Code 6

Planning Your Code

Learning Objectives:

In this unit, you will learn…

- How to plan out what you will code
- Ways to manage your project and structure your time

Planning Your Code

In Code 5 you planned out what your app will look like and how users will interact with it. In this next section you will learn two techniques to help you plan out your code and think about how your app will actually work. It's really important to plan your code so you can figure out if you need to learn something you don't already know and think through how things will work before getting to the computer!

Pseudocode

Writing **pseudocode** is an important part of the planning process of your app. Pseudocode uses plain language with some code mixed in to explain what your app does. You should use programming terms that you know, such as loop or conditionals, and any App Inventor components that you may want to use such as ListView or buttons. There aren't many rules for how to write pseudocode but the goal is to get an idea of how your app will work before you program it.

Let's look at an example from Code 3. In this app, the user can search a database of women scientists. The app displays the names in a ListView and the user can select a scientist to learn more about her.

Search Button

- When the user hits search
- The app uses a **for loop** to search through all of the names and descriptions stored in **TinyDB**
- **If** there is a match
  - The app adds the scientist name to an empty list variable called *searchResults*

- **If** *searchResults* is empty after the loop ends
  - The app notifies the user that there are no matches
- **Else**
  - The app displays *searchResults* in a **ListView**

Scientist Selection

- When the user makes a selection from **ListView1**
- The app opens a *Screen2*
- The app retrieves the description and name of the scientist from **TinyDB**
- The app displays the description and name of the scientist

Activity

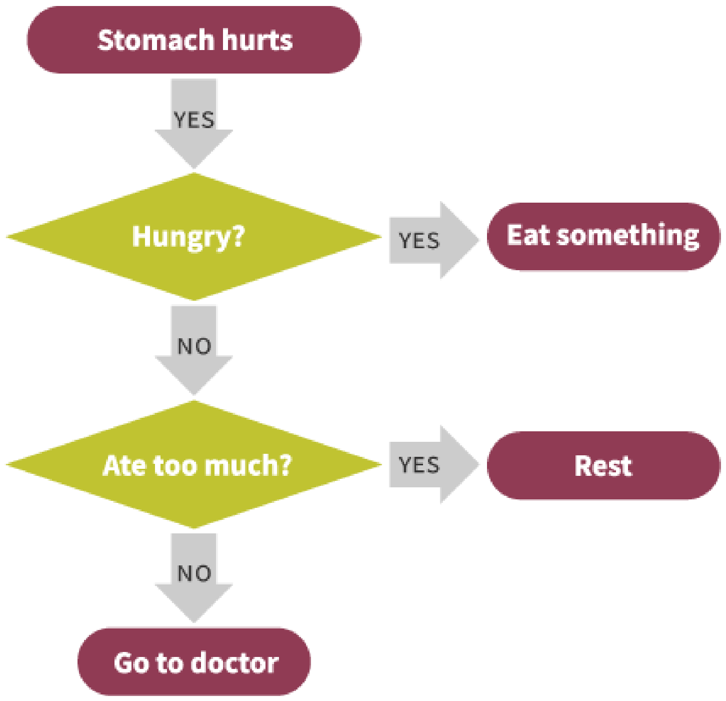
Write pseudocode for two event handlers in your app. If you're having trouble breaking down how an event handler in your app will work, you may want to ask your teammates or mentor for help!

Questions to ask yourself:

- What is the event that starts starts this code?
- Will your app need to use any stored values?
  - If so, where will the values be stored?
- Will your app need to use any variables? If so, it helps the give them a name to keep track of them.

Flowcharts

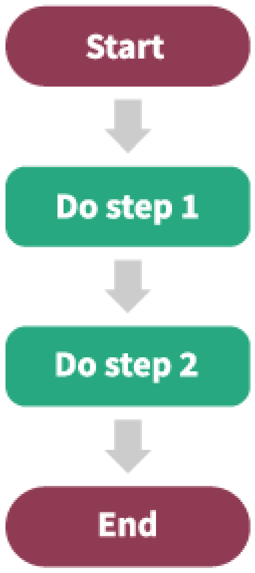
A **flowchart** is a diagram that represents an algorithm. It uses shapes and arrows to show how data moves through program. It is always read from top to bottom. Programmers use different shapes to represent different things that can happen to the data. Here is an example of an algorithm for finding out why someone's stomach hurts:



An **oval** shows where the algorithm starts and ends In the example above, there are three different places where the algorithm may end, either the person needs to eat something, they need to rest, or they need to see a doctor. These are called a **terminals.**

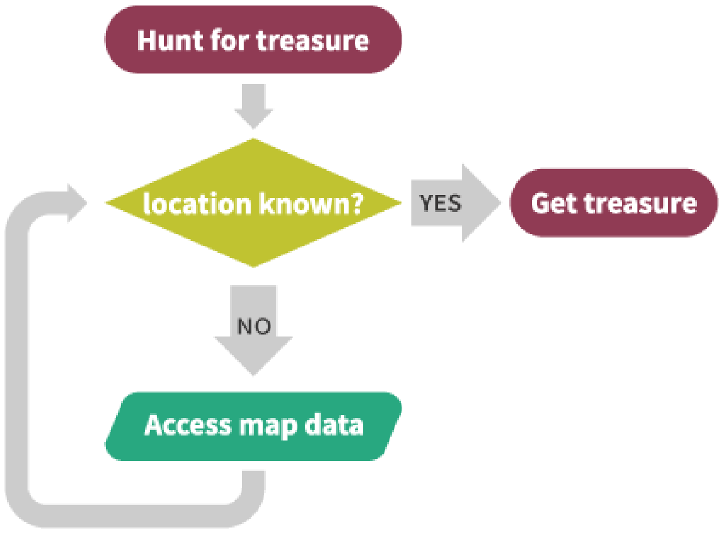
A **diamond** shows where a **decision** needs to be made. They are like "forks in the road". This is whenever a conditional needs to decide true or false for a condition. Two arrows should come out of a decision, one for **true** and one for **false.** Note how the arrows come out of the diamond in the example above, one comes out from the bottom and one from the side.

Here's another example of a flowchart:



This diagram uses a **rectangle**, which represents a **process**. A process is whatever you are having your code do. It could be alerting a user, setting a label to say something or adding two numbers together.

This flowchart shows a **while loop**. The condition and process (in this case, accessing an input) will repeat until the condition is satisfied. There's another type of loop you learned about in [Code 3](): a **for loop**. The condition in a **for loop** is to check if the **counter** has reached high enough yet, so the process will repeat a set number of times until the condition is met.



The last shape you should be aware of is the **parallelogram**. You should use this shape whenever you need to get data from a user or from somewhere else, like a website. The parallelogram is for **inputs.**

Activity

In this activity you will turn your pseudocode into a flowchart. Use the pseudocode you made for the activity above and follow these steps.

1. Put an oval around the beginning and ends of your event handler. This is called a terminal. There is often more than one ending to a flow chart. In the search button example above, the event handler ends either by displaying the search results in a **listview** or by notifying the user that there are no matches.

2. Put a diamond around where a decision needs to be made. This is whenever the app needs to

decide true or false for a condition. Remember, conditionals can lead to more than one possibility for the ending.

3. Add a parallelogram around all the inputs in your code or the places where you are getting data from either the user or another place, like a website or local database.

5. Put a rectangle around all of the processes in your code. This is whenever your app is doing something that doesn't fall into the categories above. This could be adding items to a list or sending an alert to the user.

6. Connect it all together with arrows! Pay special attention to how you are drawing the arrows for loops and conditionals and refer to the examples above if you need help.

Take a picture of one of your flowcharts and include it with your submission to earn a **point**! Don’t worry if it isn't exactly accurate, the whole purpose is to help planning and to learn something new! It is okay if your code doesn’t match your flowchart by the end of building your app, you don’t need to update your flow chart.

Pair Programming

Pair programming is when two programmers share a single workstation (one screen, keyboard and mouse among the pair), and either work together or take turns “driving” and “navigating”. In this scenario, the person sitting at the keyboard or touchscreen is the driver, and the other person is the navigator. The navigator is also actively involved in the programming task but is focused more on the bigger picture, answers questions that the driver has, and keeps her eye on the code to check for bugs. The driver and the navigator swap roles every so often.

There are advantages to pair programming, such as:

- Better quality of code since the navigator can check the work of the driver.

- Better communication between team members because the driver is providing a running commentary on what she is doing (or programming out loud), and the navigator is able to respond or ask what is happening if the driver is quiet.
- Knowledge can be shared and transferred on your team, especially if one person is more of a beginner and others are more advanced.
- It can help make your teamwork more efficient because the driver can attend to fixing a bug while the navigator can keep focused on the task and help regain focus afterwards.

Tips for pair programming:

- Don't be afraid to say, "Let's try out your idea first!" Sometimes where you're driving, you need to know when to listen to your navigator. The goal is to use the best ideas and to arrive at them through collaboration, and to avoid errors.
- Here are ten suggestions for improving the experience.

"None of us is as smart as all of us."

–Ken Blanchard, author and management expert

Time Management

At this point, you may have questions, such as:

- How to work efficiently towards this goal, and meet the deadline?
- Who will test the digital prototype and when?
- How to know when you have a working digital prototype to submit?

In order to help you get the programming part of your project on track, let's review the stages of the software development process.

- The users, their needs, and an MVP has been identified (from Code 5)

- The actions that the users will take in your app have been mapped out with a flow chart
- Software is created
- User testing happens, bugs are identified
- Bugs are fixed and the team ensures that the app is free of errors
- If the app works without any flaws on a mobile device and the team is satisfied with the end result, then it is ready to be shared

Tips:

- Set a schedule
  - You can create a Gantt Chart to help you create a schedule for when things need to be done, and know if something is dependent on something else to happen. Here is an example of how a Gantt Chart is used for software development from dreamincode.
  - Assign tasks to team members, and set deadlines and goal reminders on your calendars!
  - If you are using the 20 week curriculum, your team should give itself about 8 weeks to program your app
  - If you are using the 12 week curriculum, your team should give itself about 5 weeks to program your app
  - Make sure that your digital prototype is done and ready to be shared no later than two days before the Technovation Challenge submission deadline, which will be April 26, 2017. This will save you a lot of stress in case there are connectivity issues
- Break the tasks down and divide them up
  - Have everyone on the team work together in pairs and tackle different tasks simultaneously. This can help move the coding aspect of the project along faster.
- Prioritize
  - Make sure your app is functional first, and then it can be made to look better afterwards. Remember, you are submitting a digital prototype and the judges will be interested in knowing that it works, gets the job done, and is easy to use. You can use your demo video and your business plan to let the judges know what your future plans are for the app in terms of new features you would add. Right now you should focus on MVP.
  - Ask your mentor to help you if you are having a hard time prioritizing your tasks
- Focus
  - Turn off all devices you are not actively using during your team meeting or your programming time. You can also mute notifications and anything that will distract you.
  - Make sure you finish one task before moving on to the next. Multitasking doesn't usually save time. If you are working in pairs, the navigator can help you stay focused.

Here is the Technical Checklist that the judges will be using. You can use this to check off and be able to explain the components you used in your app and make sure that you get the highest scores possible.