Хочу здесь поделиться своим видением того как наиболее эффективно учить язык JavaScript. Мое мнение основано только на моем опыте, это стоит учитывать, я не обучал никогда программированию других людей.

Относись к этим советам как дорожной карте, по которой можно ориентироваться, чтобы понимать куда можно двигаться и как, но двигайся туда куда ХОЧЕШЬ.
Все эти советы подойдут и для многих других языков программирования (C++, C#, PHP, Python, Java, …), за исключением может только функциональных языков, у меня в этой теме нет опыта.

Мой опыт: я профессиональный разработчик со стажем в лет 20 в разных областях, который всю жизнь обучался программированию самостоятельно без каких-либо курсов. Я имею большой опыт работы с C++, C#, Java, PHP и JavaScript. Моя основная специализация сейчас - это JavaScript и фронтенд разработка.

Теория

Я считаю, что наиболее быстро освоить язык программирования можно последовательно изучая его составные части, кирпичики из которых строится программа, четко отделяя при этом сам язык от применения языка к чему либо, от библиотек, фреймворков, операций с веб страницами, алгоритмов, архитектуры и прочего.

Последовательность изучения языка JavaScript должна быть примерно такой:

1. Данные и как они хранятся обрабатываются в компьютере
2. Простые типы данных, включая массивы
3. Арифметические и логические операции и выражения.
4. Основные конструкции языка: if, switch/case, for, while, и т.д.
5. Функции
6. Более сложные типы данных: объекты, словари, коллекции, …
7. Асинхронное программирование. Асинхронные функции.
8. ООП и Классы
9. Итераторы/Генераторы
10. …

Когда ищешь учебники по языку, оглавление там должно примерно соответствовать этой последовательности изучения. Таким образом у тебя перед глазами будет хотя бы план изучения языка и примерное понимание куда двигаться, что ты уже знаешь, что еще не знаешь. И можешь посмотреть какие в принципе есть возможности у языка, прочитав оглавление книги. Это конечно не значит что другие учебники нужно выбросить и не читать их.

Постепенно по ходу изучения самого языка можно изучать и его применения. Поскольку разных применений языка много, и не понятно чем в итоге человек будет заниматься, я не могу здесь дать никаких советов. Единственное универсальное для всех направлений JavaScript разработки - это сам язык JavaScript в чистом виде.

После JavaScript можно изучить еще и TypeScript. Во многих компаниях этот навык требуется, например в React разработке. TypeScript - просто добавляет разметку типов

данных к JavaScript, что увеличивает читаемость кода, упрощает разработку, устраняет многие ошибки уже в процессе написания программы. Это будет намного проще изучить, чем сам JavaScript.

Полная документация по языку JavaScript: https://developer.mozilla.org/

Практика

Изученный материал целесообразно закреплять решением простых коротких задач. Если задачи будут слишком сложные и долгие, то ты можешь не получить достаточно удовольствия. Задачи не должны быть и слишком простыми, чтобы они не стали скучными и рутинными.
Практика тоже должна быть последовательной как и теория. Вот бесплатные задачники по JavaScript:

- https://exercism.org/tracks/javascript
- https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/
- https://javascript.info/

Есть еще сайты типа CodeWars и CodeCombat, там есть разбиение по сложности, но они не для последовательного изучения, а для тренировки навыков программирования для тех кто уже знает язык.

Стиль у этих задачников один и тот же: ты пишешь в одном окне код, потом нажимаешь кнопку Test, запускается проверка твоего кода. По началу может быть трудно понять, особенно то почему все так сложно устроено, и может быть стоит посмотреть ролики на ютубе на эту тему, где все подробно разжуют в том числе и почему задачники устроены именно так. Эти знания пригодятся потом в реальной работе, т.к. в реальной работе часто используется такой подход к разработке.

Интерес и мотивация

Подпитывают мотивацию:

- Чувство красоты от языка, примерно такое же как чувство красоты от чистой не прикладной математики.
- Знания которые долгое время будут актуальны. Полученные знания по языку JavaScript и TypeScript вряд ли устареют в ближайшие десятки лет. JavaScript слишком уж прочно засел в разработке сайтов, потому что до сих пор это единственный язык, на котором можно непосредственно писать программы под браузеры.
  Знания полученные при изучении JavaScript помогут легче осваивать другие языки программирования, ведь в JavaScript есть и процедурное программирование, и объектно-ориентированное, и элементы функционального программирования. Все эти концепции во многих языках очень похожи.
- Решение мелких задач
- Желание провести эксперимент
- Желание заработать кучу денег
- Желание создать свой проект (целесообразно чтобы он был коротким и по силам)

- Мотивацию повышает успешное достижение цели. Если цели длинные и трудно реализуемые, то мотивация снижается. Если цели слишком простые, то их реализация превращается в скучную рутину. Нужно держаться середины во всем: в выборе учебников, курсов, задач, проектов, экспериментов и т.д.

Убивают мотивацию:

- Презрение. Язык JavaScript - говно. В JavaScript можно найти множество недостатков, это несомненно. Эти недостатки можно просто знать иметь это ввиду, не испытывая никаких негативных эмоций по этому поводу. И стоит еще помнить о том, что главное в оценке языка это не список его плюсов и минусов, а можешь ли ты быстро и просто решать сложные задачи используя этот язык в его области применения. Мой ответ - несомненно да.
- Перфекционизм
    - Желание доделать проект, изучить неинтересную тему, закончить скучный курс. На начальном этапе может быть множество недоделанных проектов или пропущенных тем, интерес к которым пропал и это очень хорошо. Это признак того, что перфекционизма в обучении не так много.
    - Желание читать учебник в строгой последовательности. Желание не читать другие учебники не пробовать другие курсы пока не закончишь текущие и многое многое другое. Единой колеи для всех нет.
    - Желание учить сложную тему даже когда мозг уже отказывается работать. Поначалу будет сложно и нужно чаще давать мозгу отдыхать, переваривать информацию. Отдых - очень важная часть обучения. Целесообразно отдыхать тогда когда хочется, столько сколько хочется и возвращаться к обучению когда хочется.
- Идеи-фикс. Если хочешь надолго или навсегда убить интерес к программированию, то идеи-фикс подходят для этого идеально. Они высосут из тебя всю энергию и весь интерес, а когда очнешься будет уже поздно.

Другие советы

- **Марафон.** Если хочешь быстро изучить язык и начать зарабатывать, то обучение программированию не должно превращаться в копание в песочнице иначе потратишь кучу времени вроде как увлекательно интересно с картинками, а в результате не будешь дотягивать даже до уровня новичка. Чтобы изучать программирование нужно напрягать мозги, давать им отдыхать, потом снова напрягать, с каждым разом становясь сильнее, поглощая больше материала, так же как люди качают мышцы. Мозги перестраиваются и тренируются не сразу. Изучение программирования - это марафон на очень длинную дистанцию. Этот марафон не обязательно превращать в самоизнасилование, напрягать мозги может быть приятно, а думать о том что ты движешься к намеченной цели еще приятнее.
- **Многое можно не учить.** Информации очень много, но не вся информация актуальна прямо сейчас, и конечно не всю нужно зубрить и заучивать досконально. Часто достаточно только помнить о возможностях языка, ставить в голове или в конспектах короткие ссылки, как метки на карте, чтобы при необходимости вспомнить что такая возможность у языка есть и изучить эту тему подробнее. Нет смысла загружать мозги тем, что прямо сейчас ты не используешь и не будешь использовать часто в будущем. Следование этому принципу поможет сильно сократить время обучения и разгрузить мозги. Хорошо запомнится в итоге то, что ты часто используешь на практике и это будет идеальным использованием своей памяти.

- **Цель и путь к цели.** Чтобы быстро изучить язык нужно также двигаться в правильном направлении. А чтобы двигаться в правильном направлении, нужно понимать какая конечная цель, какие промежуточные шаги к этой цели, должен быть хотя бы примерный план. Главной целью для новичка, на мой взгляд, должно быть устройство на работу стажером или Junior разработчиком, т.к. основное обучение программированию происходит в процессе реальной работы, особенно если на работе есть ментор. Зная об этой цели можно выработать примерный план ее реализации, узнать например какие конкретно навыки и знания требуются для большинства наиболее популярных вакансий.
- **Ментор.** Чтобы двигаться в правильном направлении так же нужно понимать на что стоит тратить время, а на что не обязательно. Новичку в этом разобраться невозможно, т.к. для того чтобы знать все эти нюансы уже нужен большой опыт программирования. Поэтому на начальном этапе пригодятся услуги ментора, человека который сам является опытным программистом и который может подсказать правильное направление, что и как делать, сделает ревью кода, укажет на ошибки, ответит на вопросы, и т.д. Это распространенная практика, многие компании нанимают менторов со стороны, чтобы они направляли новичков.
- **Английский язык.** Английский язык очень важен в программировании, но не потому что сами языки на английском, многие документации и книги на английском - это все мелочи. Главное - это уметь общаться с клиентами, заказчиками, менеджерами, другими программистами, которые не знают твоего родного языка. Этот навык значительно увеличит возможности поиска работы и при том намного более высокооплачиваемой работы.
- **ChatGPT 4** (https://chat.openai.com/?model=gpt-4) может поработать ментором для новичков. Не обязательно платить 20$ в месяц, достаточно получить платный доступ к API и платить только за использование. Чатом по API можно пользоваться здесь: https://platform.openai.com/playground. Он еще интернет сервисы и учебники хорошо подсказывает.
- **Идеальное понимание языка.** Умение читать практически любой код, понимать его, запускать в уме очень помогает в понимании чужого кода, отладке, проектировании, с этим навыком появляется больше идей, можно находить лучшие решение методом исключения невозможных, и т.д. Это еще и помогает в изучении, т.к. многие идеи и алгоритмы проще выразить в виде кода. Язык программирования еще и в какой-то степени язык на котором программисты общаются между собой. Часто в их общении можно услышать - хватит болтать покажи пример кода. Понимание каждой мелочи в коде сильно упрощает проектирование, отладку, поддержание чистоты и качества кода, поиск лучших решений. К этому стоит стремиться, выискивать непонятные вещи и разбираться в них.
- **Шаблоны.** Большая часть программирования, это не придумывание чего-то с нуля, не изобретение велосипедов, а многократное переиспользование шаблонного кода. Этих шаблонов тысячи и нет смысла их все учить. Они запоминаются сами с получением опыта программирования, чтения чужого кода и перенятия чужих идей. Хорошие шаблоны кода для разных задач можно найти на сайте https://stackoverflow.com, хотя даже там бывает залайканный говнокод.
- **Метод уточки.** Если ты не можешь понять в чем ошибка в программе, попробуй выполнить ее по шагам, объясняя КАЖДУЮ деталь, так чтобы даже сидящая рядом с тобой уточка все поняла. Этот же метод часто используется чтобы убедиться что в программе нет ошибок, я его использую каждый раз после того как написал кусок кода, просматриваю и выполняю в уме.

- **Пошаговая отладка.** Инструменты пошаговой отладки очень помогают разобраться в своем коде, найти ошибки, убедиться что твое понимание кода соответствует реальности. При пошаговой отладке ты можешь буквально смотреть как код выполняется шаг за шагом и смотреть значения переменных на каждом шаге. Это очень помогает в изучении языка, в понимании того как там все работает изнутри. Пошаговую отладку JavaScript можно запустить в браузере, в DevTools, достаточно только вставить код в консоль, но перед кодом написать слово debugger. Я этот метод использую когда мне нужно убедиться, что в сложном коде нет явных ошибок алгоритма.
- **Навыки самостоятельного исследования.** Исследовательская работа является очень важной и неотъемлемой частью процесса разработки чего угодно. За какую задачу не возьмись, практически любая требует исследований, если конечно у тебя еще нет опыта выполнения в точности такой же задачи. Исследование по сути является поиском наилучшего ответа на такие вопросы за приемлемое время: какой инструмент здесь использовать, как организовать код, какое архитектурное решение применить к этой задаче, каковы возможности этого инструмента, как решить эту задачу, и т.д. Исследование - это мини научная работа, т.к. выполняется оно с использованием научного метода: сбор информации, генерация идей, проверка идей, выбор наилучшей.
- **Навыки самообучения.** Учиться программированию необходимо все время пока работаешь программистом, постоянно повышать свой уровень. Даже если программисту удается попасть в такую нишу, где учить больше ничего не нужно, выучил все и работай, то скорее всего это будет скучная и низкооплачиваемая работа, т.к. конкуренция здесь будет выше, интереса к такой работе будет меньше, т.к. все время делаешь одно и тоже. Лучшие и более высокооплачиваемые специалисты - это те которые умеют решать любые проблемы, включая самые сложные и казалось бы нерешаемые. Ведь в этом и есть задача программиста - решать задачи поставленные клиентами, которые часто бывают сложными или нетипичными. А таким специалистом можно стать только с навыками самообучения и самостоятельного исследования.