[Becoming a better programmer](https://www.khanacademy.org/computing/computer-programming/programming/good-practices/a/planning-a-programming-project)

Now that you understand the basics of programming, learn techniques that will help you be more productive and write more beautiful code.

\*

Planning a programming project

Becoming a programmer isn't just about learning the syntax and the concepts of a programming language: it's about figuring out how to use that knowledge to make programs. You've made a bunch of programs in this course, in the challenges and projects, but now you should come up with ideas for new programs - ideas that you're personally really excited about - and try to turn those into actual programs.

You probably won't know everything you need for your program when you start it, and that's totally okay -- you'll be motivated to learn those new things because of how much you want to make your program real. Programmers are constantly learning new things for new projects, and that's part of why we love it so much.

Let's step through the process of planning a programming project:

1. What do you want to make?

When I first started programming, I found myself constantly thinking of new programs to make and writing those down in a list. I was addicted to the power of creation, and there was so much my brain wanted to make. If you're like that, then you probably already have an idea of what you want to make, and perhaps you have your own list.

If you don't already have an idea, then here are some questions to help your brainstorming:

What's your favorite game - arcade game, board game, sports game? Could you make a simplified, digital version of that? Could you mix it up a bit, like give it a different theme or main characters?

What are your other favorite academic fields? If you love art, could you make an art-making program? If you love history, how about an interactive timeline? If you love science, how about a scientific simulation?

What's your favorite movie or TV show? Could you make a digital version of a scene or character from it? Maybe make a game based on it?

What's a real-life gadget that you love? Could you make a simulation of it?

Once you've picked an idea, you should write a description of it. For example, if I decided to make a clone of "Breakout", because that's my favorite retro arcade game, I might write:

Breakout: a game where you control a paddle at the bottom of the screen, and you use it to hit a ball upwards and at angles to break bricks. The goal is to break all the bricks, and not let the ball through the ground too many times.

You'll flesh that description out later, but for now, that gives you a good enough idea to keep going in the planning process.

2. What technology will you use?

In this step, you need to consider which technologies (languages/libraries/environments) you're familiar with or able to learn easily, and which of them are the most well suited for the job. For many of you, that list may be one item long, "1. JS+ProcessingJS", and that makes your decision easy.

Our JS+ProcessingJS environment works great for making animations, games, visualizations, and simulations. Just check out the community programs to see the wide range of programs that people make here.

Our environment doesn't work for other things like multi-player games, mobile apps, data-crunching applications. If you know other languages/environments (like JS+HTML, Python, SCRATCH, Swift, etc) and you're thinking of building something that doesn't make as much sense with ProcessingJS, then consider which of those technologies would be best suited for your program. If you want to build those things but don't know other technologies, you might want to come up with a new program idea. You can learn a new technology for a new project, but especially if you're just getting started in programming, it's a good idea to get really good at your first language first.

If I decided to make a game like Breakout, I'd pick JS+ProcessingJS, since I already know that technology and it also works great for 2D games like that.

3. What features will it include?

This is where we get into the real planning, and where (I think) it gets fun. Your goal in this step is to figure out what you're actually making- what will it look like, what features it will include, what features it \*won't\* include.

The first thing you can do is make "mock-ups" - sketches that look like the thing you're making, but without details like coloring or exact sizing. You can make mock-ups on paper, or with online programs:

To give you an idea of what mock-ups look like, I've included mock-ups below of my Breakout clone. I sketched each scene separately and drew arrows between them to show how one scene leads to another. Those arrows will help me understand what logic I need in my program to go between program states.

Now you can use those mock-ups to help you make a feature list, where you think of every feature in your program, and make it into a list.

For my Breakout clone, this could be my feature list, broken down by scene:

Game scene

User-controlled paddle

Multiple colored bricks

Angled ball movement

Collision detection

Life display

Score display

Sound effects

Main Scene

Play button

Help button

Help Scene

Text

Back button

Win Scene

Headline

Fireworks animation

Lose Scene

Text

Restart button

4. But what features must it include?

If we all had infinite time to make all the programs in our heads, then they'd all include every feature in our list. But we don't, so in this step, you have to decide which features are the most important, and which features you'll do only if we have time. This will also help you figure out which order to implement features in, from most to least important.

To help you figure out the importance of each feature, ask yourself these questions:

If I shared this with a friend, which features would I want to make sure were working?

Which features am I the most excited about building?

Which features are the most unique to my program?

Which features will I learn the most from implementing?

Are there any features that seem too far beyond my current skill level?

Then, go through your feature list from the last step, and either order the list or add a rank to each feature.

For my Breakout clone feature list, I've put "P1", "P2", and "P3" next to the features, signifying top priority (P1), middle priority (P2), and lowest priority (P3). I decided to prioritize the unique game mechanics over general game features like scenes, because I find that the most exciting about this project:

As a general tip for those of you making games, here are features that I'd recommend de-prioritizing: menus, multiple levels, 3D graphics. Focus on what's unique and fun about your game, then add those extras.

You can also turn your prioritized list into project versions, so you can easily see what you need to implement in each versions, and you can always stop after a particular version and be happy with what you've made.

Here's what the versions would look like for my Breakout clone:

User-controlled paddle

Multiple colored bricks

Angled ball movement

Collision detection

Life display

Score display

Start scene with play button

Win scene with headline

Sound effects

Help button

Fireworks

Lose scene with Restart button

5. How will you implement it?

You now have an idea of what features you'll be building first in your program - but if you start now, you'll be staring at a completely blank program with no code written, and that can be intimidating. Which variables should you write first? Which functions?

One way you can figure that out is to think about the "high level architecture" of your program - breaking it into categories like "objects", "logic", "user interaction", "user data", and "scenes" - and then think about how you might implement them, like as object-oriented object types, functions, or variables.

For example, here's an architecture for my Breakout clone:

Objects

Brick

isHit

Paddle

move

Ball

Scenes

Start

Game

End

Logic

Ball-brick collision

function

use bounding box

Paddle-ball angling

invert angle

User interaction

Keyboard-paddle movement

keyPressed

Buttons for scene changes

mouseClicked

User data

Ball deaths

array

Ball hits

Once you've thought about the high-level architecture, it should become more clear what you can start coding first.

You might decide to write your whole program in pseudo-code first, which we talk about later in this tutorial. Basically, it'd mean writing the whole program in plain English text inside a comment, and then slowly turning that into actual code.

6. What's your timeline?

How much time do you have to make this program? How many weeks, and how much time each day? What features will you write each week? Your goal in this step is to figure out a timeline for your project - which is particularly important if you have a deadline, but also useful so you start to understand how much time it takes you to write a program.

Here's a timeline for my Breakout clone, assuming 2-4 hours of work each week:

Week 1: Design and pseudo-code

Week 2: Rough visuals

Week 3: Ball moving/collision mechanics

Week 4: Scoring mechanics

Week 5: Scenes (Start/Win/Lose)

Week 6: Polish, Manual tests (QA), Prep for demo

Figuring out timelines for programming projects is hard. Some things that seem easy take way longer than you expect (like some weird bug that you spend hours debugging), some things that seem hard take less time than you expect. As a general rule, assume it will take you longer than you think, and adjust as you go along.

Are you ready!?

Hopefully this gives you an idea for the process of planning a programming project, and inspires you to start a project now. Depending on what you want to build, you might also decide to go through other courses first, like Advanced JS: Games & Visualizations or Advanced JS: Natural Simulations, to give you more ideas for building games and simulations.

The important thing is to make sure you start making your own programs at some point, because that is where you'll learn the most, and also where you'll get the most joy out of programming, because you're making your dreams into reality.

Video: Planning with pseudocode

\*

What to learn next

Hide tutorial navigation

Have you watched all of the talk-throughs and completed all of the challenges up until this point? Awesome!

Before talking about what you can learn next, let's review what you've learned so far: JavaScript and ProcessingJS.

Review: JavaScript

There are many programming languages in the world, and JavaScript is one of the most popular ones. A programming language typically includes ways to store data in computer memory, ways to manipulate that data, ways to organize code and run code repeatedly. You learnt how to do all of that in JavaScript via:

Variables: how to declare variables, assign, re-assign, and local vs. global scope.

Data types: numbers, booleans, strings, arrays, and objects.

Functions: how to group code into functions, pass arguments to them, and return values from them.

Conditionals: how to use if/else statements and logical expressions.

Loops: how to use while and for loops to repeat code.

If you learn a new programming language, you will probably discover that it shares many of those constructs in common with JavaScript, and they often differ only in the exact syntax or subtle details.

Review: ProcessingJS

To give you a fun way to learn JavaScript here, we gave you a drawing & animation "library" called ProcessingJS. That library is a collection of functions that you learnt how to use, and those functions know how to draw pixels into a canvas on the page. Here are examples of what you learnt how to use from ProcessingJS:

Shapes: like rect(), ellipse(), and line()

Colors: like fill(), stroke(), and background()

Text: like text() and textSize()

Events: like draw() and mousePressed()

Math: like random() and dist()

If you just use JavaScript without loading the ProcessingJS library, you will not have access to any of those functions. That's why we always load the library for you here.

Learning more

Now, there are a ton of ways that you can keep learning programming, both by going deeper into what you've learnt so far or learning something related.

Learn more on Khan Academy

We have two advanced tracks here on Khan Academy, with articles and challenges:

Advanced JS: Games & Visualizations

Advanced JS: Natural Simulations (Recommended for those who have some familiarity with basic trigonometry and physics, like in high school).

Algorithms (Recommended for those who have done logarithms / functions, like in Algebra 2)

Learn more JavaScript

There is still more to learn about JavaScript as a language, and several places that you can learn online. You may find that you already know the basics in these tutorials, so skip over them until you don't know something, or review them as practice.

TreeHouse: JavaScript Foundations

Mozilla Developer Network: JavaScript Guide

Codecademy: JavaScript

W3Schools JS Tutorial

Eloquent JavaScript

Learn more ProcessingJS

ProcessingJS is a powerful library, and we've only introduced half of it.

Besides our advanced tutorials on Games & Visualization and Natural Simulations, you can also watch tutorials created by members of our community:

Advanced Tutorials

Making a Pop-Dots Game

Work on a big project

Sometimes, the best way to really learn how to program is to work on a project that you're really passionate about.

Brainstorm ideas for a game, simulation, or story, and pick one that you think would be hard but not \*too\* hard.

Plan it out, using pseudo-code, sketching on paper, or writing it up in a document.

Start programming it. When you're stuck, try to find programs that do similar things (either from our official talk-throughs or from the community) and ask questions on them.

Share it with your friends and family when you're done. Reflect on how much you learnt.

Learn web development

Webpages are all written in HTML (to define their structure), CSS (to style them with different fonts and colors), and JavaScript (to make them interactive, like animating on user click). You already know the basics of JavaScript, but if you want to make webpages, you also need to learn HTML and CSS, like via these online tutorials:

Khan Academy: HTML/CSS

TreeHouse: Build a Website

CodeHS: HTML

Once you feel comfortable with HTML and CSS, you then need to learn how to use the JavaScript "DOM API", a library that lets you manipulate a webpage using JavaScript:

Khan Academy: HTML/JS

TreeHouse: Build an Interactive Website

Learn new languages

If you feel very comfortable with JavaScript, maybe you'd like to learn an entirely new language. Here are a few languages you might want to learn:

Python: used for webservers and data processing. Learn it on

Codecademy, Treehouse, or with this interactive online textbook.

Java: used to make Android apps. Learn it on Treehouse.

Objective C: used to make iPhone apps. Learn it on MakeGamesWIthUs or Treehouse.

Ruby: used with the Rails framework to create webservers. Learn it on

Codecademy or Treehouse.

PHP: used for webservers. Learn it on Treehouse or Codecademy.

Haskell: used for functional programming and popular with researchers. Learn it from Learn you a Haskell for Great Good.

Video: Help others learn!