do while loop

*A while statement that runs the loop* ***before*** *checking the condition to run again.*

It is good for an unknown number of loops, where loop must run at least once.

Called by the keyword ‘do’, followed by a list of statements in curly braces **{}**. It is then followed by the ‘while’, followed by the condition.

let looper = false;  
do {  
 console.log(‘This gets logged once, even though the condition is false’);  
}  
while (looper);

### Functions

*A repeatable group of statements.*

Functions can take multiple arguments (parameters). Functions can also return a value as though it were a variable.

Functions are declared with the keyword ‘function’, followed by the function name, and parameter names in parenthesis (). Parameters are separated by commas. A list of statements to be executed go inside of curly braces {}. The return statement is declared with the keyword ‘return’.

function newFunction(argA, argB) {  
 console.log(argA);  
 console.log(argB);  
 let c = argA + argB;  
 return c;  
}  
console.log(newFunction(1, 2));

### Objects

*An object is an identifiable “thing” in Javascript. Everything in Javascript is technically an object.*

Objects have properties and methods. Properties are variables that belong to the object, whereas methods are functions that belong to the object.

Objects are written in JSON format (Javascript Object Notation). Everything inside of curly braces **{}** is part of the object. Objects are made up of smaller objects called attributes (objects, properties and methods), which are defined by an attribute name followed by a colon (**:**), and separated by commas (**,**).

You can access an object’s attributes by putting the object’s name and a period in front of the attribute name. Objects refer to their own scope with the ‘*this*’ keyword.

let Person = {  
 // This is a property attribute  
 name: ‘Bob’,  
 // This is a method attribute  
 sayName: function() {  
 console.log(‘My name is ‘ + this.name);  
 }  
}  
  
// Log the value of the ‘name’ property  
console.log(Person.name)  
// Call the function of the ‘sayName’ attribute  
Person.sayName()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Pocket Guide** | | ✔️ | | **Company Meeting**  (Hit as many as you can.) | Good notetaking |  | | Effective feedback to questions |  | | Taking initiative or volunteering for leadership |  | | Brainstorming participation |  | | Identifying problems and causes |  | | Proposing problem solutions |  | | Constructive Criticism |  | | Maximizing team members utilization |  | | Optimizing workflow |  | |  |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Variable  |  | | --- | |  | |  | |  | |
|  | String  |  | | --- | |  | |  | |  | |
|  | Array / List  |  | | --- | |  | |  | |  | |
|  | Function  |  | | --- | |  | |  | |  | |
|  | Object  |  | | --- | |  | |  | |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Decision  |  | | --- | |  | |  | |  | |
|  | Input/Output  |  | | --- | |  | |  | |  | |
|  | Process Step  |  | | --- | |  | |  | |  | |
|  | Jump  |  | | --- | |  | |  | |  | |