

## Activity 1.1.6

# Buggy Image

## Distance Learning Support

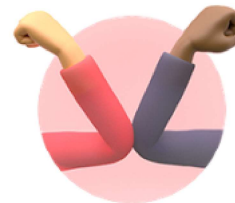
Check with your teacher about:

- ☐ What work you need to turn in and how to submit it
- ☐ Collaboration strategies
- ☐ If you are using a Chromebook, open a [blank code editor](#).



## GOALS

- Examine a program with poorly written code and determine what it is supposed to do.
- Use meaningful variable names to indicate their value and use.
- Apply methods you have already learned to improve buggy code.
- Correct errors using test cases; hand-tracing; visualizations; debuggers; and/or adding extra output statements.



## RESOURCES



Python Keywords and Built-ins



Debugging Tips



# Buggy Code

Imperfections and errors in code are known as **bugs** 🗨️. You probably have experienced the consequences of *buggy* code yourself. You may have been annoyed at a small inconvenience in an app, a glitch in a major program, or perhaps you even saw a problem that caused an entire computer to crash.

Bugs are caused by a number of factors. Perhaps the biggest reason we have buggy code is unavoidable: people are not perfect and therefore, their programs are not perfect. Other reasons *are* avoidable, such as sloppy programming and poorly tested code. We can reduce bugs in code by having good programming habits and testing code thoroughly. In this activity, you will **debug** 🗨️ a program that has multiple problems including poorly named variables, missing comments, and incorrect code. You will then make improvements to the overall program.




## CAREER CONNECTIONS


### Quality Assurance

Debugging and testing software is an important part of producing a high-quality product. Not only do programmers debug their own work as they develop programs, but they also help debug each other's code. Software developers often begin their career checking the quality of their colleague's code. This is called quality assurance, or QA. “QAing” always involves testing and often involves bug fixes. It is a great opportunity for new developers to learn the software and the programming language, get to know the team members, and see other aspects of the software development process.

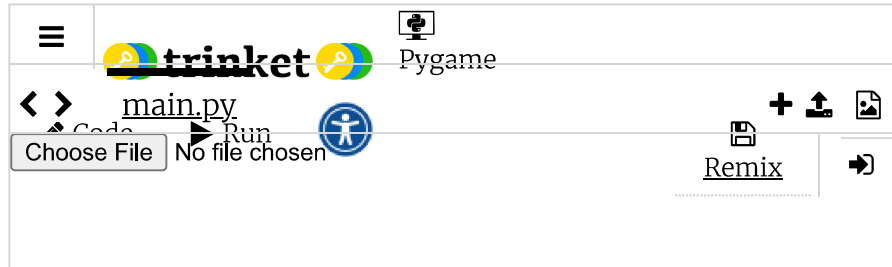
## Understand Your Code

In order to fix bugs in a program, you need to understand all of the code in it. But this can be a daunting task, especially in large programs. A good way to learn what a program does is to become familiar with its variables.

Recall that variables hold values such as numbers and strings. Bugs in programs are often related to the changing values of variables, and new programmers often struggle with how the values change throughout the code. Since programs are executed **sequentially** , from the first statement to the last, you can predict variable values.

- 1 Examine how the variables are **initialized**  and updated throughout this small program. To do this, you can hand trace a program, recording the values of variables in a table. Hand tracing a program with a trace table allows you to follow your

program, step by step, and record the values of each variable. Create a trace table for the following small program. Hand trace the values of each variable, recording how they change (or don't change) with each line of code.



Statements	x	y
x = 8	<input type="text"/>	undefined
y = 40 / x	<input type="text"/>	<input type="text"/>
x = 10	<input type="text"/>	<input type="text"/>

### Check your response

Many new programmers might have thought that the value of `y` was 4 because the last value of `x` was 10 and  $40/10$  is 4. But as you can see from the trace table, *at the time* that `y` was calculated, the value of `x` was 8. Therefore, the value of `y` is  $40/8$  or 5.

- 2 Use a `print` statement at line 4 to show the value of `y`. Does this value match the predicted value from your trace table?
- 3 Using new values for `x` and `y`, create a trace table to predict the value of `y`.

Statements	x	y
$x = 4$	<input type="text"/>	<i>undefined</i>
$y = 48 / x$	<input type="text"/>	<input type="text"/>
$x = 8$	<input type="text"/>	<input type="text"/>

4

Make the above changes in your code editor and observe the value of  $y$ . Was your prediction correct?

**Reminder:** The code in your program is executed sequentially and changing a variable's value does not affect previously executed code.

## Explore a Poorly Written Program

To get started on your debugging session, start by exploring some code that is intentionally poorly written.

5

Copy and paste the following code into VS Code and name the file `a116_buggy_image_[studentinitials].py`. You will be making

all of your changes in VS Code.

```

1 # a116_buggy_image.py
2 import turtle as trtl
3 # instead of a descriptive name of the turtle such as painter,
4 # a less useful variable name x is used
5 x = trtl.Turtle()
6 x.pensize(40)
7 x.circle(20)
8 w = 6
9 y = 70
10 z = 380 / w
11 x.pensize(5)
12 n = 0
13 while (n < w):
14     x.goto(0,0)
15     x.setheading(z*n)
16     x.forward(y)
17     n = n + 1
18 x.hideturtle()
19 wn = trtl.Screen()

```

6

Do your best to explore the code. If you cannot easily understand it, don't worry. The poorly chosen variable names and the lack of comments give you no clues as to what is going on. This code is difficult to understand or [interpret](#).


7

Run the program to see if that gives you any clues. Most likely, it is still unclear what the image is supposed to be. If you were in a software development organization, you could go to the original writers and ask for clarification. Assume you did this and their response was, "It is supposed to be a spider!"

What are some problems you see with the image that is created?

### Problems with the Program

## Investigate with Block Strings

One approach to help you understand a program is to break it into smaller chunks. To help you see what each chunk does, you can temporarily “deactivate” certain parts of code and execute other parts. In *Python*®, you can do this with **block strings** . To begin a block string, insert ' ' ' (three single quotation marks). To end the block string, insert another set of triple quotation marks at the end of the section you want to deactivate. Anything between these lines will not execute in the program.

---

In the code below, which variables (a, b, c, and d) will be created and assigned values?

```
a = 1
''' begin block string
b = 2
c = 3 '''
d = 4
```

<input type="radio"/>	a
<input type="radio"/>	b and c
<input type="radio"/>	a and d
<input type="radio"/>	a, b, c, and d

Stuck? [Show Answer](#)

[Check Answer](#)



**Temporarily Commenting:** You can think of the block string technique as temporarily commenting out code segments. However, block strings should not become a permanent part of your code. When you want to comment your code permanently, use the comment character `#`.

- 8 Look at your program in VS Code. The variable `w` seems to begin a new section of code. To see what the section before it does, deactivate a major part of your program with a block string. Begin a block string before the `w = 6` line and end it right before the `wn = turtle.Screen()` line.
- 9 Save and run the program again. What does the first section of code do? What is a better variable name for `x`?

In *Python*, as in other languages, there are syntax rules and strong *conventions*, or guidelines, for naming variables and other constructs. When everyone follows good programming conventions, its easier for everyone to interpret and modify the code. For variable names, adhere to the following rules and conventions.





## Variable Naming Rules and Conventions

### Rules (you must follow)

- Start variables with a letter or underscore.
- Use only alpha-numeric characters; letters, numbers, and the underscore.
- Do not use *Python* keywords. For a list of these, please reference [Python Keywords and Built-ins](#).

### Conventions (you should follow)

- Start variable names with a lowercase letter.
- Use underscores to separate words, such as `num_turtles`.
- If necessary, use digits as in `turtle1`, `turtle2`; however, descriptive words are often better, as in `small_turtle`, `turtle_left`, or `fast_turtle`.
- Do not use uppercase letters; variable names are case sensitive.
- Do not use lowercase `l` (el) and uppercase `O` (oh) by themselves, they are easy to confuse with numerals `1` and `0`.

10

In your code, change the variable name `x` to a name that represents its purpose.

### Hint for `x`

**Reminder:** Change the variable name `x` in every instance where the variable `x` is used.

- 11 Test your modifications by running the program.

**Note:** While learning what this program does, you may discover some bugs or you may want to change how the program works. For now, resist the temptation to change the functionality; just focus on renaming variables. You will be fixing and improving the code soon.

## Improve Your Code

As you continue to learn this program, you will discover many ways to improve it, making it easier to fix its bugs and modify its algorithms.

### Variable Names

---

Renaming variables to indicate their purpose is a great way to learn a program that has poorly named variables.

- 12 Remove your block strings and find where the variable `w` is used. Determine its purpose.

**Hint for `w`**



**Helpful Tip:** You can use the find feature, Ctrl+F, to highlight all occurrences of “`w`” in the code.

- 13 Rename the variable `w` based on its purpose.
- 14 Test your improvement by running the program to make sure it is still running correctly.

Still another way to learn what code does is to experiment with different values of variables. Even if you cause errors choosing values that are too large or too small, you can still learn something about the code.

- 15 Duplicate the line of code that initializes the `y` variable and comment it out using a block string. This will allow you to experiment with the values of the variable while keeping the original code intact. Then, change the value of `y` to learn how it is used.

#### Hint for y


- 16 Rename the `y` variable based on its purpose and then test the program. When you are satisfied with the results, restore the original value of `y` and remove any duplicate values of `y` you may have used in your testing.

You can learn what code does by adding `print` statements to observe the value of variables. You can see how they change and affect the output of the program.

- 17 Insert the statement `print("z=", z)` after `z` is initialized (given its first value). Then, add the statement `print("z*n=", z*n)` inside the `while` loop to see how `z` is used there. Using the output from the `print` statements, determine what this variable is used for. As an extra clue, you might experiment with the value of `n`.

#### Hint for z

- 18 Rename the variable `z` to a better name and test the program. You may also choose to delete the `print` statements you used to help you debug.

The last variable you need to rename (`n`) is an **incrementing counter** . This means it adds one to itself at each iteration of the `while` loop. It is initialized to 0, and in the first iteration of the `while` loop it increments to 1; in the second iteration it increments to 2, then 3, 4, 5, etc., until it reaches its designated value (in this case, one less than the original `w` variable).

A tracing table may help you understand code that uses an incrementing counter. In a loop, tracing is a bit more involved and is best hand traced separately from the rest of the program. Observe this table that hand traces the highlighted code.

	loop	statement	x	y	output
<code>x = 1</code>	1	<code>print(x*y)</code>	1	5	5
<code>y = 5</code>	1	<code>x = x + 1</code>	2	5	-
<code>while (x &lt; 5):</code>	2	<code>print(x*y)</code>	2	5	10
<code>print(x*y)</code>	2	<code>x = x + 1</code>	3	5	-
<code>x = x + 1</code>	3	<code>print(x*y)</code>	3	5	15
	3	<code>x = x + 1</code>	4	5	-
	4	<code>print(x*y)</code>	4	5	20
	4	<code>x = x + 1</code>	5	5	-

The program terminates after four loops, when the incrementing counter `x` reaches 5 and is no longer less than 5.

19


Back in your drawing program, predict the value of `n` each time the algorithm loops, and choose a better name for it. Use a tracing table to help you.

20

All of the variables should now be properly named and you should have a better understanding of the program.



### PLTW COMPUTER SCIENCE NOTEBOOK

Using your newly named variables, write the five lines of the `while` loop and explain what the **iteration**  in your program does.



### CAREER CONNECTIONS

#### Version Control Engineering

As you modify your code, it is a good idea to save each major version as you work. This helps you backtrack, or revert to a known working version if you end up introducing new problems while trying to debug existing ones. This “versioning” of software is a common practice that many professional software developers use to help reduce bugs and make testing software easier.

**21**

Save this version of your code as directed by your teacher.

## Readability

---

With your new understanding of what the code does in the program, you will add comments and spacing to further improve the readability of the program.

**22**

Insert a blank line and a comment line for each of the following sections, stating the purpose of the code segments. You can use these phrases or rephrase and use your own:

- Create a spider body
- Configure spider legs
- Draw legs

- 23 Test the program to make sure you have not accidentally introduced any new bugs.
- 24 Save a named version of your commented code as directed by your teacher.

## Variable Values

---

Now that the program is more readable and you are more familiar with it, you can tackle some problems with the algorithm. The next three steps state what the problems are. Make the necessary changes to the program to fix each bug.

- 25 The leg placement is too low in relation to the body. Move where the legs are drawn up about 20 pixels (in the y-direction).

### Hints for correct leg placement

- 26 The number of legs is incorrect. Change the program to specify the correct number of legs on the spider and test your program. (This is not the final leg placement, just a step in the debugging process.)

### Hint for correct number of legs

- 27 The angle between the legs is incorrect and inconsistent. Equally distribute the eight legs around the spider according to the number of degrees in a circle. (This is still not the final leg placement.)

### Hint for proper leg angle

- 28 Save a named version of your fixed code as directed by your teacher.

## Algorithms

---

With the three bug fixes you have made, your code is much improved. But you might argue that the image does not resemble a spider very much at all. In fact, it may look like a compass or a clock with not enough hours. You will improve the algorithm to make the image much better.

**29**

Change where the legs are rendered. Reduce the angle between the legs and place half of the legs on one side of the body and the other half on the opposite side.

**Hints for legs on opposite sides of the body**

**30**

Add two small eyes to the spider.

**Hints for spider eyes**

**31**

Save a named version of your improved code as directed by your teacher.



**Discussion Prompt:** Compare your version of the spider program with a classmate's.

- Explain how algorithms can be written in different ways and still accomplish the same tasks.
- Explain how algorithms that appear similar can yield different side effects or results.

## It's Your Turn

Your friends Tasha and AJ are working on a nature exhibit for biology. Their third teammate, Nadia, was just moved to a different class, so AJ has asked you to continue where Nadia left off. AJ shares with you Nadia's `ladybug` program that should produce the image as shown, but the program has bugs. Since you have had some practice finding and fixing bugs, you will revise Nadia's program so it works as expected.

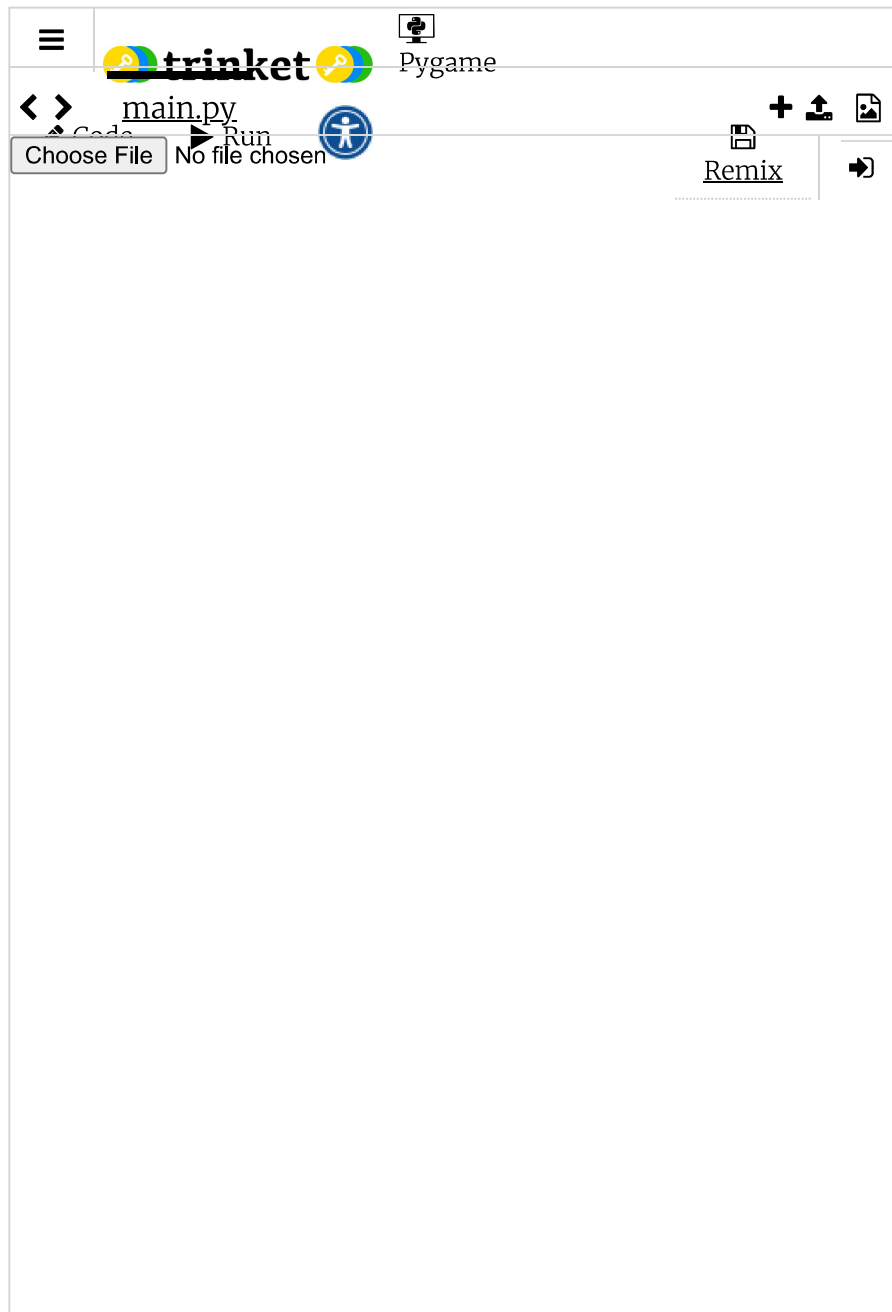


Figure 1.  
Ladybug

- 32 Visit the [Debugging Tips](#) resource to view a summary of the debugging techniques you learned in this activity.
- 33 Use the debugging techniques you have learned to debug this program. Copy and paste it into VS Code and name the file `a116_ladybug_[studentinitials].py`. Make all of your changes in



VS Code.



34

Save the corrected ladybug program as directed by your teacher.

**PLTW COMPUTER SCIENCE NOTEBOOK**

Record and explain the errors you found with this program.

Describe in your own words why the errors occurred and how you fixed them.



**Reflection Question:** How would a for loop have helped avoid the infinite looping error?

## Bug Review

After fixing the bugs with the spider program and improving its algorithms, I feel I can now

SELECT ALL THAT APPLY

- ☐ a Determine what a program is supposed to do, even if it is poorly written and lacking comments.
- ☐ b Learn what variables do, where or when they are used, and predict their possible values.
- ☐ c Apply coding methods I have already learned (such as using print statements) to learn and debug code.

Stuck? [Show Answer](#)

[Check Answer](#)

## Advanced “Bugs”

If directed by your teacher, you can make further improvements to your bug images by reusing and modifying algorithms you already know.

## Ladybug

**35**

In your ladybug program, re-use your spider-leg algorithm to add straight legs to the ladybug image. A possible variation of your ladybug image is shown.



Figure 2.  
Ladybug  
with legs

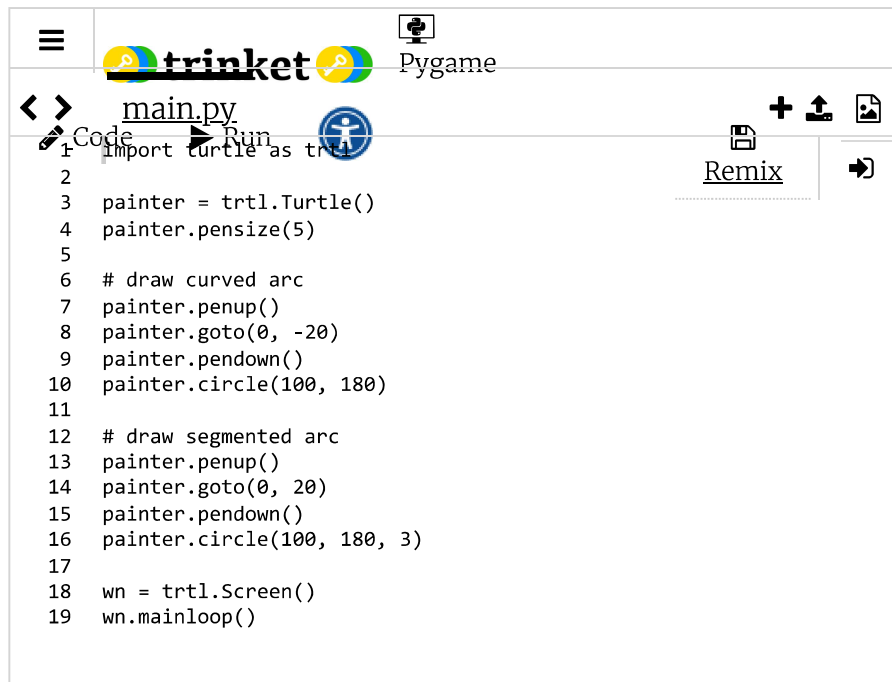
### Hints for ladybug legs

## Spider

You should be familiar with the `circle` method from previous activities. For example, `turtle.circle(20)` draws a circle with a radius of 20. Recall from Activity 1.1.2 that you can specify a second parameter, the *extent* of the circle, to draw an arc. For example, `turtle.circle(20, 180)` will draw one half of a circle with radius 20.

**36**

- Use this code editor to experiment with drawing arcs.
- Change the `radius` and `extent` parameter values of `circle`.
  - Change the `to_angle` in `setheading`.
  - Experiment with both positive and negative parameter values.



The screenshot shows the Trinket Pygame editor interface. The top bar includes the Trinket logo, a Pygame icon, and the text "Pygame". Below the bar, there are icons for "Code" (a pencil) and "Run" (a play button). The main area contains a Python script for drawing a spider. The script uses the turtle module to create a painter object, set its pen size, and draw two arcs for the legs. The bottom right corner has a "Remix" button and a share icon.

```
1 import turtle as trtl
2
3 painter = trtl.Turtle()
4 painter.pensize(5)
5
6 # draw curved arc
7 painter.penup()
8 painter.goto(0, -20)
9 painter.pendown()
10 painter.circle(100, 180)
11
12 # draw segmented arc
13 painter.penup()
14 painter.goto(0, 20)
15 painter.pendown()
16 painter.circle(100, 180, 3)
17
18 wn = trtl.Screen()
19 wn.mainloop()
```

37

In your spider program, use the `circle` and `setheading` methods in an iteration/looping algorithm to draw curved legs on a bug instead of straight ones.

### Hints for curved legs

38

Add a head to the spider by drawing a smaller circle close to the body of the spider and change the eye location. A possible variation of your spider image is shown; yours may look quite different.



Figure 3.  
Spider



### PLTW COMPUTER SCIENCE NOTEBOOK

In this activity, you used a few different methods to find errors in your programs. Some IDEs and programming languages also include tools for visualization of errors.

#### **Add to your notebook:**

The following are effective ways to find and correct errors:

- test cases
- hand tracing
- visualizations
- debuggers
- adding extra output statement(s)



**Discussion Prompt:** Share your various versions of your programs with a classmate.

## CONCLUSION

- 1 What do you think is the biggest benefit of using well-named variables?
- 2 What can a programmer do to reduce bugs in code?

---

Proceed to next activity