Variables, Conditionals, and Functions ('21-'22)

Variables Explore

Resources

A Short Introduction to EIPM

What is EIPM?

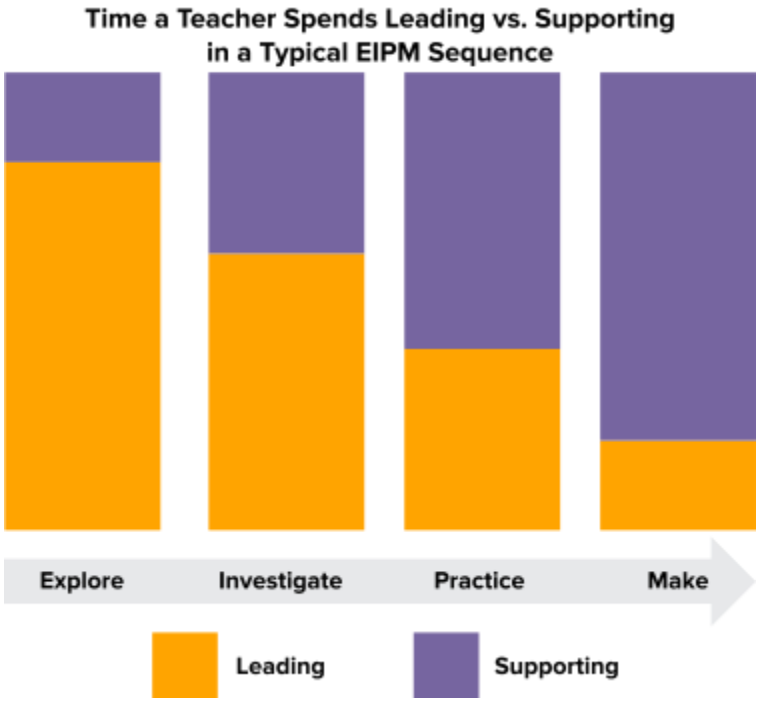
EIPM is a structured approach to teaching programming in Code.org's CSP Curriculum. It's designed to meet the needs of diverse learners, encourage collaboration, support independent creation, and clarify the role of the teacher throughout the learning process. Each letter represents a different type of lesson (E - Explore, I - Investigate, P - Practice, M - Make) which are taught in sequence for each major programming construct. A typical programming unit will feature 2 or 3 EIPM sequences, followed by a unit project and a multiple choice assessment. The visual below shows that structure for Unit 5 - Lists, Loops and Traversals, with each square being a separate lesson.

Why did you develop EIPM?

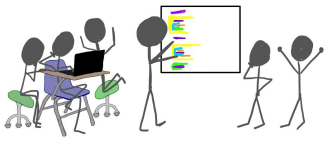
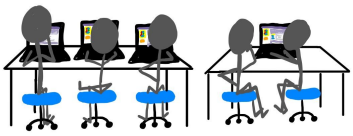
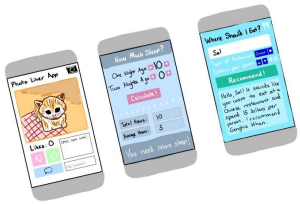
The biggest motivator in developing EIPM was direct feedback from students and teachers. While they had many positive things to say about our programming units, we also heard some consistent concerns. Due to the way our programming units are designed, students were often working in isolation, teachers lacked clarity on their role, and classrooms would arrive at the end of the unit lacking the skills to take on major independent projects. We took this feedback, consulted research and experts in the field of computing education, and created a pathway for introducing programming concepts that is more collaborative, clarifies the teacher's role, and more intentionally prepares students to create programming projects independently.

The Teacher's Role in the Classroom

The EIPM structure provides more clarity on the way students are expected to learn in each lesson and your role in facilitating that learning. In the Explore and Investigate lessons you will spend relatively more time leading group activities and class-wide discussions. In the Practice and Make lessons you will spend more time circulating the room to provide support to individuals and small groups. This predictable transition clarifies your role in any individual lesson and encourages students to work with greater independence as they develop mastery over a new concept. The graphic to the right shows the gradual transition from leading students through activities to supporting students working more independently.



A Summary of Each Lesson Type

	Overview	Goal
Explore 	Students explore the new concept through a teacher-led hands-on group activity. • Typically uses physical manipulatives • Teacher leads with support of slides and activity guides	Students begin to develop a shared mental model and understand the main ideas of the new concept.
Investigate 	Students investigate two or three sample programs that use the new concept. • Close-reading of working programs • Teacher-led discussions • Tasks to modify apps	Students become comfortable reading and modifying programs that use the new concept.
Practice 	Students practice using the new concept through a scaffolded series of programming activities. • Students work independently or in pairs • Teacher introduces debugging practices at the beginning of the Activity and circulates the room during the lesson to provide targeted support.	Students gain confidence in writing and debugging programs that use the new concept.
Make 	Students make a target app for which they are given the screen elements but little to no starter code. • Students are provided high-level steps to break down the project • Teacher supports students by directing them towards notes, previous work, and debugging strategies practiced in earlier lessons.	Students are able to independently decide when and how to use the new concept in the context of a larger project.

Variables, Conditionals, and Functions ('21-'22)

Variables Investigate

Resources

A Short Introduction to EIPM

What is EIPM?

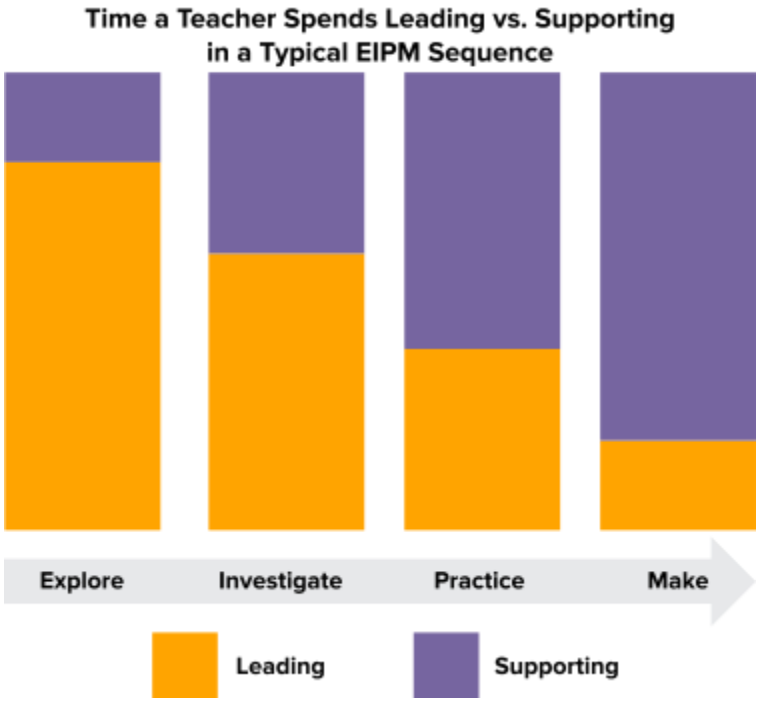
EIPM is a structured approach to teaching programming in Code.org's CSP Curriculum. It's designed to meet the needs of diverse learners, encourage collaboration, support independent creation, and clarify the role of the teacher throughout the learning process. Each letter represents a different type of lesson (E - Explore, I - Investigate, P - Practice, M - Make) which are taught in sequence for each major programming construct. A typical programming unit will feature 2 or 3 EIPM sequences, followed by a unit project and a multiple choice assessment. The visual below shows that structure for Unit 5 - Lists, Loops and Traversals, with each square being a separate lesson.

Why did you develop EIPM?

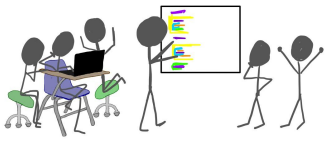
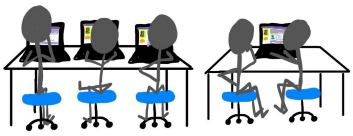
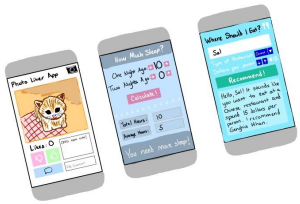
The biggest motivator in developing EIPM was direct feedback from students and teachers. While they had many positive things to say about our programming units, we also heard some consistent concerns. Due to the way our programming units are designed, students were often working in isolation, teachers lacked clarity on their role, and classrooms would arrive at the end of the unit lacking the skills to take on major independent projects. We took this feedback, consulted research and experts in the field of computing education, and created a pathway for introducing programming concepts that is more collaborative, clarifies the teacher's role, and more intentionally prepares students to create programming projects independently.

The Teacher's Role in the Classroom

The EIPM structure provides more clarity on the way students are expected to learn in each lesson and your role in facilitating that learning. In the Explore and Investigate lessons you will spend relatively more time leading group activities and class-wide discussions. In the Practice and Make lessons you will spend more time circulating the room to provide support to individuals and small groups. This predictable transition clarifies your role in any individual lesson and encourages students to work with greater independence as they develop mastery over a new concept. The graphic to the right shows the gradual transition from leading students through activities to supporting students working more independently.



A Summary of Each Lesson Type

	Overview	Goal
Explore 	Students explore the new concept through a teacher-led hands-on group activity. • Typically uses physical manipulatives • Teacher leads with support of slides and activity guides	Students begin to develop a shared mental model and understand the main ideas of the new concept.
Investigate 	Students investigate two or three sample programs that use the new concept. • Close-reading of working programs • Teacher-led discussions • Tasks to modify apps	Students become comfortable reading and modifying programs that use the new concept.
Practice 	Students practice using the new concept through a scaffolded series of programming activities. • Students work independently or in pairs • Teacher introduces debugging practices at the beginning of the Activity and circulates the room during the lesson to provide targeted support.	Students gain confidence in writing and debugging programs that use the new concept.
Make 	Students make a target app for which they are given the screen elements but little to no starter code. • Students are provided high-level steps to break down the project • Teacher supports students by directing them towards notes, previous work, and debugging strategies practiced in earlier lessons.	Students are able to independently decide when and how to use the new concept in the context of a larger project.

Variables, Conditionals, and Functions ('21-'22)

Variables Practice

Resources

Guide to Debugging

Introduction to Debugging

Debugging is the process of finding and fixing problems in code. For most programs, the time spent debugging far outweighs the time spent writing new code. Whether students or professional engineers, all programmers get bugs, and debugging is a normal part of the programming process.

Although students may see bugs as inconveniences to be eliminated as soon as possible, bugs in student programs should be seen as opportunities to reinforce positive attitudes toward debugging and persistence, identify and address student misconceptions, and further develop good debugging skills. Your role as the teacher is to avoid directly debugging for your students, but to guide students in better taking advantage of these opportunities.

Reinforcing positive attitudes

Finding bugs is the first step toward fixing them. Programmers deliberately test their code in order to uncover any possible bugs. Celebrate the discovery of new bugs as students report them in the classroom, framing finding a bug as the first step to fixing it. Model enjoying the interesting or funny behaviors a bug can cause, such as making a sprite move in unexpected ways, or distorting an image on a web page. Remind students that if programs all worked exactly as they wanted the first time, programming wouldn't be as interesting or fun. Encourage students as they get frustrated, reinforcing the debugging strategies and students' self-efficacy as they improve their debugging skills, and talk about the bugs that you get in your own programs, reminding them that everyone gets bugs, even professional software developers.

Identify and address misconceptions

Often a bug occurs because students have misconceptions around how the computer interprets their code. As part of the debugging process, have students explain their code, line by line, reminding them that the program is really only doing exactly as it was told. If the program displays an error message, ask the student to connect the message to what is happening in the code itself. Prompt them with relevant questions about how the associated programming structures (such as conditionals or loops) work, and refer them back to previous lessons, worked examples, or documentation when needed. After students have found the bug, ask them to reflect on what they learned about the associated programming structures and how they could explain it to a friend with a similar bug.

Develop debugging skills

As students get more experience debugging their programs, they will build the skills they need to debug more independently. Help students to generalize the strategies that they use by asking them to reflect on what processes were effective and reframing those processes as a general strategy. They should also learn ways to simplify the debugging process by making their code more readable with good naming conventions, clear formatting, and relevant comments; organizing code into functions or other logical chunks where appropriate; and testing small pieces of code as they go. Call out these practices as facilitating debugging as you help students with their code.

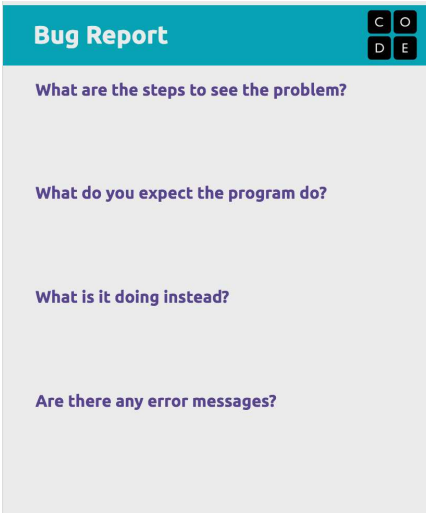
Debugging as problem solving

In computer science, debugging is framed as a form of problem solving, and students can use a version of the four step Problem Solving Process as a framework for debugging their programs. Just as in other forms of problem solving, many students may jump to the "Try" part of the framework and begin making changes to their code before understanding the nature of the bug itself. Remind them that all parts of the process are important, and that ignoring the other three steps will actually make debugging more time consuming and difficult in the long run.

Define - Describe the bug

In the context of debugging, defining the problem is describing the bug. This step can be done by anyone using the program, not just the person who will eventually be debugging it. Students will need to know the following information before they move on to the next step:

- When does it happen?
- What did you expect the program to do?
- What did it do instead?
- Are there any error messages?



Some bugs will keep the code from running at all, while others will run, but not correctly, or will run for a while, then stop or suddenly do something unexpected. All of those things are clues that will help the student find the bug in the next step.

You can encourage students to clearly describe the bugs they find by having them write up bug reports, such as in this worksheet. As you foster a positive culture around debugging, encourage students to write bug reports for their classmates' code as well as their own.

Prepare - Hunt for the bug

In most cases, hunting for the bug (the "prepare" step in the debugging process) will take up most of a programmer's time. Remind students that it's natural to take a long time to find a bug, but that there are things that will make their search easier.

1. **Why is the bug happening?**
   Often students focus on what they want the program to do and why they believe their code is correct, rather than investigating what could be causing the bug. Encourage students to start with the error messages and what is actually happening when the program is run, and try to connect that with the code that they have written, rather than explain why their code "should" be working. They can also "trace" their code, by reading it line by line, not necessarily from top to bottom, but in the order that the computer would interpret it while the program is running. Using debugging tools such as watchers, breakpoints, or the inspector tool may help them to identify why the code is running as it is.
2. **What changed right before the bug appeared?**
   If students have been testing their code along the way (as they should!), they can focus on code that they have recently changed. As they investigate that code, they should follow the logic of their program in the same order that the code runs, rather than reading the code line by line from top to bottom.
3. **How does this code compare to "correct" solutions?**
   Students should also make use of the various resources available to them, such as working projects, examples in previous lessons, and code documentation. Have them compare the patterns that they find in the documentation and exemplars to their own code, differentiating between the programming patterns that should be the same (loops, counter pattern, HTML syntax) and specifics of their program that will be different (variable names, image URLs, coordinates).

Try - Change the code

If students believe that they have found the bug, they can go ahead and try to fix it, but in many cases they may want to make changes to the code to narrow down their search. Some common strategies include:

1. **Commenting out code**
   By commenting out sections of the program, students can narrow down the part of the program that is causing the bug. After commenting out large sections of code, function calls, or html elements, test whether the bug is eliminated. This method is especially helpful when students have used good modular programming techniques.
2. **Print to the console**
   Using the console log command can help students to understand whether a conditional has been triggered, or keep track of a quickly changing variable over a period of time.
3. **Change the starting values of variables**
   Changing the starting values of variables can make it easier to reproduce a certain bug. For example, students may want to change the starting score to 99 to test a bug that only occurs when the score reaches 100. They may want to set their number of lives to a very high number to allow them to test for longer before losing the game.
4. **Amplify small effects**
   Sometimes bugs have such tiny effects that it's difficult to investigate them. Amplifying small effects, such as making elements move further on the screen or giving web page elements a background color to make them more visible, can make it easier to understand what is happening in a program.

In many cases, students will introduce new bugs during the debugging process, especially if they are randomly changing code as part of a "guess and check" method. Prompt them to explain why they are changing the code, and encourage them to make small changes and test them often, making it easier to go back if their solution didn't work.

Reflect - Document what happened

As students make changes and see the effects, they should reflect and document on their experiences. This will help them to build a better model of how the program constructs work and what debugging strategies are most effective. Students should consider the following after they have eliminated a bug from their program:

1. **What caused the bug?**
   The computer had a reason for doing what it did, and students should understand why their code caused the bug itself. There may be a rule that they can share with others in the class ("There is no closing tag for images") or a misconception ("Sprites all get drawn to the screen when `drawSprites` is called, not when they are updated.") that they can clear up.
2. **How did you find the bug?**
   Have students describe the debugging process that they used, paying special attention to how the type of bug and any error messages lent themselves to particular debugging strategies. Help them to generalize their strategies so that they can use them in a variety of situations. (e.g. "I had to capitalize the 'r' in my variable" might become "You double checked the capitalization and spelling of the variable the program didn't recognize.")
3. **What in your code made it easier or harder to debug?**
   Debugging is a great time to reinforce "clean code" practices, such as good naming conventions, use of functions or other ways of "chunking" code into logical sections, commenting, and clear formatting. Point out when comments or well named functions and variables make it easier to trace code and find an error. Debugging is also easier when students have separated out code into logical chunks and functions, which can be commented out individually.

Writing debuggable code

Various practices will make it easier for students to debug their code. Encourage students to neatly format their code, as well as make good use of comments and whitespace. Organizing code into logical chunks, using functions, and having reasonable names for classes, variables, and functions will help them to read and interpret their code as they debug. Point out times when students' good programming practices have made it easier for them to debug their own code.

A Short Introduction to EIPM

What is EIPM?

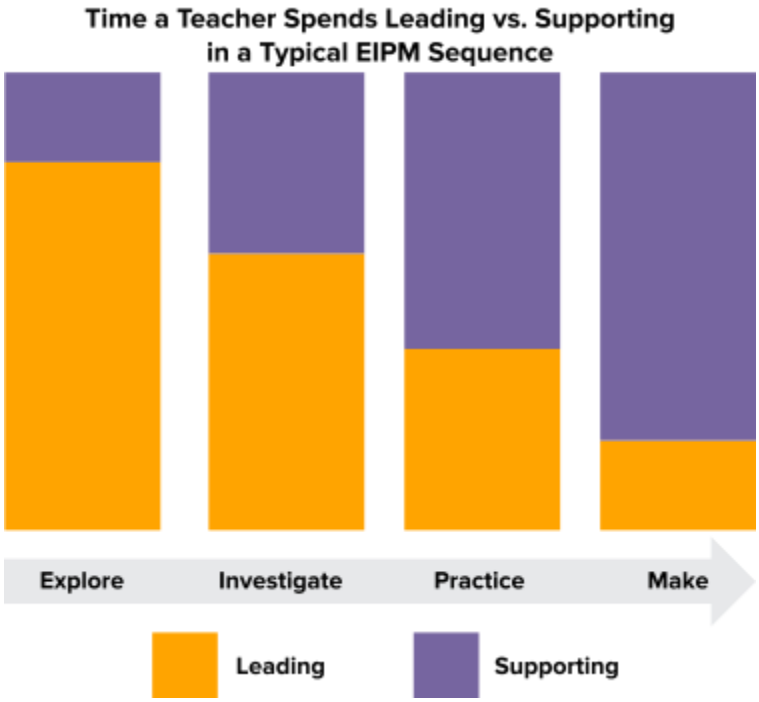
EIPM is a structured approach to teaching programming in Code.org's CSP Curriculum. It's designed to meet the needs of diverse learners, encourage collaboration, support independent creation, and clarify the role of the teacher throughout the learning process. Each letter represents a different type of lesson (E - Explore, I - Investigate, P - Practice, M - Make) which are taught in sequence for each major programming construct. A typical programming unit will feature 2 or 3 EIPM sequences, followed by a unit project and a multiple choice assessment. The visual below shows that structure for Unit 5 - Lists, Loops and Traversals, with each square being a separate lesson.

Why did you develop EIPM?

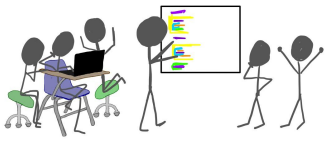
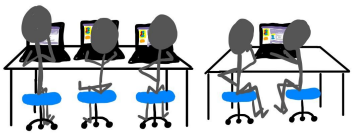
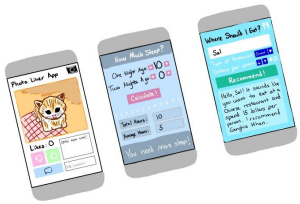
The biggest motivator in developing EIPM was direct feedback from students and teachers. While they had many positive things to say about our programming units, we also heard some consistent concerns. Due to the way our programming units are designed, students were often working in isolation, teachers lacked clarity on their role, and classrooms would arrive at the end of the unit lacking the skills to take on major independent projects. We took this feedback, consulted research and experts in the field of computing education, and created a pathway for introducing programming concepts that is more collaborative, clarifies the teacher's role, and more intentionally prepares students to create programming projects independently.

The Teacher's Role in the Classroom

The EIPM structure provides more clarity on the way students are expected to learn in each lesson and your role in facilitating that learning. In the Explore and Investigate lessons you will spend relatively more time leading group activities and class-wide discussions. In the Practice and Make lessons you will spend more time circulating the room to provide support to individuals and small groups. This predictable transition clarifies your role in any individual lesson and encourages students to work with greater independence as they develop mastery over a new concept. The graphic to the right shows the gradual transition from leading students through activities to supporting students working more independently.



A Summary of Each Lesson Type

	Overview	Goal
Explore 	Students explore the new concept through a teacher-led hands-on group activity. • Typically uses physical manipulatives • Teacher leads with support of slides and activity guides	Students begin to develop a shared mental model and understand the main ideas of the new concept.
Investigate 	Students investigate two or three sample programs that use the new concept. • Close-reading of working programs • Teacher-led discussions • Tasks to modify apps	Students become comfortable reading and modifying programs that use the new concept.
Practice 	Students practice using the new concept through a scaffolded series of programming activities. • Students work independently or in pairs • Teacher introduces debugging practices at the beginning of the Activity and circulates the room during the lesson to provide targeted support.	Students gain confidence in writing and debugging programs that use the new concept.
Make 	Students make a target app for which they are given the screen elements but little to no starter code. • Students are provided high-level steps to break down the project • Teacher supports students by directing them towards notes, previous work, and debugging strategies practiced in earlier lessons.	Students are able to independently decide when and how to use the new concept in the context of a larger project.

Variables, Conditionals, and Functions ('21-'22)

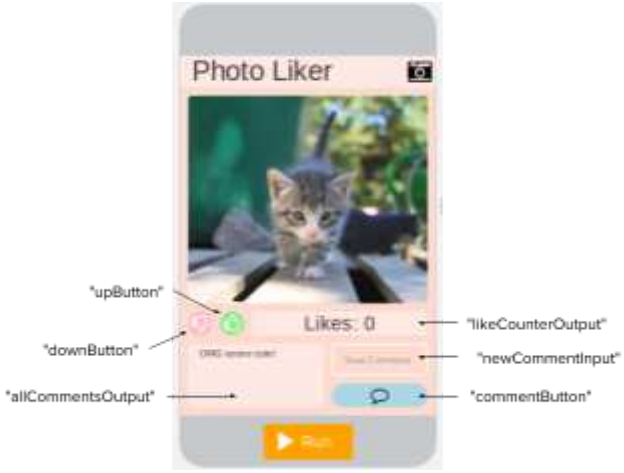
Variables Make

Resources

Name(s)_____ Period _____ Date _____

Activity Guide - Variables Make

Your Goal: Write the code to make the Photo Liker App. You've already been given every screen element and have comments that will help you design your program.



Step 1 - Try using this app

- Try all of the buttons and add a comment to the picture
- Discuss with a Partner
  - What does this app do?
  - What are the inputs?
  - What are the outputs?
  - What's one piece of information that might be stored in a variable?

Step 2 - Plan

Fill in the information in the table below for each event handler you'll need to create

Element ID	Description of What the Event Handler will Do

Fill in the table below for each variable you'll need to create.

Variable Name	What the Variable Stores

Step 3 - Write Your Code

- Write the code for the app, using your plan above and the comments provided in Code Studio to help
- Step You Can Follow
    - Create all the variables from your table above.
    - Give your variables a starting value using the assignment operator (=)
    - Create blank event handlers (onEvent) for each screen element in your table above
    - Write the code to make the "upButton" work and test it out. If it's working correctly, the "Likes" count should go up by one every time you click it
    - Write code to make the "downButton" work and test it.
    - Write code to make the "commentButton" work
    - Use your debugging skills to identify unexpected behavior and fix your program
    - Comment your code as you go, explaining what each event handler does
    - Check the Help & Tips tab for ideas about Programming Patterns you can use
- Extension Ideas
    - Set the text in "new_comment" to blank after the comment has been added
    - Add sounds to each button

Step 4 - Submit

Before you submit, check the rubric below to make sure your program meets the requirements of the task.

Category	Extensive Evidence	Convincing Evidence	Limited Evidence	No Evidence
Code: Event Handlers Created	onEvents are defined for all the required buttons.	onEvents are defined for most of the required buttons.	onEvents are defined for some of the buttons.	onEvents are not designed for any buttons.
Code: Variables	Variables are defined to store the amount of likes and the comments. Variables are named in a clear and understandable way.	Variables are defined to store the amount of likes and the comments	One variable is present that stores either the amount of likes or the comments	There are no variables which store the necessary information for the app to work correctly.
Code: Event Handlers Written	All necessary variables are updated inside of the onEvents.	Most necessary variables are updated inside of the onEvents.	Some of the necessary variables are updated inside of the onEvents.	None of the necessary variables are updated inside of the onEvents.
Code: Output Information	The screen correctly displays the amount of likes and the total comments. Sound plays when different buttons are clicked.	The screen correctly displays the amount of likes and the total comments.	The screen correctly displays either the amount of likes or the some amount of comments.	The screen does not display the amount of likes or the comments.
Code runs without errors.	No errors are present in the code.	At most one error is present in the code.	Some errors are present in the code.	Many errors are present in the code.
Coding Comments	Comments are used to correctly explain the purpose and function of all onEvents.	Comments are used to correctly explain the purpose and function of most onEvents.	Comments are used to explain the purpose and function of some onEvents.	Comments are not present

Variables, Conditionals, and Functions ('21-'22)

Conditionals Practice

Resources

Guide to Debugging

Introduction to Debugging

Debugging is the process of finding and fixing problems in code. For most programs, the time spent debugging far outweighs the time spent writing new code. Whether students or professional engineers, all programmers get bugs, and debugging is a normal part of the programming process.

Although students may see bugs as inconveniences to be eliminated as soon as possible, bugs in student programs should be seen as opportunities to reinforce positive attitudes toward debugging and persistence, identify and address student misconceptions, and further develop good debugging skills. Your role as the teacher is to avoid directly debugging for your students, but to guide students in better taking advantage of these opportunities.

Reinforcing positive attitudes

Finding bugs is the first step toward fixing them. Programmers deliberately test their code in order to uncover any possible bugs. Celebrate the discovery of new bugs as students report them in the classroom, framing finding a bug as the first step to fixing it. Model enjoying the interesting or funny behaviors a bug can cause, such as making a sprite move in unexpected ways, or distorting an image on a web page. Remind students that if programs all worked exactly as they wanted the first time, programming wouldn't be as interesting or fun. Encourage students as they get frustrated, reinforcing the debugging strategies and students' self-efficacy as they improve their debugging skills, and talk about the bugs that you get in your own programs, reminding them that everyone gets bugs, even professional software developers.

Identify and address misconceptions

Often a bug occurs because students have misconceptions around how the computer interprets their code. As part of the debugging process, have students explain their code, line by line, reminding them that the program is really only doing exactly as it was told. If the program displays an error message, ask the student to connect the message to what is happening in the code itself. Prompt them with relevant questions about how the associated programming structures (such as conditionals or loops) work, and refer them back to previous lessons, worked examples, or documentation when needed. After students have found the bug, ask them to reflect on what they learned about the associated programming structures and how they could explain it to a friend with a similar bug.

Develop debugging skills

As students get more experience debugging their programs, they will build the skills they need to debug more independently. Help students to generalize the strategies that they use by asking them to reflect on what processes were effective and reframing those processes as a general strategy. They should also learn ways to simplify the debugging process by making their code more readable with good naming conventions, clear formatting, and relevant comments; organizing code into functions or other logical chunks where appropriate; and testing small pieces of code as they go. Call out these practices as facilitating debugging as you help students with their code.

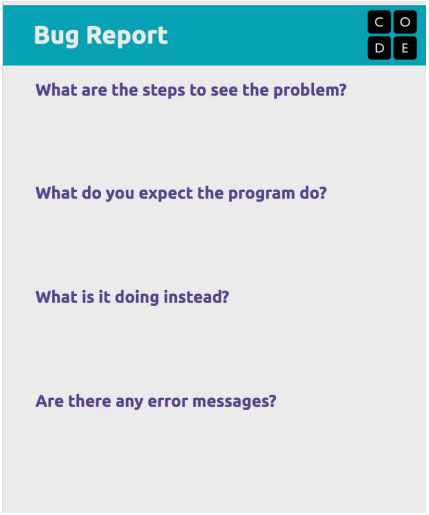
Debugging as problem solving

In computer science, debugging is framed as a form of problem solving, and students can use a version of the four step Problem Solving Process as a framework for debugging their programs. Just as in other forms of problem solving, many students may jump to the "Try" part of the framework and begin making changes to their code before understanding the nature of the bug itself. Remind them that all parts of the process are important, and that ignoring the other three steps will actually make debugging more time consuming and difficult in the long run.

Define - Describe the bug

In the context of debugging, defining the problem is describing the bug. This step can be done by anyone using the program, not just the person who will eventually be debugging it. Students will need to know the following information before they move on to the next step:

- When does it happen?
- What did you expect the program to do?
- What did it do instead?
- Are there any error messages?



Some bugs will keep the code from running at all, while others will run, but not correctly, or will run for a while, then stop or suddenly do something unexpected. All of those things are clues that will help the student find the bug in the next step.

You can encourage students to clearly describe the bugs they find by having them write up bug reports, such as in this worksheet. As you foster a positive culture around debugging, encourage students to write bug reports for their classmates' code as well as their own.

Prepare - Hunt for the bug

In most cases, hunting for the bug (the "prepare" step in the debugging process) will take up most of a programmer's time. Remind students that it's natural to take a long time to find a bug, but that there are things that will make their search easier.

1. **Why is the bug happening?**
   Often students focus on what they want the program to do and why they believe their code is correct, rather than investigating what could be causing the bug. Encourage students to start with the error messages and what is actually happening when the program is run, and try to connect that with the code that they have written, rather than explain why their code "should" be working. They can also "trace" their code, by reading it line by line, not necessarily from top to bottom, but in the order that the computer would interpret it while the program is running. Using debugging tools such as watchers, breakpoints, or the inspector tool may help them to identify why the code is running as it is.
2. **What changed right before the bug appeared?**
   If students have been testing their code along the way (as they should!), they can focus on code that they have recently changed. As they investigate that code, they should follow the logic of their program in the same order that the code runs, rather than reading the code line by line from top to bottom.
3. **How does this code compare to "correct" solutions?**
   Students should also make use of the various resources available to them, such as working projects, examples in previous lessons, and code documentation. Have them compare the patterns that they find in the documentation and exemplars to their own code, differentiating between the programming patterns that should be the same (loops, counter pattern, HTML syntax) and specifics of their program that will be different (variable names, image URLs, coordinates).

Try - Change the code

If students believe that they have found the bug, they can go ahead and try to fix it, but in many cases they may want to make changes to the code to narrow down their search. Some common strategies include:

1. **Commenting out code**
   By commenting out sections of the program, students can narrow down the part of the program that is causing the bug. After commenting out large sections of code, function calls, or html elements, test whether the bug is eliminated. This method is especially helpful when students have used good modular programming techniques.
2. **Print to the console**
   Using the console log command can help students to understand whether a conditional has been triggered, or keep track of a quickly changing variable over a period of time.
3. **Change the starting values of variables**
   Changing the starting values of variables can make it easier to reproduce a certain bug. For example, students may want to change the starting score to 99 to test a bug that only occurs when the score reaches 100. They may want to set their number of lives to a very high number to allow them to test for longer before losing the game.
4. **Amplify small effects**
   Sometimes bugs have such tiny effects that it's difficult to investigate them. Amplifying small effects, such as making elements move further on the screen or giving web page elements a background color to make them more visible, can make it easier to understand what is happening in a program.

In many cases, students will introduce new bugs during the debugging process, especially if they are randomly changing code as part of a "guess and check" method. Prompt them to explain why they are changing the code, and encourage them to make small changes and test them often, making it easier to go back if their solution didn't work.

Reflect - Document what happened

As students make changes and see the effects, they should reflect and document on their experiences. This will help them to build a better model of how the program constructs work and what debugging strategies are most effective. Students should consider the following after they have eliminated a bug from their program:

1. **What caused the bug?**
   The computer had a reason for doing what it did, and students should understand why their code caused the bug itself. There may be a rule that they can share with others in the class ("There is no closing tag for images") or a misconception ("Sprites all get drawn to the screen when `drawSprites` is called, not when they are updated.") that they can clear up.
2. **How did you find the bug?**
   Have students describe the debugging process that they used, paying special attention to how the type of bug and any error messages lent themselves to particular debugging strategies. Help them to generalize their strategies so that they can use them in a variety of situations. (e.g. "I had to capitalize the 'r' in my variable" might become "You double checked the capitalization and spelling of the variable the program didn't recognize.")
3. **What in your code made it easier or harder to debug?**
   Debugging is a great time to reinforce "clean code" practices, such as good naming conventions, use of functions or other ways of "chunking" code into logical sections, commenting, and clear formatting. Point out when comments or well named functions and variables make it easier to trace code and find an error. Debugging is also easier when students have separated out code into logical chunks and functions, which can be commented out individually.

Writing debuggable code

Various practices will make it easier for students to debug their code. Encourage students to neatly format their code, as well as make good use of comments and whitespace. Organizing code into logical chunks, using functions, and having reasonable names for classes, variables, and functions will help them to read and interpret their code as they debug. Point out times when students' good programming practices have made it easier for them to debug their own code.

Variables, Conditionals, and Functions ('21-'22)

Conditionals Make

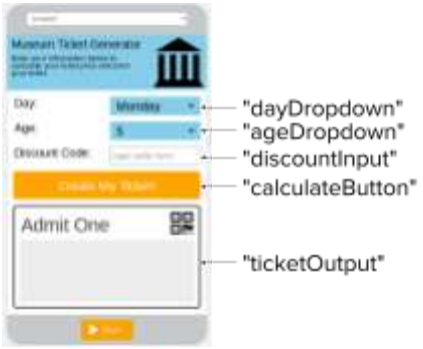
Resources

Name(s)_____ Period _____ Date _____

Activity Guide - Conditionals Make

Step 1 - Try the app
Try making tickets for different combinations of inputs.

- Make a ticket for a weekend.
- Make a ticket for a weekday (Monday - Friday) and someone 18 or younger.
- Try the discount code "FREEFRIDAY" on a Friday.



Discuss with a Partner

- What variables would you need to program this app?
- Where does this app use conditionals (if-statements)?

Step 2 - Plan
Variables: Fill in the table below for each variable you'll need to create.

Variable Name	What the Variable Stores

Conditionals: Draw a flowchart that follows the rules below. There's more than one way to do it. Use the table to make sure that your flowchart works for different combinations of age, day, and discount code.

- On the weekends ("Saturday" and "Sunday") everyone pays full price of $10
- On weekdays (Monday through Friday) if you are 18 years or younger you pay $5.
- If you use the discount code "FREEFRIDAY" on a Friday you get in for $0. No other discount codes will work and the code only works on Fridays.

Age	Day	Discount Code	Price
18	Monday	*none*	$5
18	Saturday	*none*	$10
50	Monday	*none*	$10
50	Saturday	*none*	$10
18	Tuesday	FREEFRIDAY	$5
50	Friday	FREEFRIDAY	$0
18	Friday	FREE	$5
50	Friday	FREE	$10

Step 3 - Write Your Code

- Write the code for the app, using your plan above and the comments provided in Code Studio to help
- Step You Can Follow
    - Create all the variables from your table above.
    - Give your variables a starting value using the assignment operator (=)
    - Create an event handler (onEvent) for the "calculate_button"
    - Inside the event handler start writing code for your conditional using your flowchart
    - Test your code as you go using the table below. At the end it should work for every combination of outputs in the table on the first page.
    - Use your debugging skills to identify unexpected behavior and fix your program
    - Comment your code as you go, explaining what each event handler does
- Check the Help & Tips tab for ideas about Programming Patterns you can use
- Extension Ideas
    - Add another discount code for a different day of the week or age range
    - Make guests 65 and older always pay $5 unless they use a discount code

Step 4 - Submit

Before you submit, check the rubric below to make sure your program meets the requirements of the task.

Category	Extensive Evidence	Convincing Evidence	Limited Evidence	No Evidence
Input	onEvents are created for all the required inputs.	onEvents are created for most of the inputs.	onEvents are created for some of the inputs.	onEvents are not created for any inputs.
Storage: Variables	Variables are created and appropriately used for all pieces of information used in the app.	Most information is stored in a variable and appropriately updated throughout the app.	Some information is stored in a variable and appropriately updated throughout the app.	There are no variables which store the necessary information for the app to work correctly.
Processing: Conditional Logic	The code correctly determines the price for all combinations of inputs (age, price, discount code).	The code correctly determines the price for most but not all combinations of inputs (age, price, discount code).	The code correctly determines the price for some but not all combinations of inputs (age, price, discount code).	The code does not correctly determine the price for any combination of inputs (age, price, discount code).
Code: Output	The screen correctly displays the day, age, and price of the ticket.	The screen displays most but not all information correctly in the ticket.	The screen displays some but not all information correctly in the ticket.	The screen does not correctly display any information in the ticket.
Code runs without errors.	No errors are present in the required code.	On or two errors are present in the required code.	Three or four errors are present in the required code.	More than four errors are present in the required code.
Coding Comments	Comments are used to correctly explain the purpose and function of all onEvents and conditional logic.	Comments are used to explain the purpose and function of most onEvents and conditional logic.	Comments are used to explain the purpose and function of some onEvents and conditional logic.	Comments are not present.

Variables, Conditionals, and Functions ('21-'22)

Functions Explore / Investigate

Resources

Name(s)_____ Period _____ Date _____

Activity Guide - Song Lyrics

Style 1	Style 2
00 Feeling my way through the darkness 01 Guided by a beating heart 02 I can't tell where the journey will end 03 But I know where to start 04 05 They tell me I'm too young to understand 06 They say I'm caught up in a dream 07 Well life will pass me by if I don't open up my eyes 08 Well that's fine by me 09 10 So wake me up when it's all over 11 When I'm wiser and I'm older 12 All this time I was finding myself 13 And I didn't know I was lost 14 15 So wake me up when it's all over 16 When I'm wiser and I'm older 17 All this time I was finding myself 18 And I didn't know I was lost 19 20 I tried carrying the weight of the world 21 But I only have two hands 22 I hope I get the chance to travel the world 23 But I don't have any plans 24 I wish that I could stay forever this young 25 Not afraid to close my eyes 26 Life's a game made for everyone 27 And love is the prize 28 29 So wake me up when it's all over 30 When I'm wiser and I'm older 31 All this time I was finding myself 32 And I didn't know I was lost 33 34 So wake me up when it's all over 35 When I'm wiser and I'm older 36 All this time I was finding myself 37 And I didn't know I was lost 38 39 I didn't know I was lost 40 I didn't know I was lost 41 I didn't know I was lost 42 I didn't know I was lost 43 44 So wake me up when it's all over 45 When I'm wiser and I'm older 46 All this time I was finding myself 47 And I didn't know I was lost	00 Feeling my way through the darkness 01 Guided by a beating heart 02 I can't tell where the journey will end 03 But I know where to start 04 05 They tell me I'm too young to understand 06 They say I'm caught up in a dream 07 Well life will pass me by if I don't open up my eyes 08 Well that's fine by me 09 10 Sing Chorus (the lyrics are below) 11 12 Sing Chorus (the lyrics are below) 13 14 I tried carrying the weight of the world 15 But I only have two hands 16 I hope I get the chance to travel the world 17 But I don't have any plans 18 19 I wish that I could stay forever this young 20 Not afraid to close my eyes 21 Life's a game made for everyone 22 And love is the prize 23 24 Sing Chorus (the lyrics are below) 25 26 Sing Chorus (the lyrics are below) 27 28 I didn't know I was lost 29 I didn't know I was lost 30 I didn't know I was lost 31 I didn't know I was lost 32 33 Sing Chorus (the lyrics are below) 34 35 Chorus Lyrics: 36 So wake me up when it's all over 37 When I'm wiser and I'm older 38 All this time I was finding myself 39 And I didn't know I was lost

Variables, Conditionals, and Functions ('21-'22)

Functions Make

Resources

Name(s)_____ Period _____ Date _____

Activity Guide - Functions Make

Step 1 - Try the app

- Try many of the different options.
- Pay attention to what is happening on the screen when you move the slider or choose an item from the dropdown.
- When does the screen update?
- What happens if you choose lavender and Lucinda Sans from the dropdowns? Try choosing lightreen and moving the slider until you receive feedback.

Discuss with a Partner

- What does this app do?
- What are the inputs?
- What are the outputs?
- How could a function be used in this app?

Step 2 - Plan

Variables: Fill in the table below for each variable you'll need to create.

Variable Name	What the Variable Stores
color	*the background color the user selects*
fontFamily	

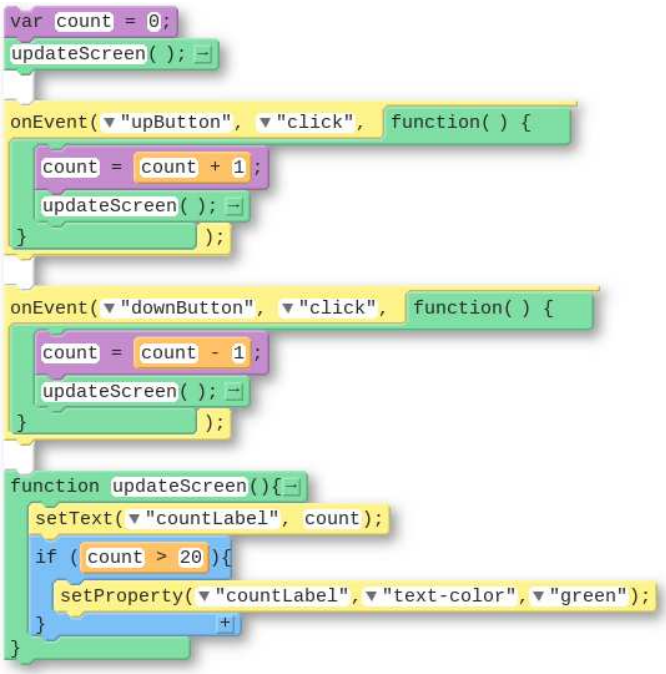
Conditionals: An if-else-if statement is used to check if certain options have been selected. Set up the conditional below using the variables you created above.

- Note: You can be creative here! Choose your own combinations and feedback messages.

```
if (color == _____ && fontFamily == _____ ){
    setText("feedbackOutput", _____);
} else if ( _____ == _____ && _____ == _____ ){
    setText("feedbackOutput", _____);
} else {
    setText("feedbackOutput", _____);
}
```

Functions: Consider what should be included in a function that updates the screen. Write out your plan below.
Things to think about:

- What elements on the screen need to be updated using the variables above?
- Does the conditional above belong in the function? Why or why not?
- When will the function be called?

	Review the `updateScreen() Pattern` to help you plan your function. 

Inputs: What are the inputs for the app? These will all be turned into onEvents.

Input	Action	Result
"quoteInput"	*input*	*The text on the screen appears, one character at a time as it's typed.*
"fontFamilyInput"	*change*	

Step 3 - Write Your Code

- Write the code for the app, using your plan above and the comments provided in Code Studio to help
- Step You Can Follow
    - Create all the variables from your table above.
    - Give your variables a starting value using the assignment operator (=)
    - Create a conditional that checks if various options are selected.
    - Create a function that updates the screen.
    - Create event handlers (onEvent) for the inputs in your table above
    - Use your debugging skills to identify unexpected behavior and fix your program
    - Comment your code as you go, explaining what each event handler and function does
- Extension Ideas
    - Create a dropdown with image names and decorate your quote!

Step 4 - Submit

Before you submit, check the rubric below to make sure your program meets the requirements of the task.

Category	Extensive Evidence	Convincing Evidence	Limited Evidence	No Evidence
Input	onEvents are created for all the required inputs.	onEvents are created for most of the inputs.	onEvents are created for some of the inputs.	onEvents are not created for any inputs.
Storage: Variables	Variables are created and appropriately used for all pieces of information used in the app.	Most information is stored in a variable and appropriately updated throughout the app.	Some information is stored in a variable and appropriately updated throughout the app.	There are no variables which store the necessary information for the app to work correctly.
Code: Conditionals	An if-else-if statement is used which correctly checks if certain options have been selected and displays feedback.	An if-else-if statement is used that partially checks if certain options have been selected and displays feedback.	An if-else statement or an if statement is used that checks if one option has been selected.	No conditional is present.
Code: Functions	A function is used which correctly updates all output elements. The function is called in all onEvents.	A function is used which correctly updates most of the output elements. The function is called in all onEvents.	A function is used which updates some of the output elements or the function is only called in some onEvents.	There is no function which updates the screen.
Code runs without errors.	No errors are present in the required code.	One or two errors are present in the required code.	Three or four errors are present in the required code.	More than four errors are present in the required code.
Coding Comments	Comments are used to correctly explain the purpose and function of all onEvents and functions.	Comments are used to explain the purpose and function of most onEvents and functions.	Comments are used to explain the purpose and function of some onEvents and functions.	Comments are not present.

Variables, Conditionals, and Functions ('21-'22)

Project Decision Maker App Part 1

Resources

Name(s)_____ Period _____ Date _____

U4 Practice PT Rubric

Rubric

Category	Convincing Evidence	Approaching Evidence	Limited Evidence	No Evidence
App Development Planning Guide:	Planning guide is fully completed.	Planning guide is mostly completed.	Planning guide is somewhat complete.	Planning guide is not complete.
Video	Video shows the program running including input, program functionality, and output.	Video shows the program running and two of the following: input, program functionality, and output.	Video shows the program running and one of the following: input, program functionality, or output.	No video was made.
Written Response 1:	Response accurately describes the purpose, functionality, and inputs/outputs of the app.	Response describes the purpose and functionality, or the inputs/outputs of the app.	Response partially describes the purpose and functionality, or the inputs/outputs of the app.	Response does not describe the purpose, functionality, and inputs/outputs of the app.
Written Response 2:	Response clearly describes an idea or recommendation provided by a partner / peer and how it improved the app.	Response describes an idea or recommendation provided by a partner / peer and how it improved the app, lacking clarity.	Response describes an idea or recommendation provided by a partner, but does not explain how it improved the app.	Response does not describe an idea or recommendation provided by a partner.
User Interface:	The User Interface is easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is mostly easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is lacking in some readability or it's not clear how to use the app.	The User Interface is difficult to navigate and it's not clear how the app is designed to be used. Text is unreadable.
Code: Warnings & Error Messages	No warnings or error messages appear when the app is run.	A few warnings or error messages appear when the app is run..	Many warnings or error messages appear when the app is run.	The app does not run at all.
Code: Variables	At least one number and one String are each stored in a variable and used to make a decision.	One data type (numbers or Strings) is stored in at least two variables and used to make a decision.	One variable stores either a number or String and is used to make a decision.	No variables are set up or used to make a decision.
Code: Function	A function is used to update the screen. The function is called at least two times in the program.	A function is used to update the screen. The function is called one time in the program.	A function is created to update the screen but is not called in the program.	A function was not created to update the screen.
Code: Conditional	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional correctly uses a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least three different responses that could be displayed.	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional does not correctly use a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least two different responses that could be displayed.	A conditional is created inside of the function, but does not use information stored in variables to make a decision or display it on the screen.	No conditionals are present in the function.
Code: Comments	The update screen function has a comment which clearly explains its purpose and functionality.	The update screen function has a comment which clearly explains its purpose or functionality.	A comment is present, but it does not clearly explain anything about the function.	No comments are present.

Name(s)_____ Period _____ Date _____

U4 Practice PT - Decision Maker App Planning Guide

Project Description

For this project you will create an app that helps a user make a decision. Your app must take in at least one number and one string from the user that will help to make the decision. All of this information will be used as part of the decision making process. In addition, your code must include at least one function used to update the screen.

You will submit

- Your final app
- A video that shows your program running
- This completed project-planning guide

App Requirements

- At least one number and one string used to make and report a decision with a conditional statement
- A function which updates the screen and is called at least twice in the program
- Conditional statement includes at least one logical operator (`&&`, `||` or `!`)
- There are at least three different possible output answers (i.e. “Yes, you can adopt a cat!”, “No, you can’t adopt a cat”, and “Congratulations, you can adopt a kitten!).
- Every function contains a comment explaining purpose and functionality
- Clear and easy to navigate user interface
- Cleanly written code which is free of errors

Steps

- Brainstorm an app idea for making a decision
- Interview classmates for ideas on what information would be needed to make the decision
- Draft a flowchart of the decision making process
- Design your app’s user interface
- Design and program your app in App Lab
- Collect feedback from your classmates and update your app
- Record a screen capture of your app being used
- Submit your final app

Investigate

Step 1. Brainstorm App Ideas: Your app should be designed to help a user make a decision. For this project your user is your classmate. The decision can be small or big, like what to eat for lunch or where to apply for a job. Keep in mind how your idea might help solve a problem for your user.

Idea 1:

_____

Idea 2:

_____

Idea 3:

_____

Step 2. Choose One Idea: Talk through your ideas with a classmate. Pick the one that you are most interested in.

App Idea: _____

Step 3. Survey Your Classmates: To design your app you'll need to understand your users. For this project your user is your classmate, and you'll need to understand what information will be needed to make the decision.

Find two classmates and talk to them about your topic for a couple minutes. Then fill in this table.

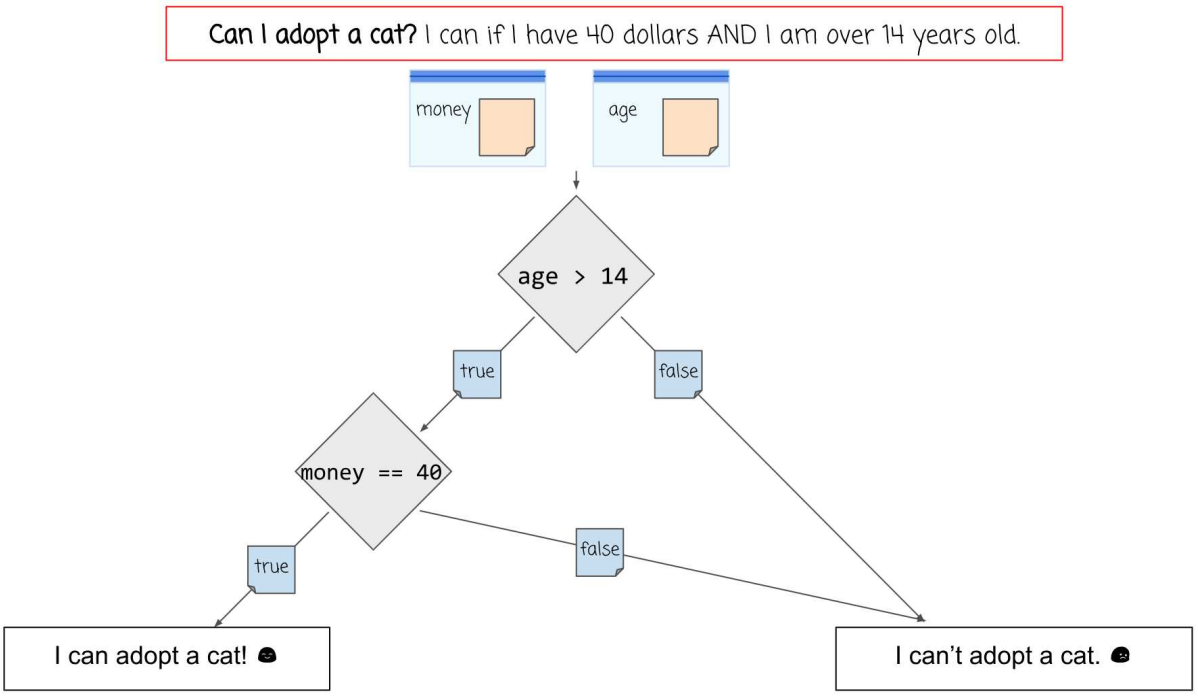
Name	What information is needed to make this decision?

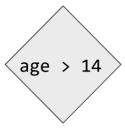
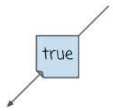
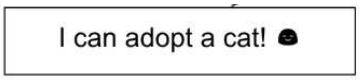
Step 4: Storing information: What variables will be used to store information?

Name	Information Stored	Variable Type (string, number, Boolean)
Ex: age	*The age of the user*	*number*

Step 5: Flowchart: Draft a flowchart to show the decision making process

Example:

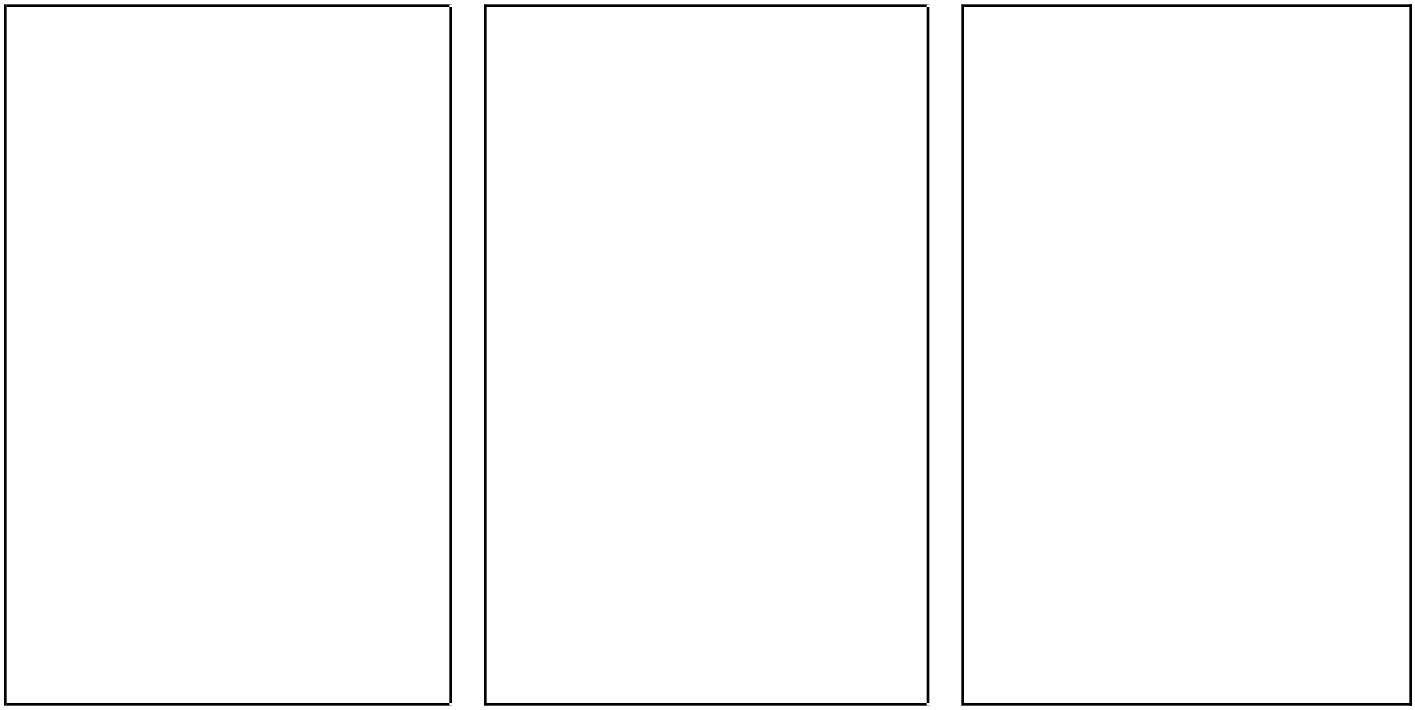


Component	Purpose
	Start your flowchart with a question. What decision are you trying to make?
	Baggies are used to represent the variables which store information. In your flowchart, draw a small rectangle for the variables.
	A diamond represents a decision point, based on the original question. Write the Boolean expression that will be used to determine the answer.
	True and False arrows designate the paths taken, based on the result of a decision (diamond). Note that every decision may have only 2 possible paths that result from it, one for true and one for false.
	A rectangle at the end of a decision path represents a possible result.
	A **simple arrow** indicates that we are moving from one action to the next without considering any decision. These will generally be used to link a set of actions to be completed one after the other.
Flowchart	

Design

Step 6. Design User Interface: In the space below draw a rough sketch of your user interface. This means you should include all the buttons, text, and images that the user will be able to use. Write notes or draw arrows showing how different user interface elements should work.

Note: There are no screen requirements for this app - you may use one or more screens.



Prototype

Step 7. Start Building Your App: Build your app. Along the way make sure you:

- Use the design you drew as a starting point, but it's OK to update as you go.
- Reference the flow chart when setting up your conditional statements
- Use your debugging skills to check that your app is working
- Comment all functions explaining purpose (what does it do) and functionality (how does it work)

Test

Step 8. Testing: You will need to test your app to make sure it works as expected. To do that find at least two classmates to use your app. While they use the app watch them to see if anything is broken or confusing. Afterwards ask them to share any specific improvements they'd like to see.

Name	Things that could be improved based on watching them use the app	Improvements this person recommends

Step 9. Pick Improvements: Pick at least one improvement you plan to make to your app based on feedback you collected from your classmate.

Improvement 1:

_____

Improvement 2 (Optional):

_____

Step 10: Complete Your App: Finish your app!

Step 11: Record Video. Record a video that demonstrates the running of your program as described below. Your video may NOT contain voice narration and must be no more than 1 minute in length

Your video must demonstrate your program running, including:

- ☐ Input to your program
- ☐ At least one aspect of the functionality of your program
- ☐ Output produced by your program

Question 1: Provide a written response that:

- describes the overall purpose of the program
- describes the functionality of your app
- describes the input and outputs of your app

(Approx 150 words)

Question 2: This project was created using a development process that required you to incorporate the ideas of your partner and feedback from your classmates. Provide a written response that describes one part of your app that was improved through input from EITHER your partner or feedback you received from classmates. Include:

- Who specifically provided the idea or recommendation
- What their idea or recommendation was
- The specific change you made to your app's user interface or functionality in response to the recommendation
- How you believe this change improved your app

(Approx 150 words)

Rubric

Category	Convincing Evidence	Approaching Evidence	Limited Evidence	No Evidence
App Development Planning Guide:	Planning guide is fully completed.	Planning guide is mostly completed.	Planning guide is somewhat complete.	Planning guide is not complete.
Video	Video shows the program running including input, program functionality, and output.	Video shows the program running and two of the following: input, program functionality, and output.	Video shows the program running and one of the following: input, program functionality, or output.	No video was made.
Written Response 1:	Response accurately describes the purpose, functionality, and inputs/outputs of the app.	Response describes the purpose and functionality, or the inputs/outputs of the app.	Response partially describes the purpose and functionality, or the inputs/outputs of the app.	Response does not describe the purpose, functionality, and inputs/outputs of the app.
Written Response 2:	Response clearly describes an idea or recommendation provided by a partner / peer and how it improved the app.	Response describes an idea or recommendation provided by a partner / peer and how it improved the app, lacking clarity.	Response describes an idea or recommendation provided by a partner, but does not explain how it improved the app.	Response does not describe an idea or recommendation provided by a partner.
User Interface:	The User Interface is easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is mostly easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is lacking in some readability or it's not clear how to use the app.	The User Interface is difficult to navigate and it's not clear how the app is designed to be used. Text is unreadable.
Code: Warnings & Error Messages	No warnings or error messages appear when the app is run.	A few warnings or error messages appear when the app is run..	Many warnings or error messages appear when the app is run.	The app does not run at all.
Code: Variables	At least one number and one String are each stored in a variable and used to make a decision.	One data type (numbers or Strings) is stored in at least two variables and used to make a decision.	One variable stores either a number or String and is used to make a decision.	No variables are set up or used to make a decision.
Code: Function	A function is used to update the screen. The function is called at least two times in the program.	A function is used to update the screen. The function is called one time in the program.	A function is created to update the screen but is not called in the program.	A function was not created to update the screen.
Code: Conditional	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional correctly uses a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least three different responses that could be displayed.	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional does not correctly use a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least two different responses that could be displayed.	A conditional is created inside of the function, but does not use information stored in variables to make a decision or display it on the screen.	No conditionals are present in the function.
Code: Comments	The update screen function has a comment which clearly explains its purpose and functionality.	The update screen function has a comment which clearly explains its purpose or functionality.	A comment is present, but it does not clearly explain anything about the function.	No comments are present.

Variables, Conditionals, and Functions ('21-'22)

Project Decision Maker App Part 2

Resources

Name(s)_____ Period _____ Date _____

U4 Practice PT Rubric

Rubric

Category	Convincing Evidence	Approaching Evidence	Limited Evidence	No Evidence
App Development Planning Guide:	Planning guide is fully completed.	Planning guide is mostly completed.	Planning guide is somewhat complete.	Planning guide is not complete.
Video	Video shows the program running including input, program functionality, and output.	Video shows the program running and two of the following: input, program functionality, and output.	Video shows the program running and one of the following: input, program functionality, or output.	No video was made.
Written Response 1:	Response accurately describes the purpose, functionality, and inputs/outputs of the app.	Response describes the purpose and functionality, or the inputs/outputs of the app.	Response partially describes the purpose and functionality, or the inputs/outputs of the app.	Response does not describe the purpose, functionality, and inputs/outputs of the app.
Written Response 2:	Response clearly describes an idea or recommendation provided by a partner / peer and how it improved the app.	Response describes an idea or recommendation provided by a partner / peer and how it improved the app, lacking clarity.	Response describes an idea or recommendation provided by a partner, but does not explain how it improved the app.	Response does not describe an idea or recommendation provided by a partner.
User Interface:	The User Interface is easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is mostly easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is lacking in some readability or it's not clear how to use the app.	The User Interface is difficult to navigate and it's not clear how the app is designed to be used. Text is unreadable.
Code: Warnings & Error Messages	No warnings or error messages appear when the app is run.	A few warnings or error messages appear when the app is run..	Many warnings or error messages appear when the app is run.	The app does not run at all.
Code: Variables	At least one number and one String are each stored in a variable and used to make a decision.	One data type (numbers or Strings) is stored in at least two variables and used to make a decision.	One variable stores either a number or String and is used to make a decision.	No variables are set up or used to make a decision.
Code: Function	A function is used to update the screen. The function is called at least two times in the program.	A function is used to update the screen. The function is called one time in the program.	A function is created to update the screen but is not called in the program.	A function was not created to update the screen.
Code: Conditional	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional correctly uses a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least three different responses that could be displayed.	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional does not correctly use a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least two different responses that could be displayed.	A conditional is created inside of the function, but does not use information stored in variables to make a decision or display it on the screen.	No conditionals are present in the function.
Code: Comments	The update screen function has a comment which clearly explains its purpose and functionality.	The update screen function has a comment which clearly explains its purpose or functionality.	A comment is present, but it does not clearly explain anything about the function.	No comments are present.

Name(s)_____ Period _____ Date _____

U4 Practice PT - Decision Maker App Planning Guide

Project Description

For this project you will create an app that helps a user make a decision. Your app must take in at least one number and one string from the user that will help to make the decision. All of this information will be used as part of the decision making process. In addition, your code must include at least one function used to update the screen.

You will submit

- Your final app
- A video that shows your program running
- This completed project-planning guide

App Requirements

- At least one number and one string used to make and report a decision with a conditional statement
- A function which updates the screen and is called at least twice in the program
- Conditional statement includes at least one logical operator (`&&`, `||` or `!`)
- There are at least three different possible output answers (i.e. “Yes, you can adopt a cat!”, “No, you can’t adopt a cat”, and “Congratulations, you can adopt a kitten!).
- Every function contains a comment explaining purpose and functionality
- Clear and easy to navigate user interface
- Cleanly written code which is free of errors

Steps

- Brainstorm an app idea for making a decision
- Interview classmates for ideas on what information would be needed to make the decision
- Draft a flowchart of the decision making process
- Design your app’s user interface
- Design and program your app in App Lab
- Collect feedback from your classmates and update your app
- Record a screen capture of your app being used
- Submit your final app

Investigate

Step 1. Brainstorm App Ideas: Your app should be designed to help a user make a decision. For this project your user is your classmate. The decision can be small or big, like what to eat for lunch or where to apply for a job. Keep in mind how your idea might help solve a problem for your user.

Idea 1:

_____

Idea 2:

_____

Idea 3:

_____

Step 2. Choose One Idea: Talk through your ideas with a classmate. Pick the one that you are most interested in.

App Idea: _____

Step 3. Survey Your Classmates: To design your app you'll need to understand your users. For this project your user is your classmate, and you'll need to understand what information will be needed to make the decision.

Find two classmates and talk to them about your topic for a couple minutes. Then fill in this table.

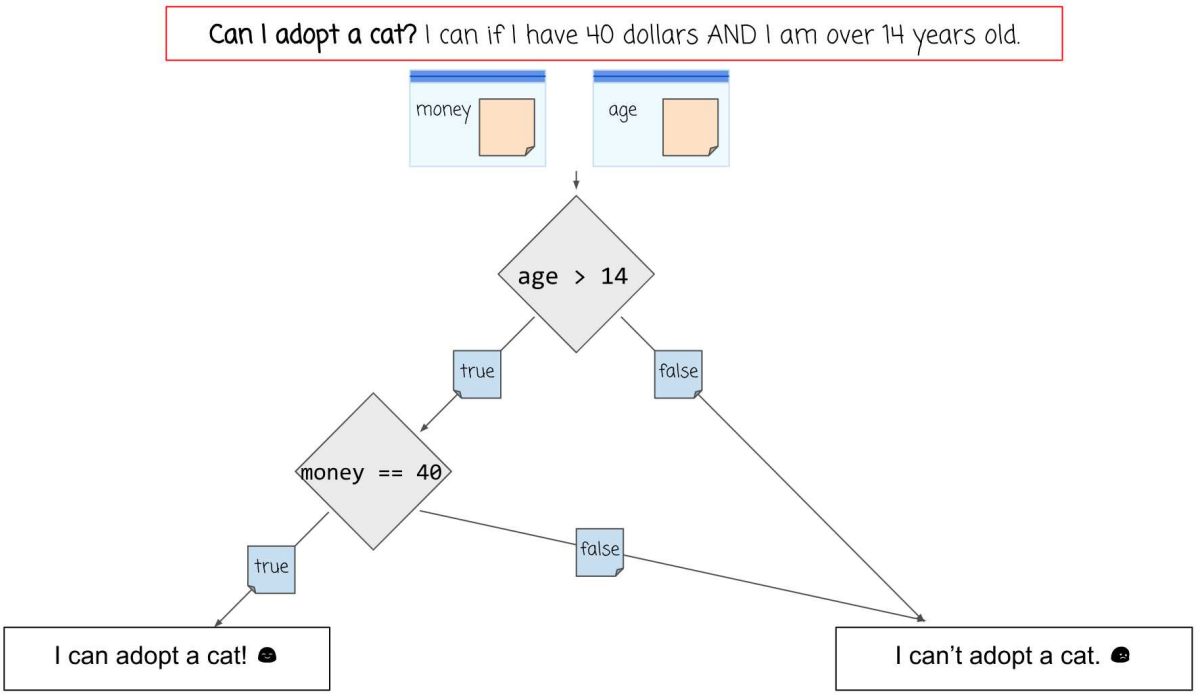
Name	What information is needed to make this decision?

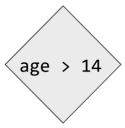
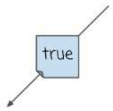
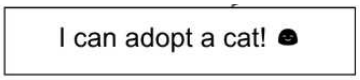
Step 4: Storing information: What variables will be used to store information?

Name	Information Stored	Variable Type (string, number, Boolean)
Ex: age	*The age of the user*	*number*

Step 5: Flowchart: Draft a flowchart to show the decision making process

Example:

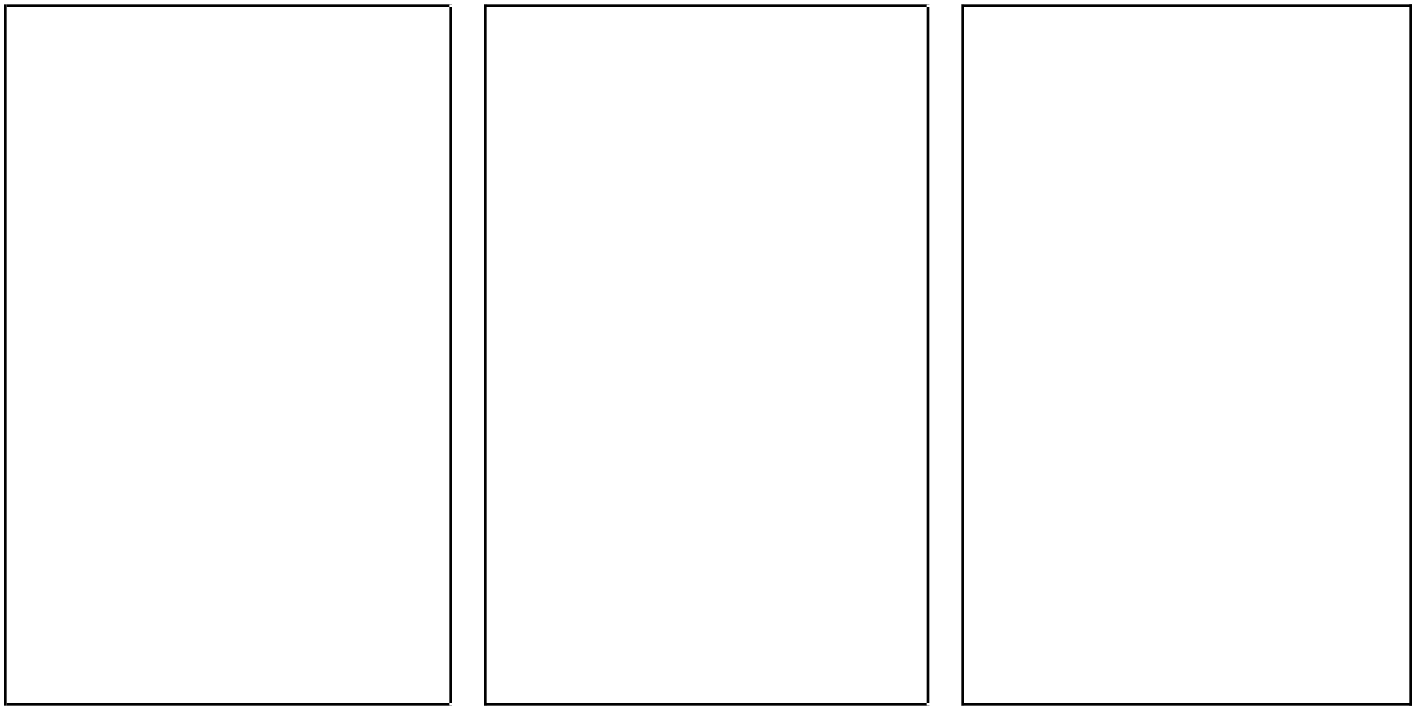


Component	Purpose
	Start your flowchart with a question. What decision are you trying to make?
	Baggies are used to represent the variables which store information. In your flowchart, draw a small rectangle for the variables.
	A diamond represents a decision point, based on the original question. Write the Boolean expression that will be used to determine the answer.
	True and False arrows designate the paths taken, based on the result of a decision (diamond). Note that every decision may have only 2 possible paths that result from it, one for true and one for false.
	A rectangle at the end of a decision path represents a possible result.
	A **simple arrow** indicates that we are moving from one action to the next without considering any decision. These will generally be used to link a set of actions to be completed one after the other.
Flowchart	

Design

Step 6. Design User Interface: In the space below draw a rough sketch of your user interface. This means you should include all the buttons, text, and images that the user will be able to use. Write notes or draw arrows showing how different user interface elements should work.

Note: There are no screen requirements for this app - you may use one or more screens.



Prototype

Step 7. Start Building Your App: Build your app. Along the way make sure you:

- Use the design you drew as a starting point, but it's OK to update as you go.
- Reference the flow chart when setting up your conditional statements
- Use your debugging skills to check that your app is working
- Comment all functions explaining purpose (what does it do) and functionality (how does it work)

Test

Step 8. Testing: You will need to test your app to make sure it works as expected. To do that find at least two classmates to use your app. While they use the app watch them to see if anything is broken or confusing. Afterwards ask them to share any specific improvements they'd like to see.

Name	Things that could be improved based on watching them use the app	Improvements this person recommends

Step 9. Pick Improvements: Pick at least one improvement you plan to make to your app based on feedback you collected from your classmate.

Improvement 1:

_____

Improvement 2 (Optional):

_____

Step 10: Complete Your App: Finish your app!

Step 11: Record Video. Record a video that demonstrates the running of your program as described below. Your video may NOT contain voice narration and must be no more than 1 minute in length

Your video must demonstrate your program running, including:

- ☐ Input to your program
- ☐ At least one aspect of the functionality of your program
- ☐ Output produced by your program

Question 1: Provide a written response that:

- describes the overall purpose of the program
- describes the functionality of your app
- describes the input and outputs of your app

(Approx 150 words)

Question 2: This project was created using a development process that required you to incorporate the ideas of your partner and feedback from your classmates. Provide a written response that describes one part of your app that was improved through input from EITHER your partner or feedback you received from classmates. Include:

- Who specifically provided the idea or recommendation
- What their idea or recommendation was
- The specific change you made to your app's user interface or functionality in response to the recommendation
- How you believe this change improved your app

(Approx 150 words)

Rubric

Category	Convincing Evidence	Approaching Evidence	Limited Evidence	No Evidence
App Development Planning Guide:	Planning guide is fully completed.	Planning guide is mostly completed.	Planning guide is somewhat complete.	Planning guide is not complete.
Video	Video shows the program running including input, program functionality, and output.	Video shows the program running and two of the following: input, program functionality, and output.	Video shows the program running and one of the following: input, program functionality, or output.	No video was made.
Written Response 1:	Response accurately describes the purpose, functionality, and inputs/outputs of the app.	Response describes the purpose and functionality, or the inputs/outputs of the app.	Response partially describes the purpose and functionality, or the inputs/outputs of the app.	Response does not describe the purpose, functionality, and inputs/outputs of the app.
Written Response 2:	Response clearly describes an idea or recommendation provided by a partner / peer and how it improved the app.	Response describes an idea or recommendation provided by a partner / peer and how it improved the app, lacking clarity.	Response describes an idea or recommendation provided by a partner, but does not explain how it improved the app.	Response does not describe an idea or recommendation provided by a partner.
User Interface:	The User Interface is easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is mostly easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is lacking in some readability or it's not clear how to use the app.	The User Interface is difficult to navigate and it's not clear how the app is designed to be used. Text is unreadable.
Code: Warnings & Error Messages	No warnings or error messages appear when the app is run.	A few warnings or error messages appear when the app is run..	Many warnings or error messages appear when the app is run.	The app does not run at all.
Code: Variables	At least one number and one String are each stored in a variable and used to make a decision.	One data type (numbers or Strings) is stored in at least two variables and used to make a decision.	One variable stores either a number or String and is used to make a decision.	No variables are set up or used to make a decision.
Code: Function	A function is used to update the screen. The function is called at least two times in the program.	A function is used to update the screen. The function is called one time in the program.	A function is created to update the screen but is not called in the program.	A function was not created to update the screen.
Code: Conditional	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional correctly uses a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least three different responses that could be displayed.	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional does not correctly use a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least two different responses that could be displayed.	A conditional is created inside of the function, but does not use information stored in variables to make a decision or display it on the screen.	No conditionals are present in the function.
Code: Comments	The update screen function has a comment which clearly explains its purpose and functionality.	The update screen function has a comment which clearly explains its purpose or functionality.	A comment is present, but it does not clearly explain anything about the function.	No comments are present.

Variables, Conditionals, and Functions ('21-'22)

Project Decision Maker App Part 3

Resources

Name(s)_____ Period _____ Date _____

U4 Practice PT Rubric

Rubric

Category	Convincing Evidence	Approaching Evidence	Limited Evidence	No Evidence
App Development Planning Guide:	Planning guide is fully completed.	Planning guide is mostly completed.	Planning guide is somewhat complete.	Planning guide is not complete.
Video	Video shows the program running including input, program functionality, and output.	Video shows the program running and two of the following: input, program functionality, and output.	Video shows the program running and one of the following: input, program functionality, or output.	No video was made.
Written Response 1:	Response accurately describes the purpose, functionality, and inputs/outputs of the app.	Response describes the purpose and functionality, or the inputs/outputs of the app.	Response partially describes the purpose and functionality, or the inputs/outputs of the app.	Response does not describe the purpose, functionality, and inputs/outputs of the app.
Written Response 2:	Response clearly describes an idea or recommendation provided by a partner / peer and how it improved the app.	Response describes an idea or recommendation provided by a partner / peer and how it improved the app, lacking clarity.	Response describes an idea or recommendation provided by a partner, but does not explain how it improved the app.	Response does not describe an idea or recommendation provided by a partner.
User Interface:	The User Interface is easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is mostly easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is lacking in some readability or it's not clear how to use the app.	The User Interface is difficult to navigate and it's not clear how the app is designed to be used. Text is unreadable.
Code: Warnings & Error Messages	No warnings or error messages appear when the app is run.	A few warnings or error messages appear when the app is run..	Many warnings or error messages appear when the app is run.	The app does not run at all.
Code: Variables	At least one number and one String are each stored in a variable and used to make a decision.	One data type (numbers or Strings) is stored in at least two variables and used to make a decision.	One variable stores either a number or String and is used to make a decision.	No variables are set up or used to make a decision.
Code: Function	A function is used to update the screen. The function is called at least two times in the program.	A function is used to update the screen. The function is called one time in the program.	A function is created to update the screen but is not called in the program.	A function was not created to update the screen.
Code: Conditional	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional correctly uses a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least three different responses that could be displayed.	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional does not correctly use a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least two different responses that could be displayed.	A conditional is created inside of the function, but does not use information stored in variables to make a decision or display it on the screen.	No conditionals are present in the function.
Code: Comments	The update screen function has a comment which clearly explains its purpose and functionality.	The update screen function has a comment which clearly explains its purpose or functionality.	A comment is present, but it does not clearly explain anything about the function.	No comments are present.

Name(s)_____ Period _____ Date _____

U4 Practice PT - Decision Maker App Planning Guide

Project Description

For this project you will create an app that helps a user make a decision. Your app must take in at least one number and one string from the user that will help to make the decision. All of this information will be used as part of the decision making process. In addition, your code must include at least one function used to update the screen.

You will submit

- Your final app
- A video that shows your program running
- This completed project-planning guide

App Requirements

- At least one number and one string used to make and report a decision with a conditional statement
- A function which updates the screen and is called at least twice in the program
- Conditional statement includes at least one logical operator (`&&`, `||` or `!`)
- There are at least three different possible output answers (i.e. “Yes, you can adopt a cat!”, “No, you can’t adopt a cat”, and “Congratulations, you can adopt a kitten!).
- Every function contains a comment explaining purpose and functionality
- Clear and easy to navigate user interface
- Cleanly written code which is free of errors

Steps

- Brainstorm an app idea for making a decision
- Interview classmates for ideas on what information would be needed to make the decision
- Draft a flowchart of the decision making process
- Design your app’s user interface
- Design and program your app in App Lab
- Collect feedback from your classmates and update your app
- Record a screen capture of your app being used
- Submit your final app

Investigate

Step 1. Brainstorm App Ideas: Your app should be designed to help a user make a decision. For this project your user is your classmate. The decision can be small or big, like what to eat for lunch or where to apply for a job. Keep in mind how your idea might help solve a problem for your user.

Idea 1:

_____

Idea 2:

_____

Idea 3:

_____

Step 2. Choose One Idea: Talk through your ideas with a classmate. Pick the one that you are most interested in.

App Idea: _____

Step 3. Survey Your Classmates: To design your app you'll need to understand your users. For this project your user is your classmate, and you'll need to understand what information will be needed to make the decision.

Find two classmates and talk to them about your topic for a couple minutes. Then fill in this table.

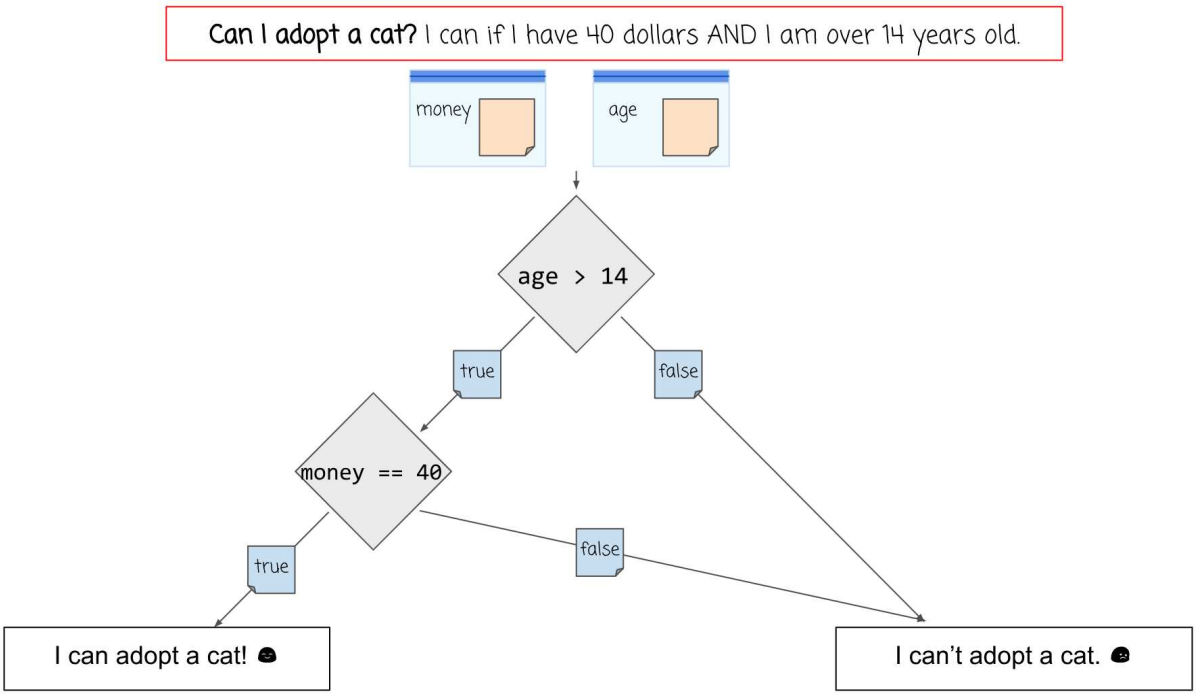
Name	What information is needed to make this decision?

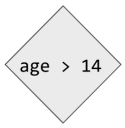
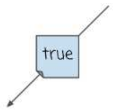
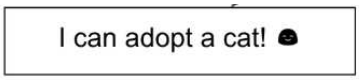
Step 4: Storing information: What variables will be used to store information?

Name	Information Stored	Variable Type (string, number, Boolean)
Ex: age	*The age of the user*	*number*

Step 5: Flowchart: Draft a flowchart to show the decision making process

Example:

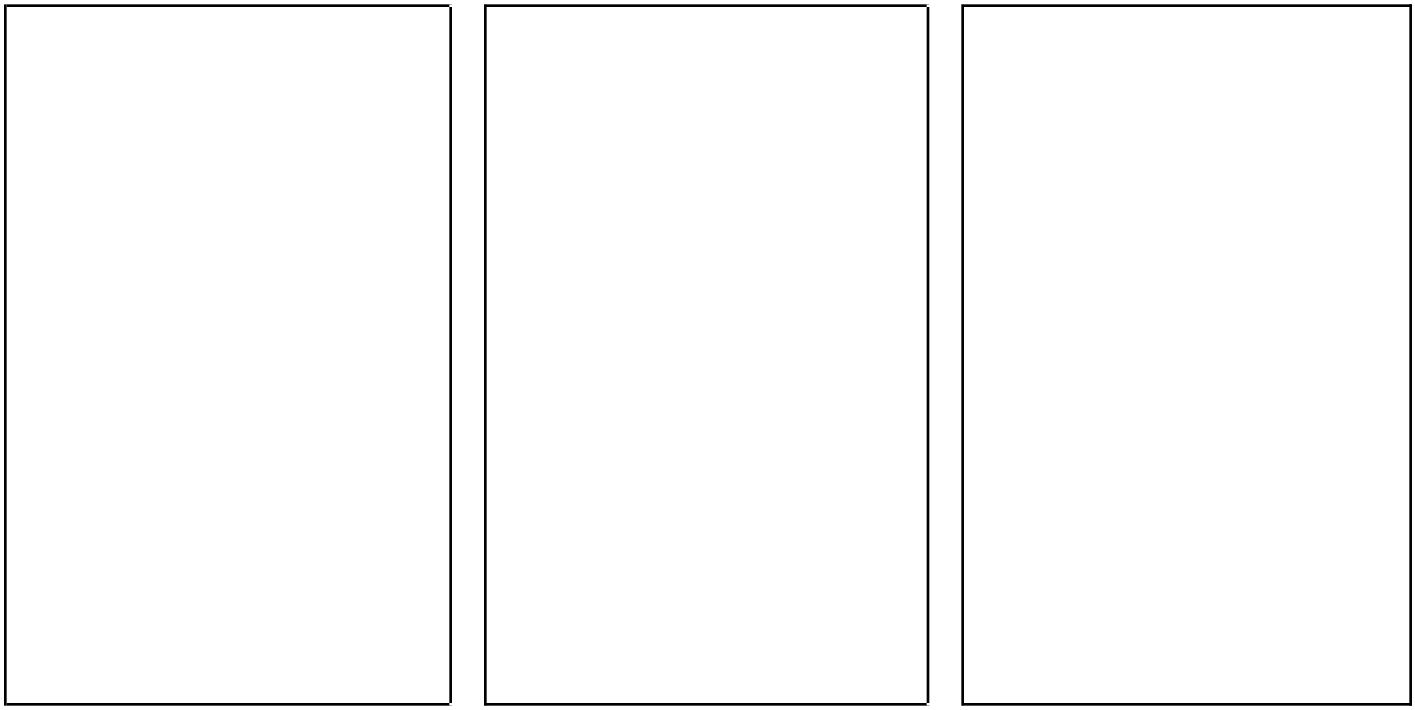


Component	Purpose
	Start your flowchart with a question. What decision are you trying to make?
	Baggies are used to represent the variables which store information. In your flowchart, draw a small rectangle for the variables.
	A diamond represents a decision point, based on the original question. Write the Boolean expression that will be used to determine the answer.
	True and False arrows designate the paths taken, based on the result of a decision (diamond). Note that every decision may have only 2 possible paths that result from it, one for true and one for false.
	A rectangle at the end of a decision path represents a possible result.
	A **simple arrow** indicates that we are moving from one action to the next without considering any decision. These will generally be used to link a set of actions to be completed one after the other.
Flowchart	

Design

Step 6. Design User Interface: In the space below draw a rough sketch of your user interface. This means you should include all the buttons, text, and images that the user will be able to use. Write notes or draw arrows showing how different user interface elements should work.

Note: There are no screen requirements for this app - you may use one or more screens.



Prototype

Step 7. Start Building Your App: Build your app. Along the way make sure you:

- Use the design you drew as a starting point, but it's OK to update as you go.
- Reference the flow chart when setting up your conditional statements
- Use your debugging skills to check that your app is working
- Comment all functions explaining purpose (what does it do) and functionality (how does it work)

Test

Step 8. Testing: You will need to test your app to make sure it works as expected. To do that find at least two classmates to use your app. While they use the app watch them to see if anything is broken or confusing. Afterwards ask them to share any specific improvements they'd like to see.

Name	Things that could be improved based on watching them use the app	Improvements this person recommends

Step 9. Pick Improvements: Pick at least one improvement you plan to make to your app based on feedback you collected from your classmate.

Improvement 1:

_____

Improvement 2 (Optional):

_____

Step 10: Complete Your App: Finish your app!

Step 11: Record Video. Record a video that demonstrates the running of your program as described below. Your video may NOT contain voice narration and must be no more than 1 minute in length

Your video must demonstrate your program running, including:

- ☐ Input to your program
- ☐ At least one aspect of the functionality of your program
- ☐ Output produced by your program

Question 1: Provide a written response that:

- describes the overall purpose of the program
- describes the functionality of your app
- describes the input and outputs of your app

(Approx 150 words)

Question 2: This project was created using a development process that required you to incorporate the ideas of your partner and feedback from your classmates. Provide a written response that describes one part of your app that was improved through input from EITHER your partner or feedback you received from classmates. Include:

- Who specifically provided the idea or recommendation
- What their idea or recommendation was
- The specific change you made to your app's user interface or functionality in response to the recommendation
- How you believe this change improved your app

(Approx 150 words)

Rubric

Category	Convincing Evidence	Approaching Evidence	Limited Evidence	No Evidence
App Development Planning Guide:	Planning guide is fully completed.	Planning guide is mostly completed.	Planning guide is somewhat complete.	Planning guide is not complete.
Video	Video shows the program running including input, program functionality, and output.	Video shows the program running and two of the following: input, program functionality, and output.	Video shows the program running and one of the following: input, program functionality, or output.	No video was made.
Written Response 1:	Response accurately describes the purpose, functionality, and inputs/outputs of the app.	Response describes the purpose and functionality, or the inputs/outputs of the app.	Response partially describes the purpose and functionality, or the inputs/outputs of the app.	Response does not describe the purpose, functionality, and inputs/outputs of the app.
Written Response 2:	Response clearly describes an idea or recommendation provided by a partner / peer and how it improved the app.	Response describes an idea or recommendation provided by a partner / peer and how it improved the app, lacking clarity.	Response describes an idea or recommendation provided by a partner, but does not explain how it improved the app.	Response does not describe an idea or recommendation provided by a partner.
User Interface:	The User Interface is easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is mostly easy to navigate and it's clear how the app is designed to be used. All text is readable.	The User Interface is lacking in some readability or it's not clear how to use the app.	The User Interface is difficult to navigate and it's not clear how the app is designed to be used. Text is unreadable.
Code: Warnings & Error Messages	No warnings or error messages appear when the app is run.	A few warnings or error messages appear when the app is run..	Many warnings or error messages appear when the app is run.	The app does not run at all.
Code: Variables	At least one number and one String are each stored in a variable and used to make a decision.	One data type (numbers or Strings) is stored in at least two variables and used to make a decision.	One variable stores either a number or String and is used to make a decision.	No variables are set up or used to make a decision.
Code: Function	A function is used to update the screen. The function is called at least two times in the program.	A function is used to update the screen. The function is called one time in the program.	A function is created to update the screen but is not called in the program.	A function was not created to update the screen.
Code: Conditional	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional correctly uses a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least three different responses that could be displayed.	A conditional is used inside of the function to make a decision based on information stored in variables. The conditional does not correctly use a logical operator (&&, ||, or !) in the Boolean expression. The decision is displayed on the screen. There are at least two different responses that could be displayed.	A conditional is created inside of the function, but does not use information stored in variables to make a decision or display it on the screen.	No conditionals are present in the function.
Code: Comments	The update screen function has a comment which clearly explains its purpose and functionality.	The update screen function has a comment which clearly explains its purpose or functionality.	A comment is present, but it does not clearly explain anything about the function.	No comments are present.