

Computer Programming in Java

Bush School *CPJava Fall 2023*

Welcome to CPJava!



Dru & not Gru!

Chandru Narayan
Bicyclist Astronomer Engineer
Teaching Astronomy and CS



Introductions!

1. State your name - how would you like to be addressed?
2. Your personal pronoun?
3. What are your hobbies?
4. Say something interesting or peculiar about yourself
5. Do you have any exposure to programming?
6. What made you choose this course?
7. What are your expectations from this course?
- 8.

A unique **first** course in programming with some advanced topics!

1. Visual and Project-based learning
2. Computing and the Web
3. Programming using Java
4. Simulate Natural Systems
5. Develop Games
6. Learn Math & Physics Concepts
7. Machine Learning (as time permits)
8. AP Exam preparation (optional)

Course Requirements

1. The CPJava course does not require you to have prior programming experience
2. Prerequisites include Algebra 1
3. For students wanting to take the APCS A exam there will some extra assignments that will need to complete. They will not need to do a Final Project.
4. Students not taking the APCS A will complete a Final Project

Expectations of Learning

- Advanced Java Programming
- Object Oriented including inheritance
- How to make and maintain your own website
- Some basic HTML & CSS
- How to use GitHub
- Searching and sorting
- Recursion
- Robust code design and style
- APCS A topics
- Code Vectors, Newton's Laws, Machine Learning etc
- + Fun stuff like Asteroids, Super Mario Games and Computer Art that they don't teach you in college!

CPJava course is online!

- The complete 1-year coursework is online!
- We'll start at the top and work our way to the bottom through this course

You are here! Click here to access.

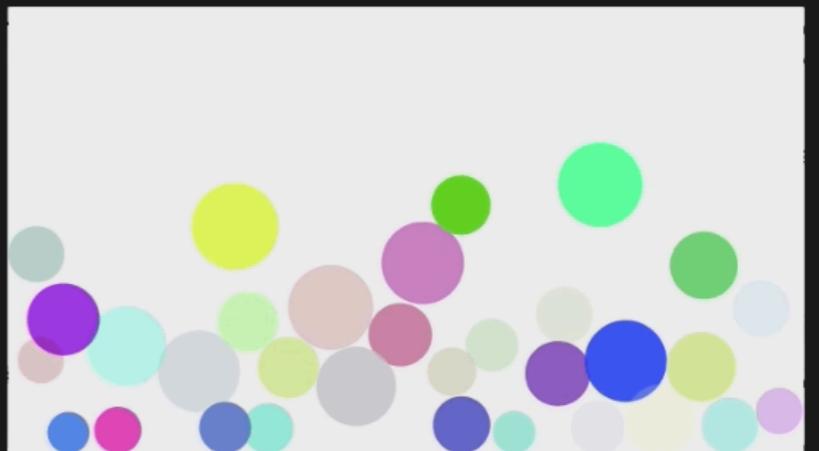
Bookmark this
You will need it every day!



/ CPJava Course
Bush School Computer Programming in Java Course

[View on GitHub](#)

Bush School CPJava Fall Semester 2020



[Click here for live code and click bubbles inside resulting tab or window](#)

CPJava - Computer Programming in Java Course

This course is designed to introduce computer programming in the Java language. Learning to use a computer language is a necessary skill for all students regardless of discipline. In this course we will teach the fundamentals of computer programming from the stand point of simulation, automation, and problem solving of real-world systems and natural processes. At the same time, the design and implementation of computer programs is taught from the context of fundamental aspects of computer science, including the development and analysis of algorithms, the development and use of fundamental data structures, the study of standard algorithms and typical applications, and the use of logic and formal methods.

In addition, the year-long course will cover many of the topics necessary for preparation to the AP Computer Science A (APCSA) examination in Spring of the following year. This is an introductory course in computer programming using Java. As such, no specific programming prerequisites are needed to take this course. However, additional preparation may be needed to fully prepare a student for the AP CSA exam with no prior knowledge of computer programming.

Course Schedule

LESSON UNITS	APPROXIMATE DURATION	TOPICS
FALL 2023 SEMESTER	12-Weeks	Sep 2023 to Jan 2024
Unit 1	2-Weeks	Introduction to Java, Tools walkthrough, Classroom processes, Environment setup, Java Primitive types and Operators
Unit 2	2-Weeks	Objects, Methods, String, Pointers, Integer, Double, Math
Unit 3	2-Weeks	Control flow, Booleans, If statements, Object traversals, Integer, Double, Math
Unit 4	3-Weeks	Iteration, While loops, For loops, Nested Loops, Loop Analysis
Unit 5	3-Weeks	Anatomy of a class, Constructor, Accessor, Mutator Methods, this keyword
SPRING 2024 SEMESTER	15-Weeks	Jan to June 2024
Unit 6	2-Weeks	Arrays, Array Traversal, Data Structures, Project work
Unit 7	3-Weeks	ArrayList, Big data, Ethics of Data Collection, Privacy
Unit 8	2-Weeks	String, substring, Wrapper Classes, 2D Arrays, Table representations
Unit 9	3-Weeks	Inheritance, Encapsulation, Hierarchy, Polymorphism, Multi-part Project
Unit 10	3-Weeks	Recursion, Recursive Search, Recursive Sort
Unit 11	2-Weeks	Final Project Peer Sharing Final Project Presentation or APCS A Exam

A complete Syllabus is available at the [CPJava Course website](#)

Grading Policy

Points are assigned for:

Completing Classwork Exercises

Submitting Projects

Professionalism & Integrity

APCSA exam is on May 8th 2024

Talk to me if you are going to take it!

Grades

Grades are assessed each Semester

50% Projects

(Includes Final Project or APCSA Exam)

35% Classwork / Assignments

15% Student Portfolio

Details are available
at the CPJava Course website

How to succeed

Simply write lots of Java code

Publish your work to the web (Github)

You cannot learn programming by reading or
watching!

Make lots of mistakes - truly the best way to learn
to code!

Working cooperatively with your peers - Pair
Programming!

Don't be afraid to try new things!

Professionalism

- How you conduct yourself during your work
- Includes (but not limited to)
 - Classroom etiquette
 - Reliability and accountability
 - Ethics
 - Working cooperatively with your peers

Office Hours and Class Assistance

- Conference Hours in Wis 207
 - Refer to Portal
- Class Assistance
 - Wyatt Thelan is the TA for this class!

Tools/Resources for CPJava

CPJava website - Lesson Plans and Projects

Bush Portal - Course Syllabus, Schedule and Grades

CPJava Google Classroom - You should have received an invitation

CPJava Slack - You should have received an invitation

Applications for Laptop

- Github, Processing, Visual Code

- See Instructions in Google Classroom

CSAwesome Online

- Online Textbook, Lessons, Exercises, Detailed Reference

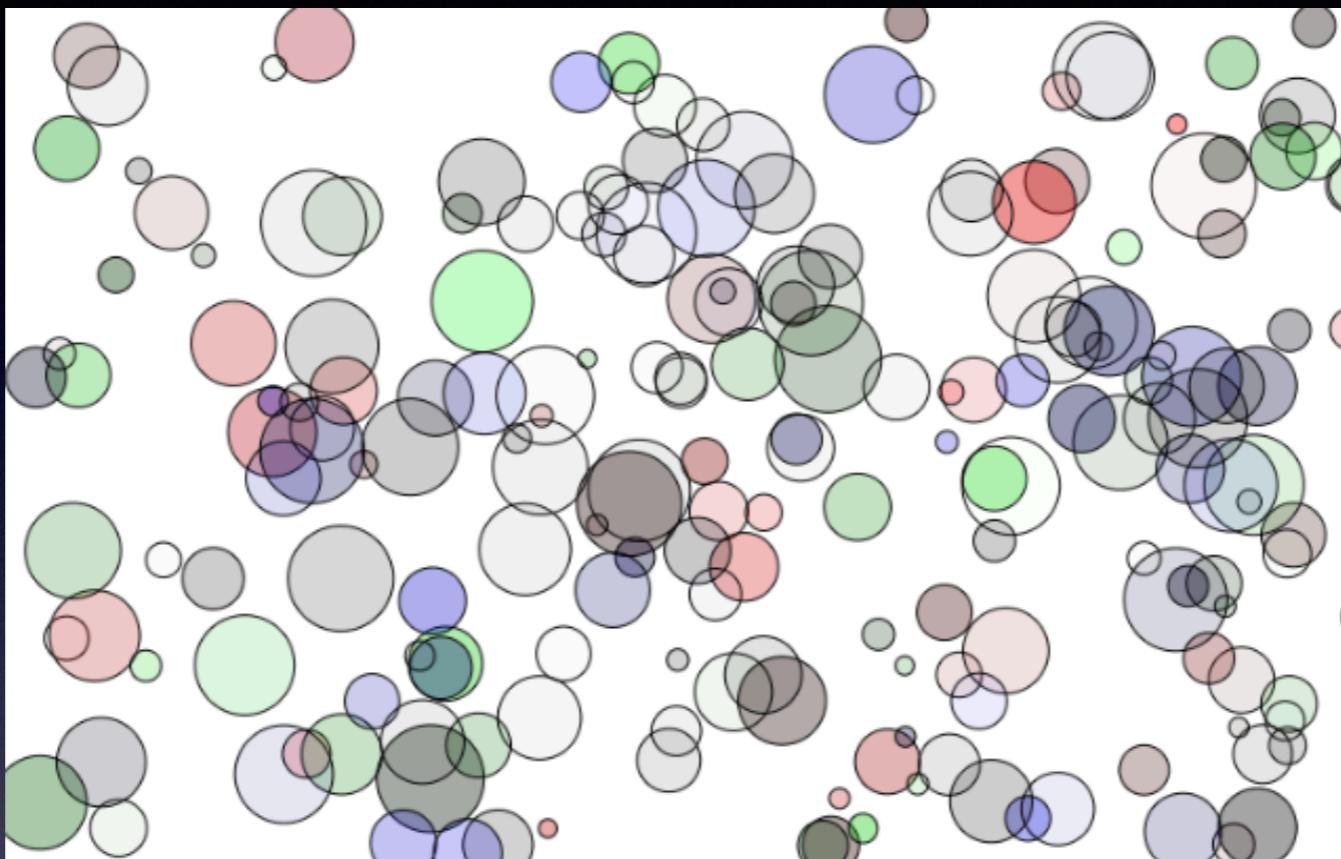
System Requirements

A laptop Computer - Mac or Windows
(Chromebook may not be adequate)

Sufficient disk space, a functioning laptop battery
fully charged!

Functioning Camera and Microphone - especially
required for Paired Programming

Let's run our first Java Program!



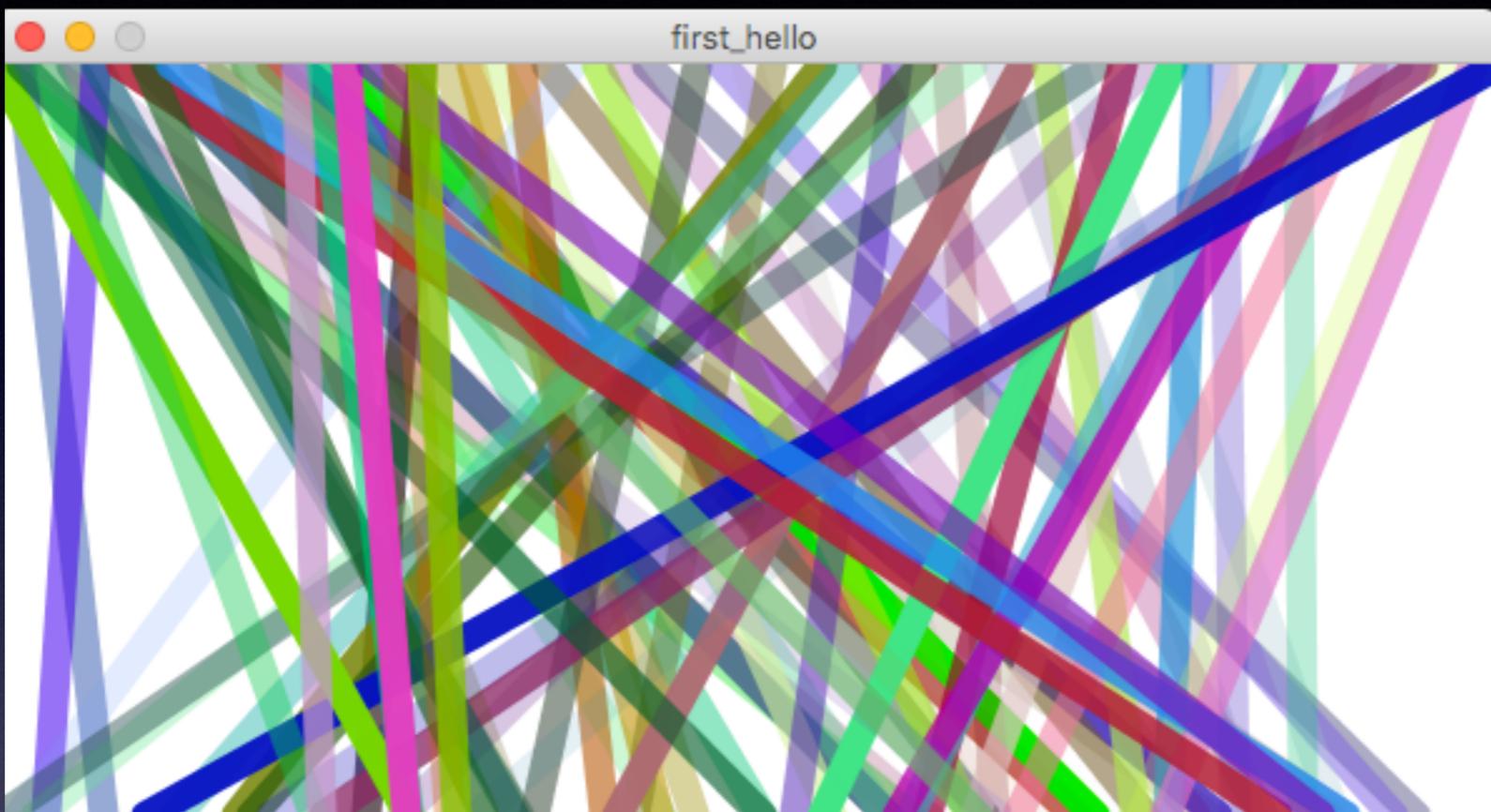
Bubbles - click to run the program

You can sort by color (key 's') and freeze each bubble by clicking on it

Can you freeze all the bubbles?

You will develop visual code like this in Java and much more!

Let's modify our first Java Program!



Access and complete the First Hello and Sticks Google Classroom Assignment during class!

Java Basics

Unit 0

Unit 0-A

The common building blocks in programming languages
are:
Variables
Loops
if statements
Functions (aka “methods”)

Over the next two weeks we’ll go over how these work in
Java

A basic Java Hello program

- Click on the link to the “Four 4s Challenge” of just go to [CP Java Website](#)
- It might look complicated at first, but . . .



The screenshot shows a Java code editor interface. At the top, there's a navigation bar with icons for file operations, the 'trinket' logo, a 'Java beta' button, a 'Run' button, a 'Share' button, and a dropdown menu. Below the bar, the code editor displays a file named 'ABasicJavaProgram.java'. The code itself is:

```
1 public class ABasicJavaProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello You!");
6     }
7 }
```

A basic Java program

- It turns out that this line is much more important than the others
- So in subsequent slides we can ignore everything except the code statements circled in green - Practice in Processing editor!



```
trinket Java beta Run Share ABasicJavaProgram.java
1 public class ABasicJavaProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello You!");
6     }
7 }
```

A basic Java program in Processing

- Ignore any references to Trinket in the subsequent slides.
Instead we will use Processing. It's Much easier!

The screenshot shows the Processing 4.0b4 IDE interface. The title bar reads "sketch_230907a | Processing 4.0b4". The main code editor window displays the following Java code:

```
sketch_230907a
void setup() {
    System.out.println("Hello You!");
}
void draw() {
```

A green oval highlights the first two lines of code: "void setup()" and "System.out.println("Hello You!");". Below the code editor, a preview window shows the output "Hello You!". At the bottom of the screen, there are tabs for "Console" and "Errors", and an "Updates" button with a notification count of 1.

Statements

- The circled code is an example of a *statement*
- It's like an English sentence
- A semi-colon marks the end of a statement

DO THIS —→

System.out.println("Hello You!");

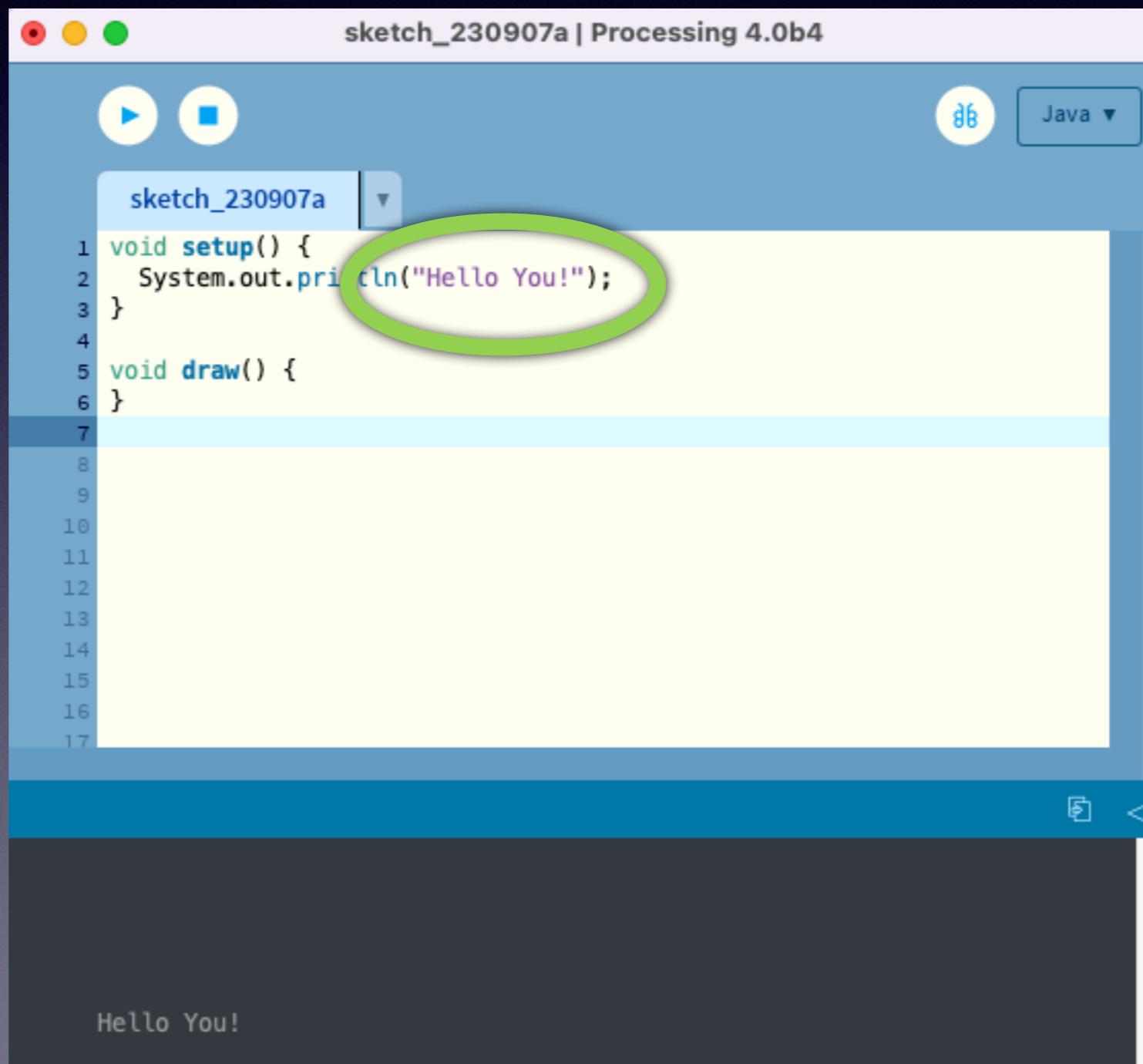
NOT THIS

```
sketch_230907a | Processing 4.0b4
sketch_230907a
void setup() {
    System.out.println("Hello You!");
}
void draw() {
```

Hello You!

String

- "Hello You!" is a Java **String**
- A **String** is a collection of letters, digits, punctuation and / or spaces
- The beginning and end of the **String** are marked with double quotes ("")



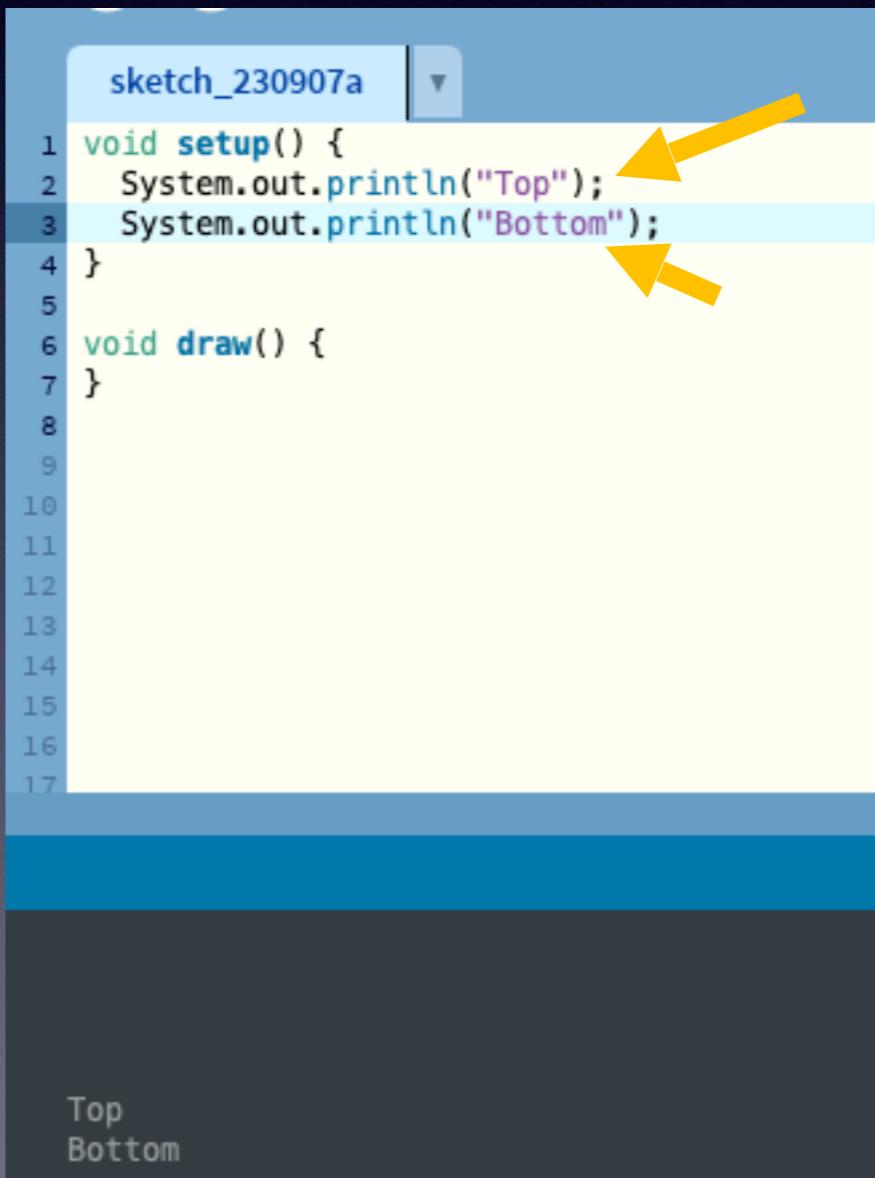
sketch_230907a | Processing 4.0b4

```
sketch_230907a
1 void setup() {
2     System.out.println("Hello You!");
3 }
4
5 void draw() {
6 }
```

Hello You!

print() vs println()

- `System.out.println()` prints first and then goes to the next line



The screenshot shows the Arduino IDE interface with a sketch titled "sketch_230907a". The code in the editor is:

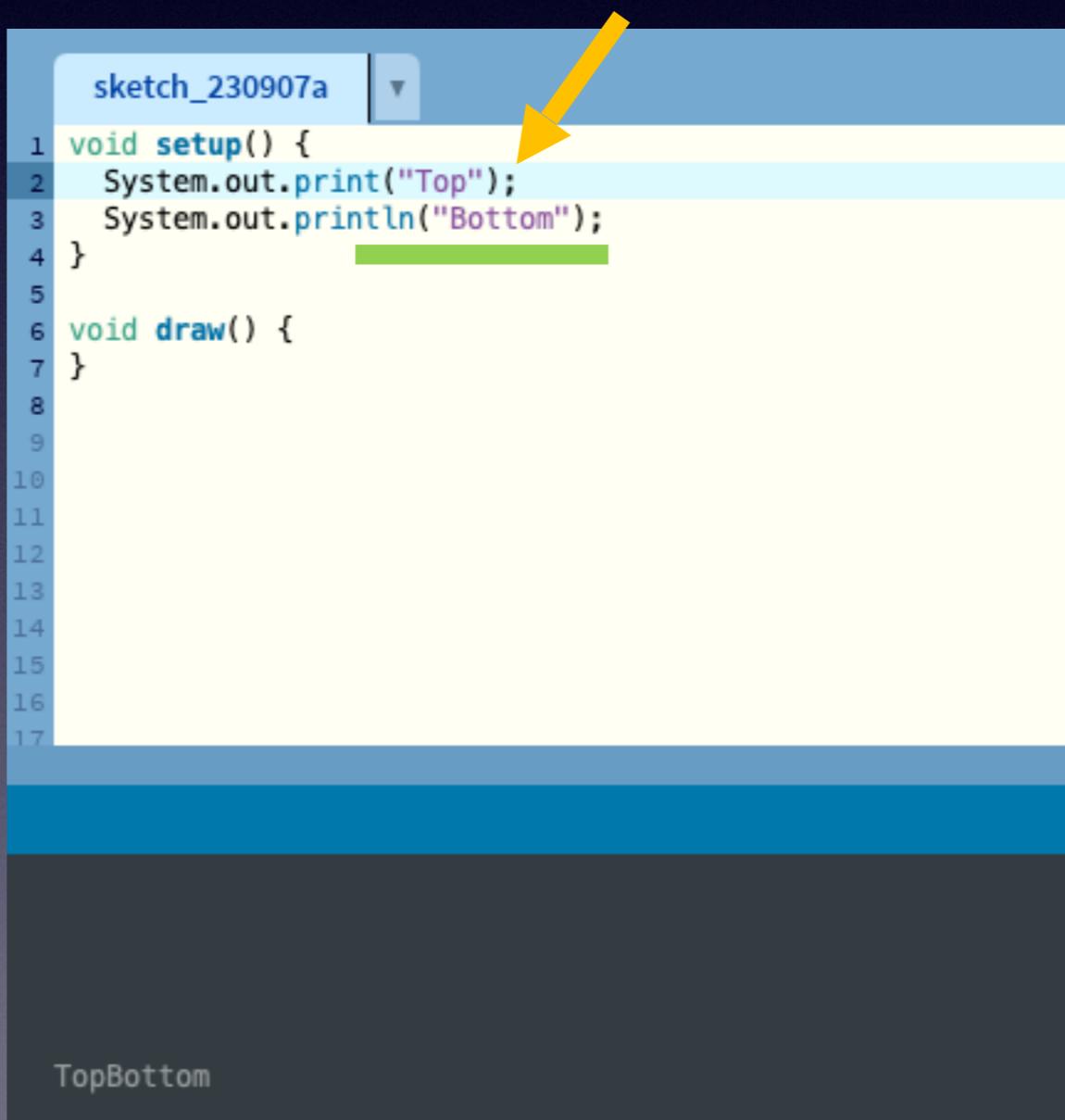
```
1 void setup() {
2     System.out.println("Top");
3     System.out.println("Bottom");
4 }
5
6 void draw() {
7 }
```

Two yellow arrows point to the second and third lines of the code, which are both `System.out.println` statements. The output window at the bottom of the IDE shows the results of these prints:

```
Top
Bottom
```

print() vs println()

- **System.out.print()** prints, but it does NOT go to the next line
- If we change the first statement to **System.out.println()**
"Bottom" is printed on the same line as "Top"

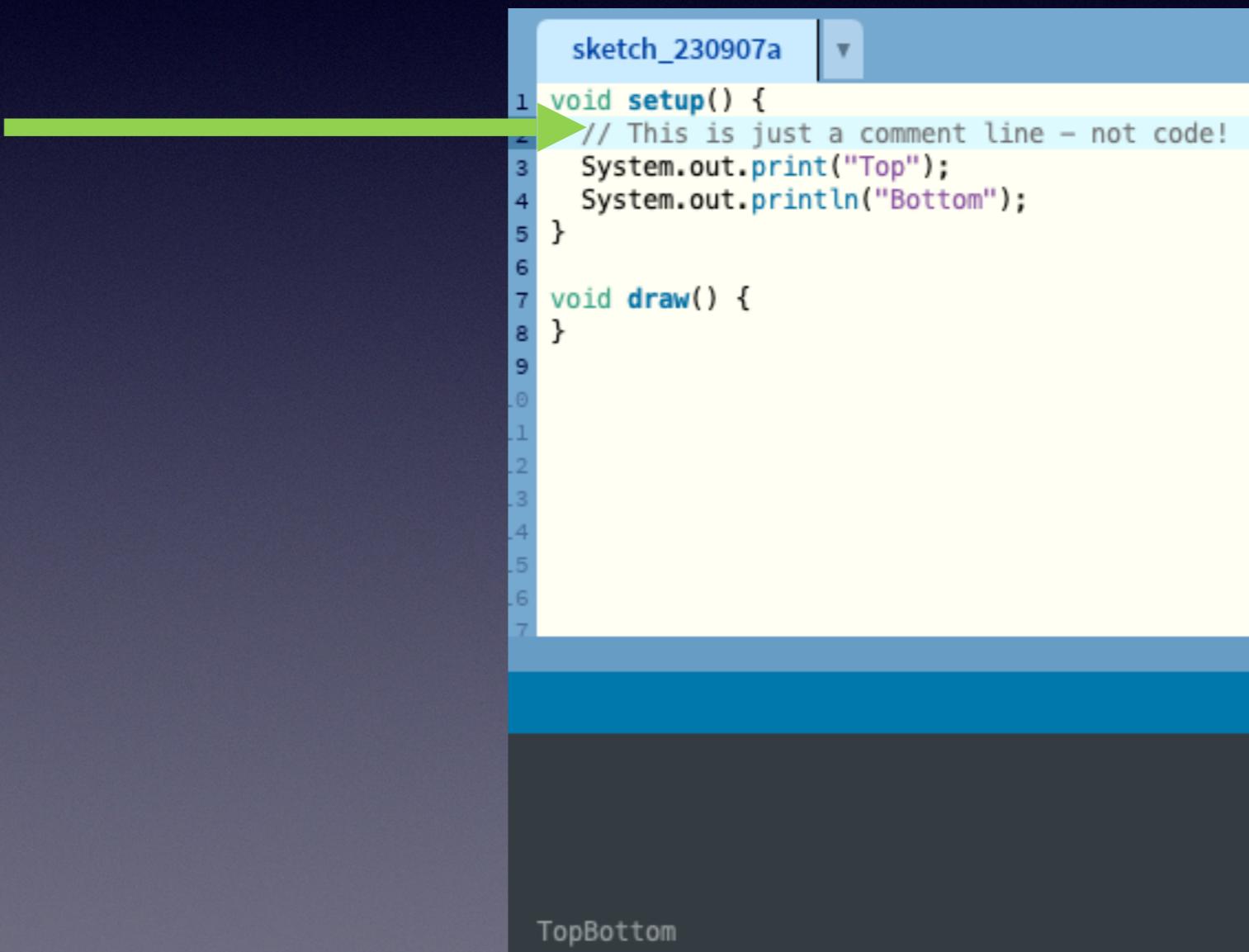


```
sketch_230907a
1 void setup() {
2     System.out.print("Top");
3     System.out.println("Bottom");
4 }
5
6 void draw() {
7 }
```

TopBottom

Comments

- Comments have no effect on the execution of the program, but they make it easier for other programmers (and your future self) to understand what you meant to do



```
sketch_230907a
1 void setup() {
2 // This is just a comment line - not code!
3 System.out.print("Top");
4 System.out.println("Bottom");
5 }
6
7 void draw() {
8 }
```

The screenshot shows the Arduino IDE interface with a sketch titled "sketch_230907a". The code editor displays the following code:

```
1 void setup() {
2 // This is just a comment line - not code!
3 System.out.print("Top");
4 System.out.println("Bottom");
5 }
6
7 void draw() {
8 }
```

A green arrow points to the second line of the code, which is a single-line comment starting with two slashes (//). The output window at the bottom of the IDE shows the text "TopBottom", indicating that the code has been uploaded and run successfully.

Escape Sequences

- Special characters
- In Java, Escape Sequences begin with a backslash \
- *A good way to remember the difference between a backslash and a forward slash is that a backslash leans backwards (\), while a forward slash leans forward (/)*

Common Java Escape Sequences

- **\n** Insert a newline in the text at this point



```
sketch_230907a
1 void setup() {
2     // This is just a comment line - not code!
3     // System.out.print("Top");
4     // System.out.println("Bottom");
5     System.out.print("Top\nBottom");
6 }
7
8 void draw() {
9 }
10
11
12
13
14
15
16
17
```

Top
Bottom

- Guess what you will see if you insert another println stmt?

Common Java Escape Sequences

- **\t** Insert a tab in the text at this point

The screenshot shows a Java code editor window titled "sketch_230907a". The code is as follows:

```
1 void setup() {  
2     // This is just a comment line - not code!  
3     // System.out.print("Top");  
4     // System.out.println("Bottom");  
5     // System.out.print("Top\nBottom");  
6     System.out.println("x\tx\tx\tx");  
7 }  
8  
9 void draw() {  
10 }  
11  
12  
13  
14  
15  
16  
17
```

Three green arrows point upwards from the bottom of the slide towards the three tabs in the line "System.out.println("x\tx\tx\tx");". Below the code editor, there is a horizontal bar with four green double-headed arrows pointing left and right, each labeled with an 'x' at its ends.

Common Java Escape Sequences

- `\'` Insert a single quote character
- `\\"` Insert a double quote character
- `\\"\\` Insert a backslash character

The screenshot shows a Java code editor window titled "sketch_230907a". The code is as follows:

```
1 void setup() {  
2     // This is just a comment line - not code!  
3     // System.out.print("Top");  
4     // System.out.println("Bottom");  
5     // System.out.print("Top\nBottom");  
6     // System.out.println("x\tx\tx\tx");  
7     System.out.println("He said \"a backslash looks like this\" \\\\");  
8 }  
9  
10 void draw() {  
11 }  
12  
13  
14  
15  
16  
17
```

Annotations with arrows point to specific parts of the code:

- A green arrow points to the double quotes in the string "He said \"a backslash looks like this\"".
- A green arrow points to the first backslash in the string "He said \"a backslash looks like this\"".
- A yellow arrow points to the second backslash in the string "He said \"a backslash looks like this\"".
- A green arrow points to the backslash character "\\" at the end of the line.
- A yellow arrow points to the backslash character "\\" at the end of the line.
- A green arrow points to the backslash character "\\" at the end of the line.

At the bottom of the editor, the output of the code is shown:

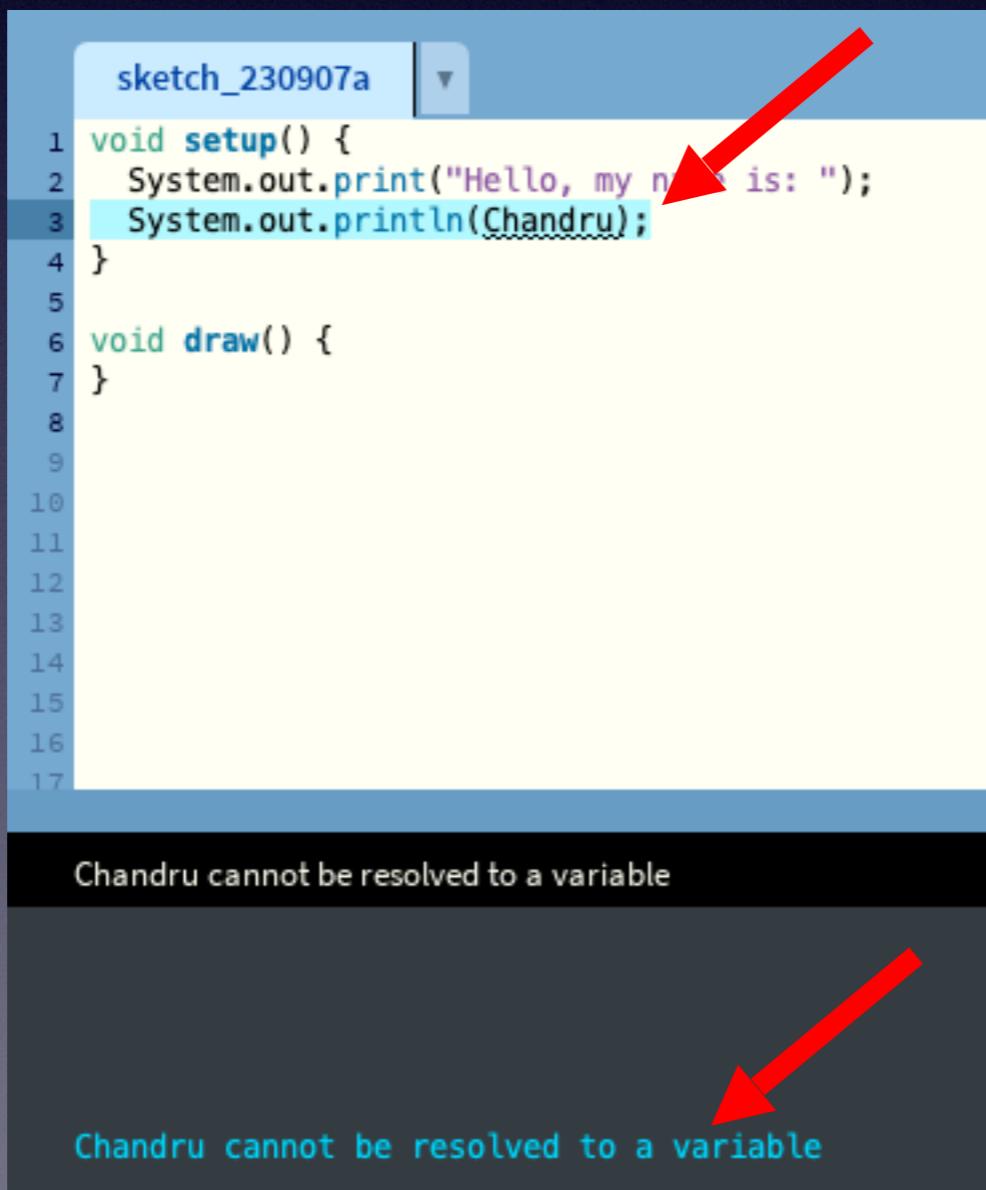
```
He said "a backslash looks like this" \\
```

Debugging

- Errors in programs are called “bugs”
- The process of fixing program errors is called “debugging”
- It’s good to work around other programmers when you are learning a new programming language
- Asking for help with debugging is a part of learning

Errors

- When you write Java programs you will often get an **error message**
- When you are learning a new programming language, errors are a fact of life
- Errors are ok, just fix them and move on



The image shows a screenshot of a Java development environment. At the top, there's a title bar labeled "sketch_230907a". Below it is a code editor window containing the following Java code:

```
1 void setup() {  
2     System.out.print("Hello, my name is: ");  
3     System.out.println(Chandru);  
4 }  
5  
6 void draw() {  
7 }  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17
```

A red arrow points from the word "Chandru" in the third line of code down to a tooltip at the bottom of the screen. The tooltip reads "Chandru cannot be resolved to a variable". Another red arrow points from the same word "Chandru" up to another identical tooltip located higher on the screen.

Syntax error

- In this case I made a *syntax* error
- *Syntax* is the grammar and spelling of a computer language
- Here I forgot the double quotes around my name
- Fix this error and run - does it work?

The screenshot shows a code editor window titled "sketch_230907a". The code is a simple Java sketch with two methods: setup and draw. The setup method contains two println statements. The first statement has a closing parenthesis after "Hello, my name is:", and the second statement has a closing parenthesis after "Chandru". A red arrow points to the word "Chandru" in the second println statement. Below the code editor, a black bar displays the error message "Chandru cannot be resolved to a variable".

```
1 void setup() {  
2     System.out.print("Hello, my name is: ");  
3     System.out.println(Chandru);  
4 }  
5  
6 void draw() {  
7 }  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17
```

Chandru cannot be resolved to a variable

Chandru cannot be resolved to a variable

Logic error

- This time I misspelled my name
- The computer doesn't know my name, so the program **runs correctly** without an error message

The screenshot shows a Java code editor window titled "sketch_230907a". The code in the editor is:

```
1 void setup() {  
2     System.out.print("Hello, my name is: ");  
3     System.out.println("Chandru")  
4 }  
5  
6 void draw() {  
7 }  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17
```

A yellow oval highlights the misspelling of "Chandru" in the third line of the code. The output window below shows the result of running the code: "Hello, my name is: Chandru", with a green oval highlighting the correct output.

(Run time) Exceptions

- Sometimes a logic error crashes the computer and stops the running program
- Here I made the arithmetic error of dividing by zero

The screenshot shows a Java code editor window titled "sketch_230907a". The code is as follows:

```
1 void setup() {  
2     // generate some simple output  
3     System.out.println(1/0);  
4 }  
5  
6 void draw() {  
7 }  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17
```

A yellow oval highlights the line `System.out.println(1/0);`. Below the editor, a black bar displays the error message `ArithmaticException: / by zero`. A large red oval encircles this error message.

Arithmetic in Java

+ **-** ***** **/**

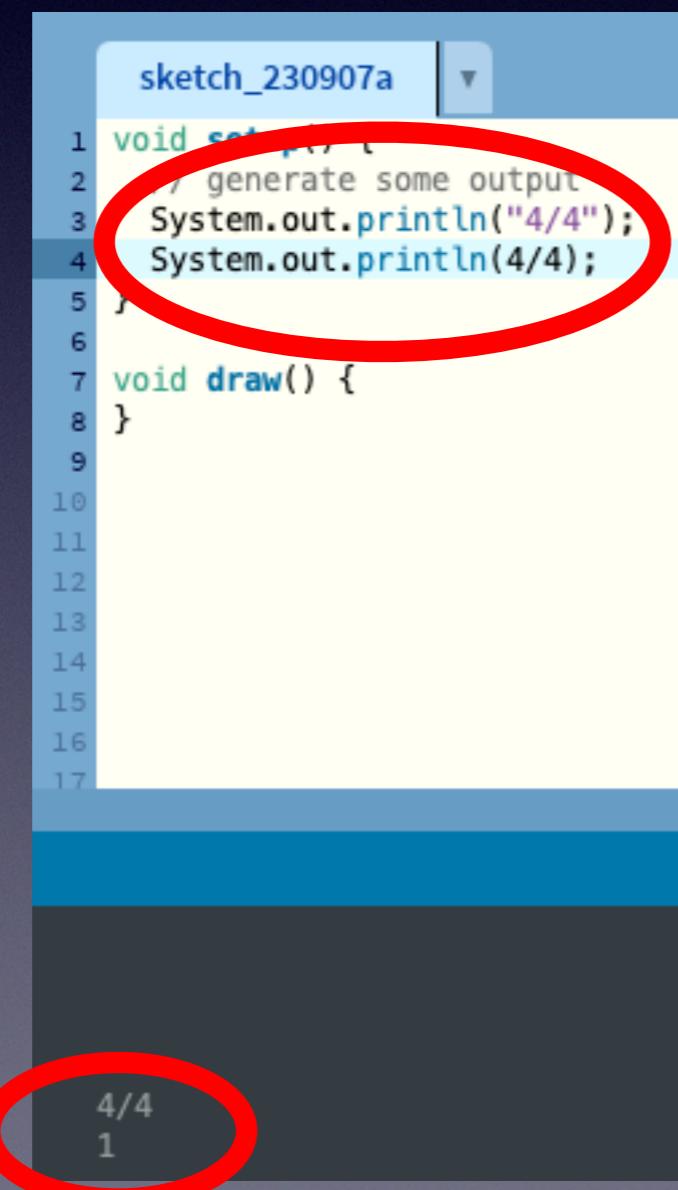
- Addition
- Subtraction
- Multiplication
- Division

Literals vs. Expressions

- Double quotes around text tells Java it is an expression
- Java will **print** an expression exactly as written

Literals vs. Expressions

- Here's an expression "4/4"
- Java prints it in exactly the same form

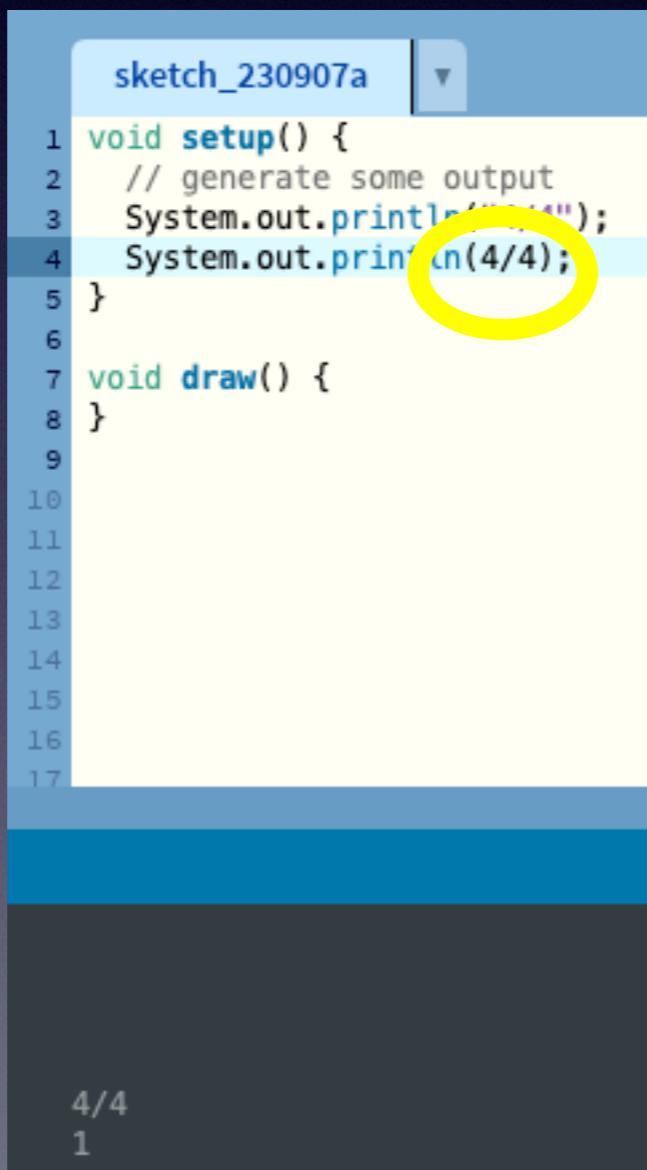


```
sketch_230907a
1 void setup() {
2     // generate some output
3     System.out.println("4/4");
4     System.out.println(4/4);
5 }
6
7 void draw() {
8 }
```

The screenshot shows a Java code editor window titled "sketch_230907a". The code contains two `System.out.println` statements. The first statement prints the string "4/4", and the second statement prints the result of the integer division expression `4/4`. Both of these statements are highlighted with a large red oval. The output window at the bottom of the editor also displays the result, with the string "4/4" and the integer "1" both circled in red.

Literals vs. Expressions

- Here's an expression **4/4**
- Java evaluates it to get an answer 1
- And then **prints** it



```
sketch_230907a
1 void setup() {
2     // generate some output
3     System.out.println("4");
4     System.out.println(4/4);
5 }
6
7 void draw() {
8 }
```

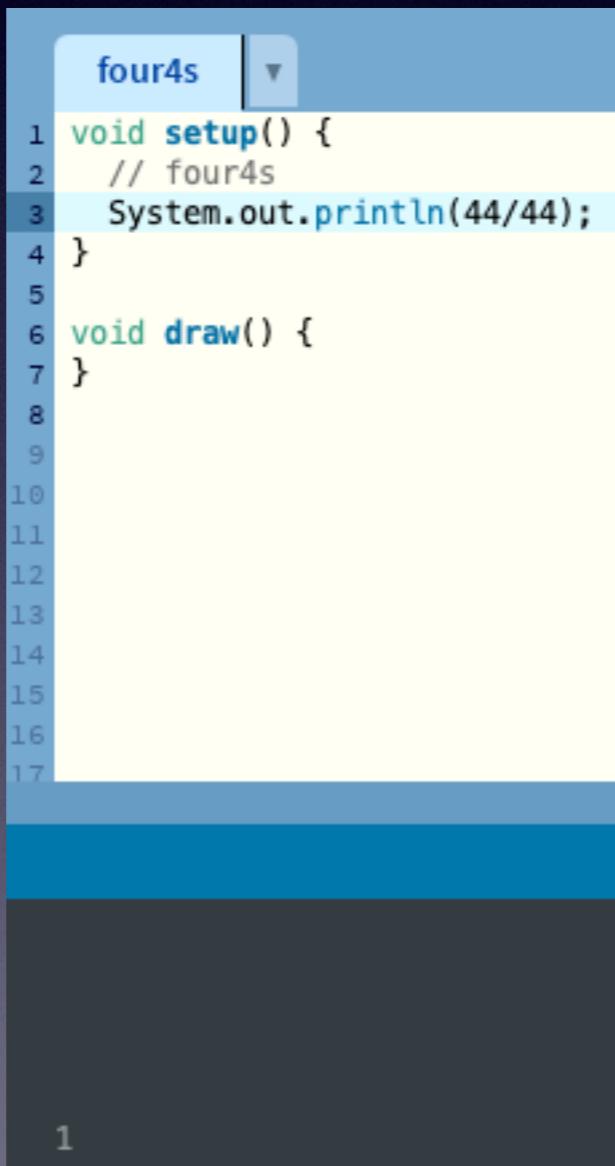
4/4
1

Four 4s challenge

- Use exactly four 4's to write an expression that evaluates to every integer from 1 to 10, using only $+$ $-$ $*$ $/$ and $()$
- No decimals, factorials, square roots, exponents, etc.

Four 4s challenge

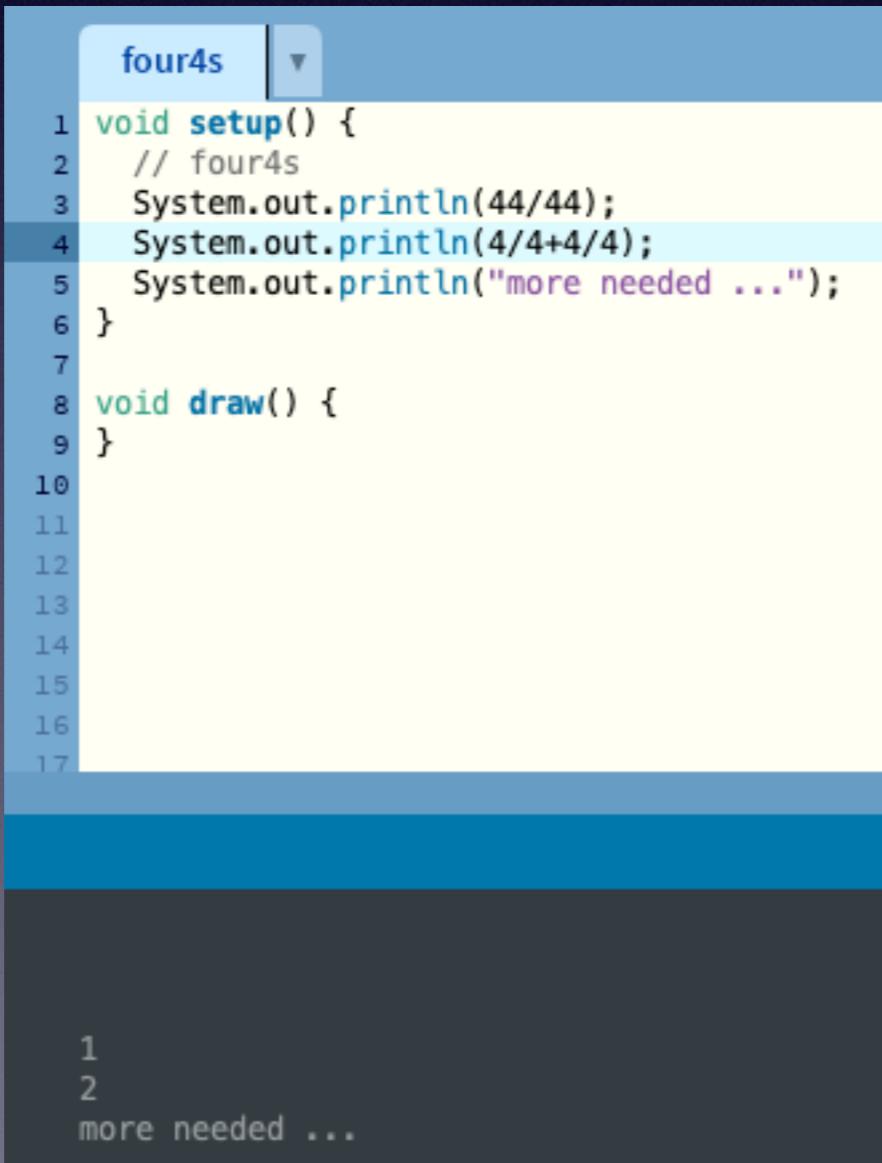
- Print 10 expressions that use arithmetic and four 4s that evaluate to 1 through 10
- Here's one way to do the first



```
four4s
1 void setup() {
2     // four4s
3     System.out.println(44/44);
4 }
5
6 void draw() {
7 }
```

Four 4s challenge (hint)

- Here's another way to do the first (this is a hint!)
- Note the use of PEMDAS !! (Ask me if you do not know)
- Now you will need to write additional println stmts to print 1 to 10
- If you have extra time, try to get 11, 12, 13, etc.
- Submit by slack or to the Google assignment “Four 4s”



The screenshot shows a code editor window with a dark theme. The title bar says "four4s". The code in the editor is:

```
1 void setup() {  
2     // four4s  
3     System.out.println(44/44);  
4     System.out.println(4/4+4/4);  
5     System.out.println("more needed ...");  
6 }  
7  
8 void draw() {  
9 }  
10  
11  
12  
13  
14  
15  
16  
17
```

At the bottom of the editor, there is a status bar with the numbers "1", "2", and "more needed ...".

Unit 0-B

Java Functions and Parameters

Functions (“methods”)

- Organize code just like paragraphs organize writing
- Functions should do just one job or task
- Functions in Java are identified with parenthesis
- The parenthesis may or may not have something inside called an argument
- Here are 2 examples of “calling” a system defined function

```
System.out.println("Hello World");  
in.nextInt();      argument↑  
no argument↑
```

Functions can be used for an *effect* and/or *value*

- `System.out.println()` has an *effect*, it causes something to be displayed
- `in.nextInt()` is used to get the *value* of what the user typed
- Other functions that we will learn about in a couple days can be used to calculate mathematical *values*

void functions (aka methods)

- You can *define* (create) your own functions with the Java keyword **void**
- It should look like this !

The screenshot shows the Processing IDE window titled "sketch_220913a | Processing 4.0b4". The code editor contains the following Java code:

```
void blah() {  
    System.out.println("blah, blah, blah");  
}  
  
void setup() {  
    blah();  
}  
  
void draw() {}
```

Annotations with red arrows and text explain the code:

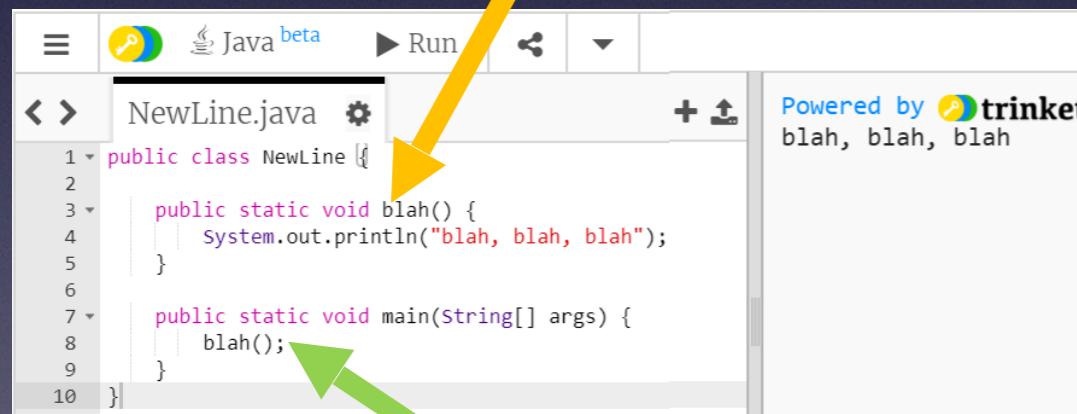
- An arrow points from the first function definition to the right, with the text: "This is a function or method".
- An arrow points from the call to the `blah()` function in the `setup()` function to the right, with the text: "This `setup()` function calls the `blah()` function".
- An arrow points from the empty `draw()` function to the right, with the text: "The `draw()` function is doing nothing now".
- A large annotation box at the bottom right contains the text: "What is you moved the call for the `blah()` function from the `setup()` function() to the `draw()` function ??".

The processing window also shows the output "blah, blah, blah" in the bottom panel.

void functions (aka methods)

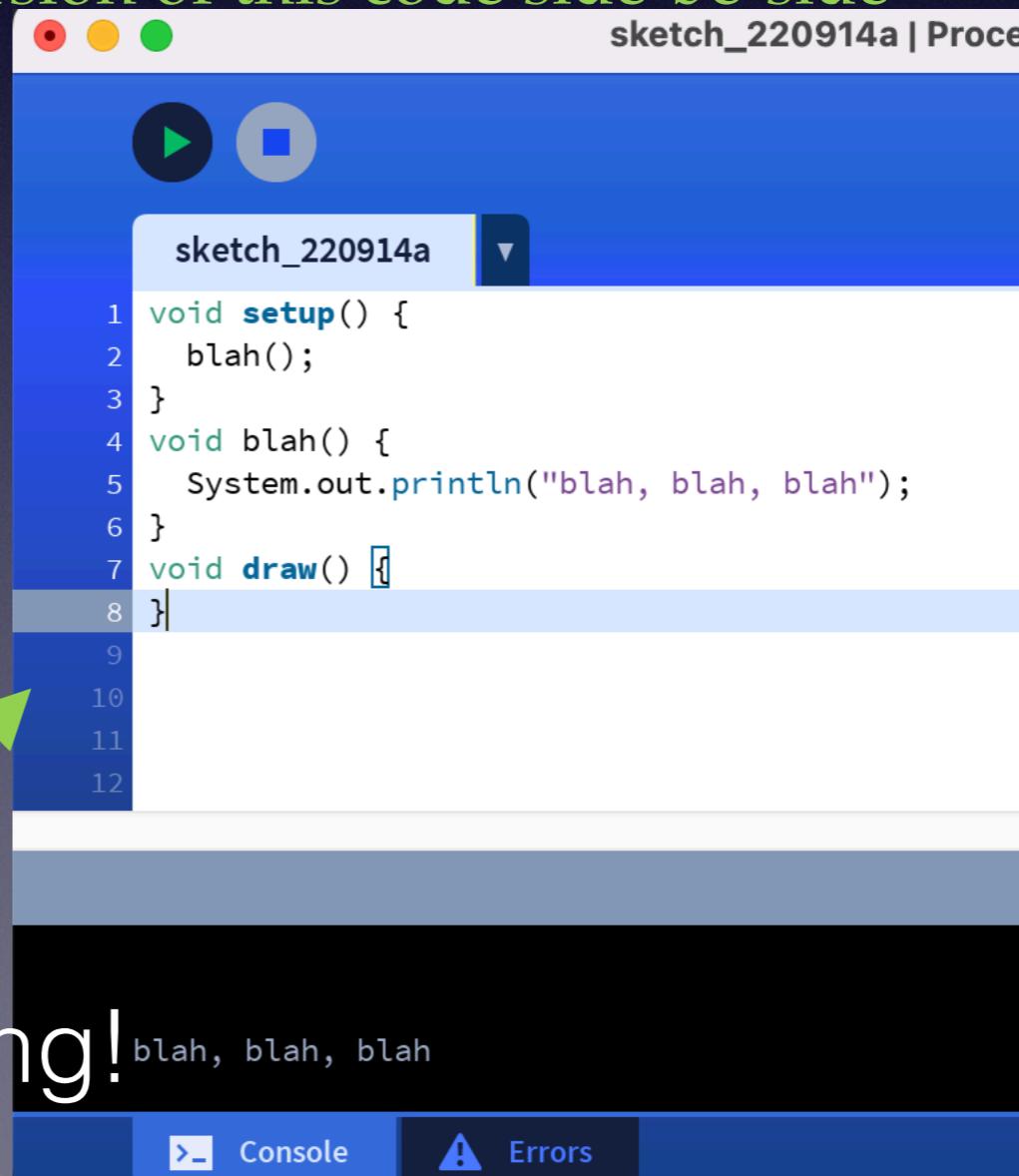
- Functions are like paragraphs of computer code
- Every Java program has a **main** function. There can be many other functions
- More on this later. For now we will use processing which does not need the main() function. Processing version of this code side-be-side

Function blah()



```
1 public class NewLine {  
2     public static void blah() {  
3         System.out.println("blah, blah, blah");  
4     }  
5     public static void main(String[] args) {  
6         blah();  
7     }  
8 }
```

main()



```
1 void setup() {  
2     blah();  
3 }  
4 void blah() {  
5     System.out.println("blah, blah, blah");  
6 }  
7 void draw() {  
8 }  
9  
10  
11  
12
```

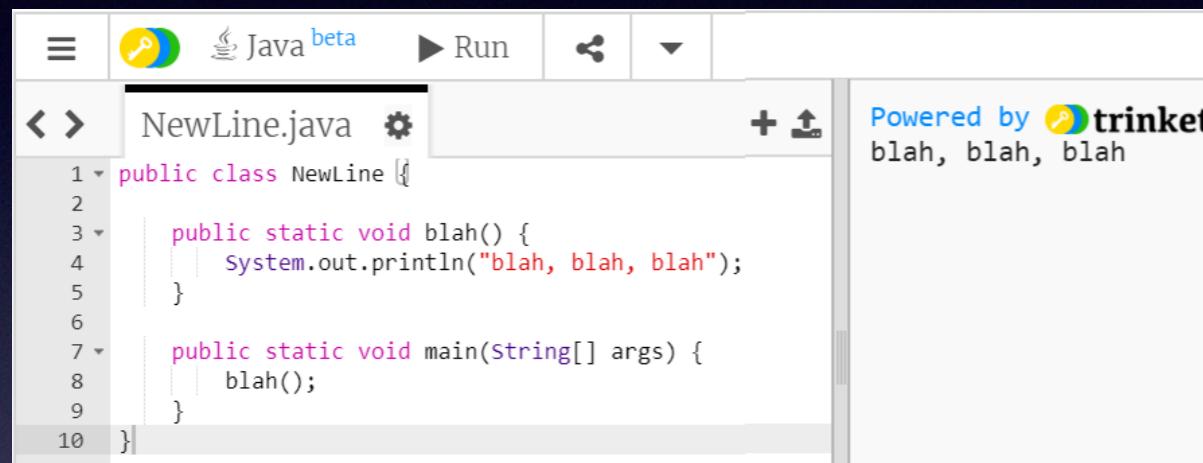
blah, blah, blah

Console Errors

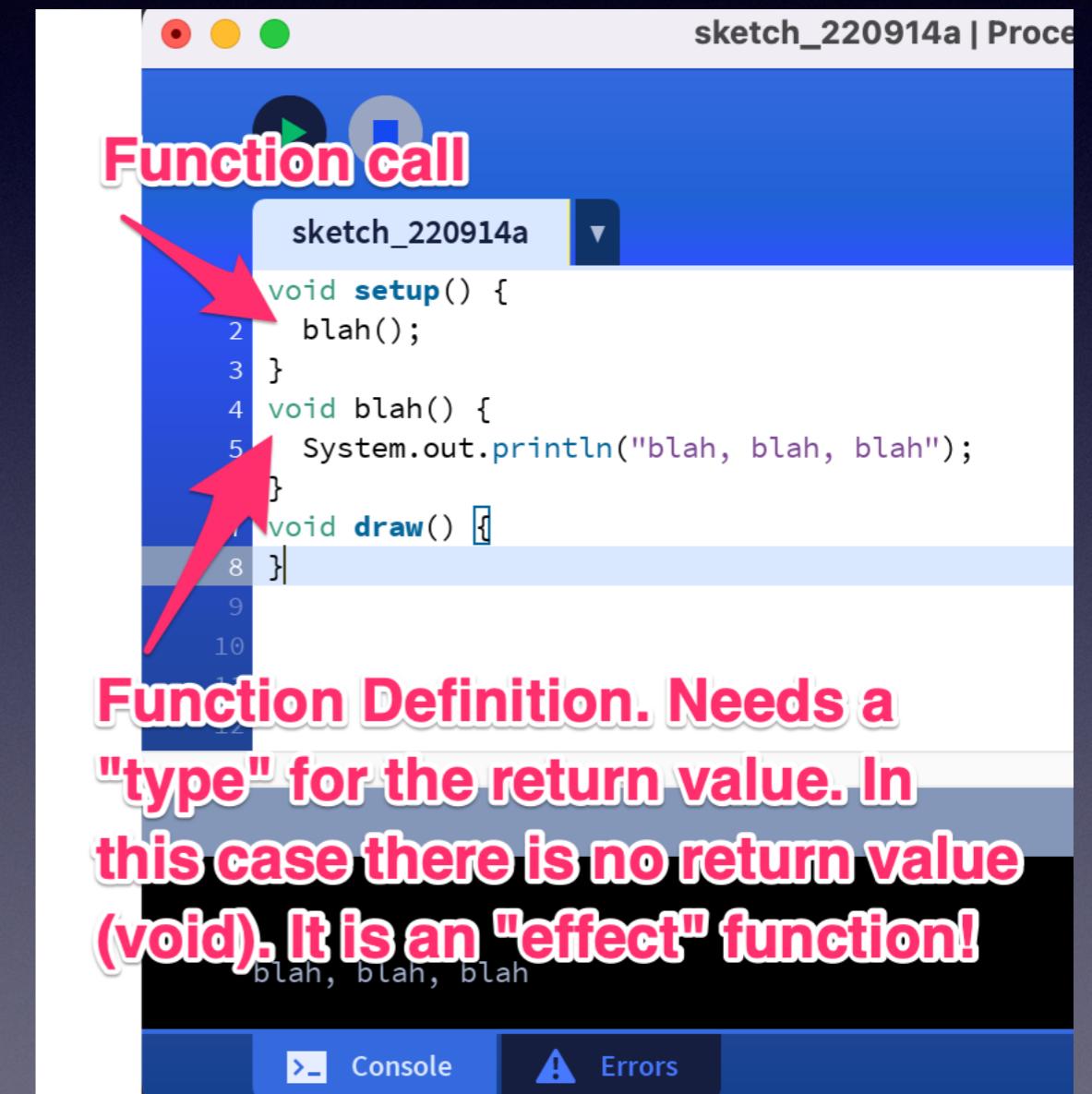
No main() needed in processing!

void functions (aka methods)

- You can *call* (use) your function by typing its name without **void**



```
1 public class NewLine {  
2     public static void blah() {  
3         System.out.println("blah, blah, blah");  
4     }  
5     public static void main(String[] args) {  
6         blah();  
7     }  
8 }  
9  
10 }
```



Function call

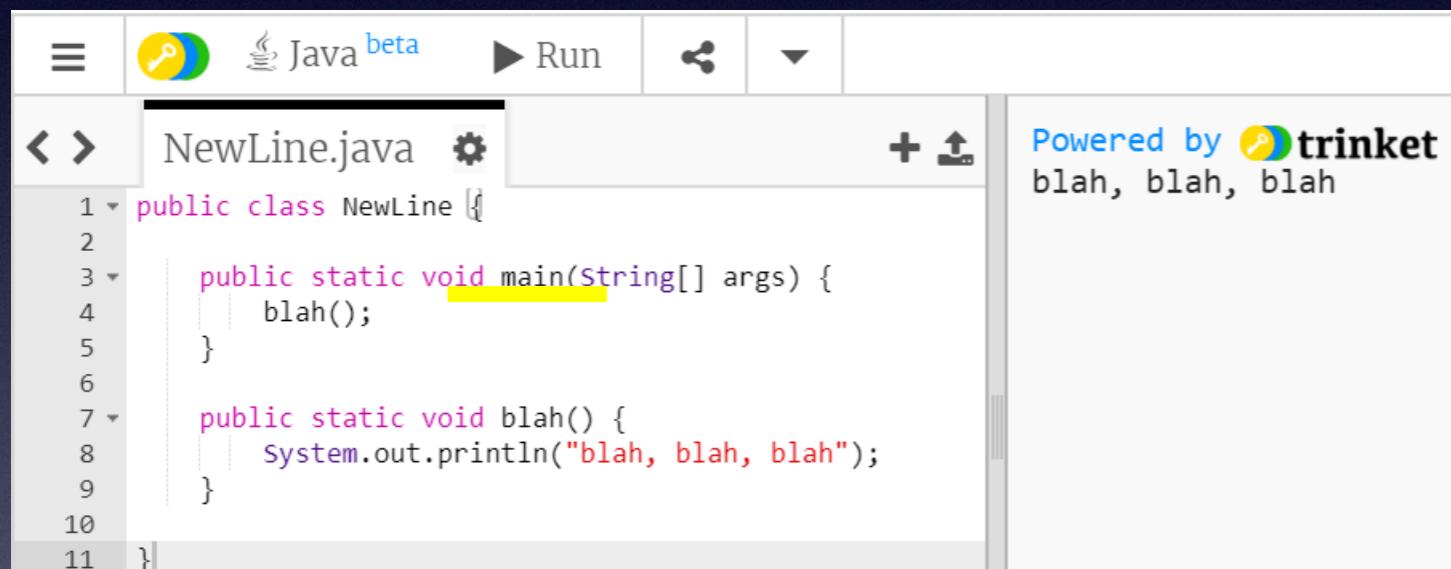
```
sketch_220914a | Processing  
Function call  
sketch_220914a  
void setup() {  
 2   blah();  
3 }  
4 void blah() {  
5   System.out.println("blah, blah, blah");  
6 }  
7 void draw() {  
8 }  
9  
10 
```

Function Definition. Needs a "type" for the return value. In this case there is no return value (**void**). It is an "effect" function!

Console Errors

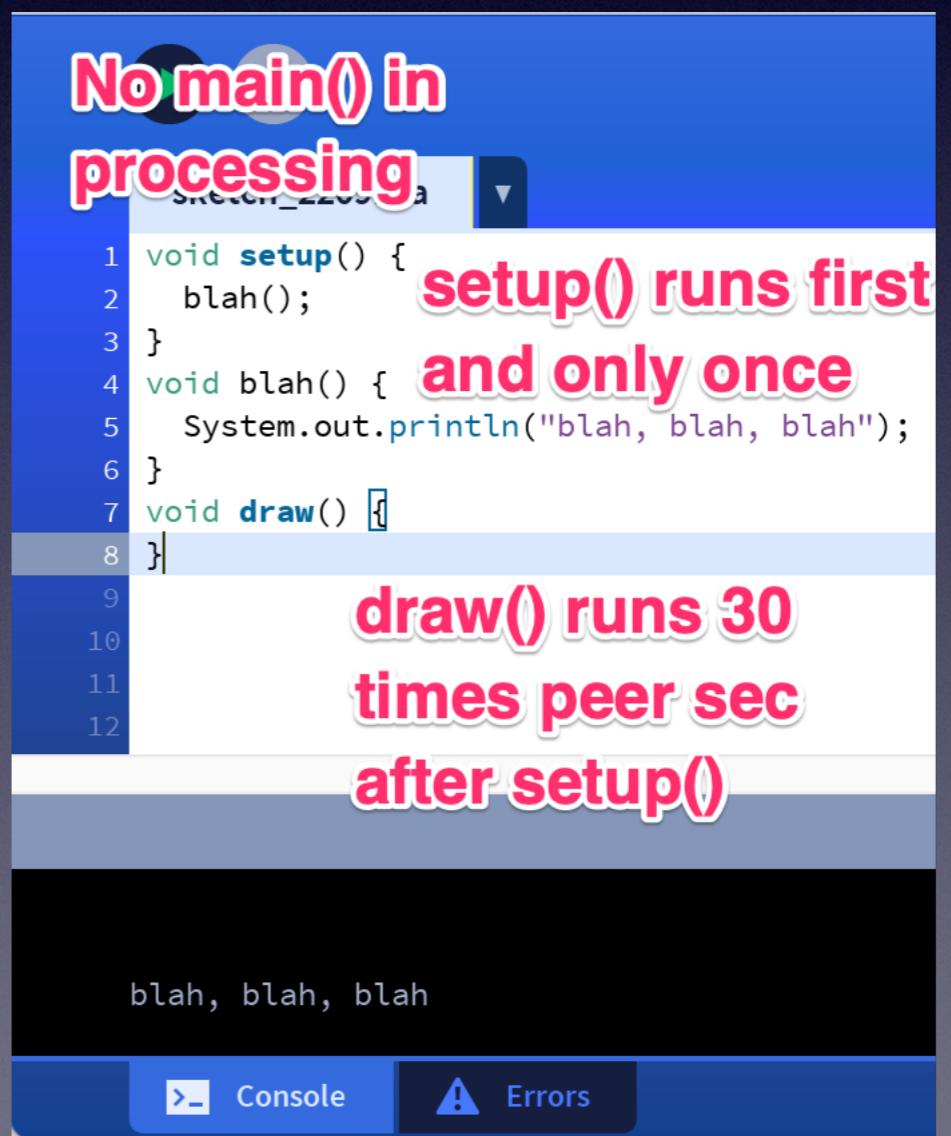
void functions (aka methods)

- Note that the order the functions are *defined* is unimportant
- Java always runs the **main** function first



```
NewLine.java
public class NewLine {
    public static void main(String[] args) {
        blah();
    }

    public static void blah() {
        System.out.println("blah, blah, blah");
    }
}
```



No main() in processing

```
void setup() {
    blah();
}
void blah() {
    System.out.println("blah, blah, blah");
}
void draw() {
```

setup() runs first
and only once

draw() runs 30
times per sec
after setup()

blah, blah, blah

Console Errors

void functions (aka methods)

- The order of the function *calls* is important
- Functions can be reused as many times as you want

```
sketch_220914a
```

```
2 blah();
3 boring();
4 blah();
5 }
6 void blah() {
7     System.out.println("blah, blah, blah");
8 }
9 void boring() {
10    System.out.println("boooooring ...");
11 }
12
13
```

```
blah, blah, blah
boooooring ...
blah, blah, blah
```

- Variable declarations in the parenthesis of the function definition are called *parameters*
- The function call has matching *arguments*
- The values in variables **h** & **m** are copied into variables **hour** & **minute**

The screenshot shows a Java code editor window titled "print_time | Pr". The code defines a class named "print_time" with the following content:

```
1 int h = 7;
2 int m = 35;
3
4 void printTime(int hour, int minute) {
5     System.out.print(hour);
6     System.out.print(":");
7     System.out.print(minute);
8 }
9
10 void setup() {
11     printTime(h, m);
12 }
13 void draw()
```

Annotations with arrows highlight specific parts of the code:

- A red arrow points from the variable declaration `int h = 7;` to the parameter `int hour` in the `printTime` method signature.
- A green arrow points from the variable declaration `int m = 35;` to the parameter `int minute` in the `printTime` method signature.
- A red arrow points from the `h` variable in the `setup` method call to the `h` parameter in the `printTime` method signature.
- A green arrow points from the `m` variable in the `setup` method call to the `minute` parameter in the `printTime` method signature.

The status bar at the bottom of the editor shows the time "7:35".

Arguments must match Parameters in number, order and type

- What's the problem?
- Order, the parameters are **int String** order but the arguments are in **String int** order

The screenshot shows a Java code editor with the following code:

```
some_function
1 void someFunction(int num, String word) {
2     System.out.println(num + ", " + word);
3 }
4
5 void setup() {
6     int n = 1;
7     String w = "one";
8     someFunction(w, n);
9 }
10
11 void draw() {
12 }
```

A red arrow points from the parameter declaration in line 1 to the argument in line 8. A blue arrow points from the argument in line 8 to the function call in line 8. A blue arrow also points from the function name in line 8 to the function definition in line 1.

The function "someFunction()" expects parameters like: "someFunction(int, String)"

Variables can only be used int the function where they are declared

- What's the problem?
- **n** and **w** were **declared** in the **main** function and are not available in **someFunction**

The screenshot shows the Processing 4.0b2 interface with the following code:

```
some_function
1 void someFunction(int num, String word) {
2     System.out.println(n + ", " + w);
3 }
4
5 void setup() {
6     int n = 1;
7     String w = "one";
8     someFunction(n, w);
9 }
10
11 void draw() {
12 }
```

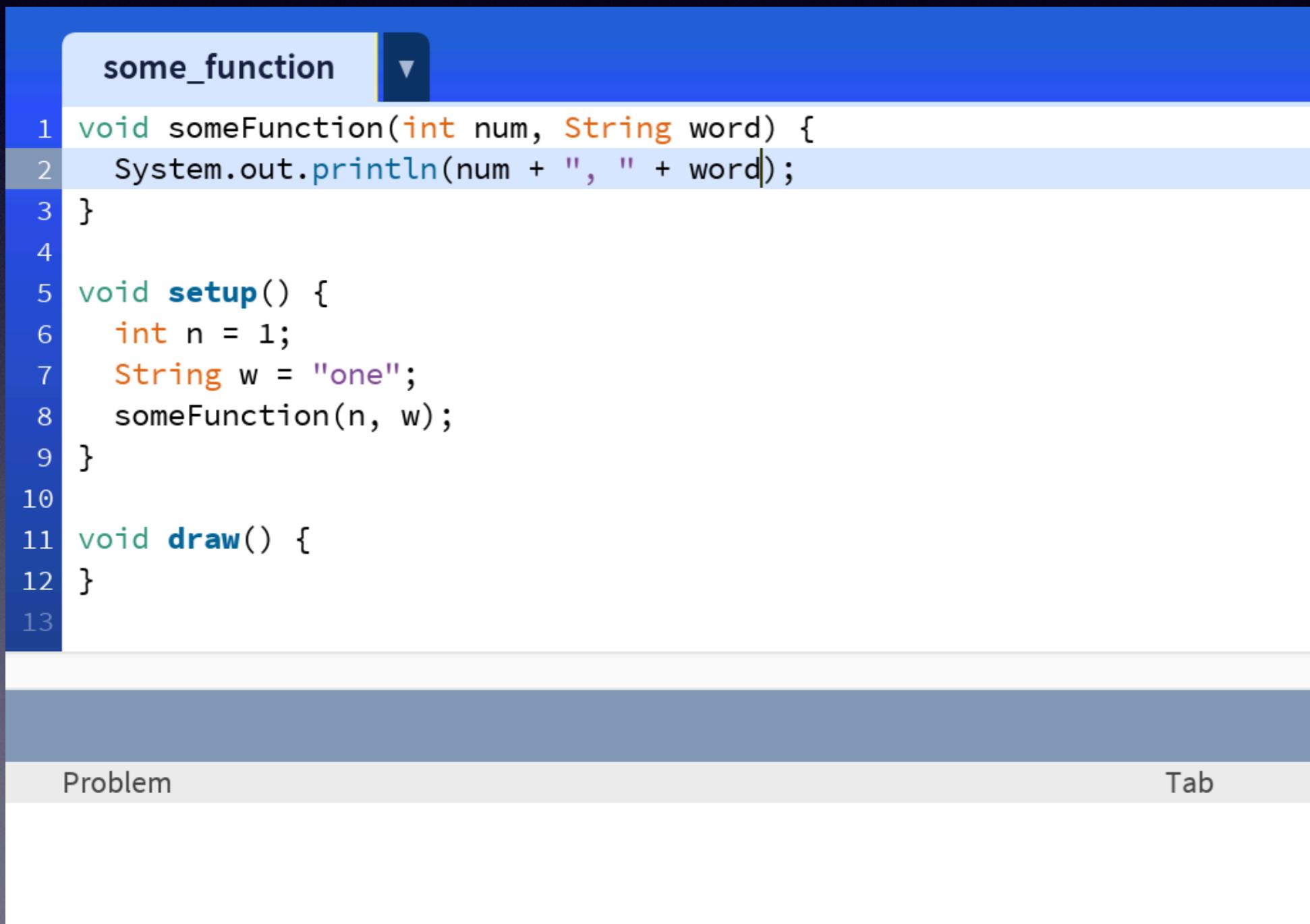
A red error bar at the bottom highlights the line `System.out.println(n + ", " + w);` with the message: "The variable "n" does not exist". Below the code editor, a table details the error:

Problem	Tab	Line
• The variable "n" does not exist	some_function	2
• The variable "w" does not exist	some_function	2

At the bottom of the interface, there are tabs for "Console" and "Errors".

Now the program is correct

- The arguments match the parameters in number order and type
- The correct variables are used in each function



The screenshot shows a code editor window with a dark theme. At the top, there is a tab labeled "some_function". The main area contains the following Java code:

```
1 void someFunction(int num, String word) {  
2     System.out.println(num + ", " + word);  
3 }  
4  
5 void setup() {  
6     int n = 1;  
7     String w = "one";  
8     someFunction(n, w);  
9 }  
10  
11 void draw() {  
12 }  
13
```

The code uses color-coded syntax highlighting: green for keywords like `void`, `int`, `String`, and `System.out.println`; orange for types like `int` and `String`; and purple for strings like `"one"`. The code editor has a "Problem" tab at the bottom left and a "Tab" button at the bottom right.

PrintTime Function Assignment

Complete the PrintTime Function Assignment by typing in code below plus your code individually. Submit to slack. Complete the Google Classroom Assignment

```
print_time ▾
1 void printEuropeanTime(int hour, int minute) {
2     // Your Java code
3 }
4
5 void printAmericanTime(int hour, int minute) {
6     // Your Java code
7 }
8
9 void setup() {
10    String dy = "Wednesday";
11    String mn = "September";
12    int dt = 20;
13    int yr = 1961;
14    int h = 7;
15    int m = 35;
16    // Insert Function call to printAmericanTime
17    // Insert Functioncall to printEuropeanTime
18 }
```

Problem	Tab	Line
The value of the parameter hour is not used	print_time	1
The value of the parameter minute is not used	print_time	1
The value of the parameter hour is not used	print_time	5
The value of the parameter minute is not used	print_time	5

Unit 0-C

Variables Types and Operators

Unit 0-C

- Variables
- Types
- Declarations
- Initializations
- Comments
- % (modulus) and Integer division
- + and Strings

Variables

- Think of a variable as a place to store a value that you will use later
- The value of a variable can *change* (think *vary*)
- A Java variable has size limits for its *type* (for example, integers are limited to values between -2,147,483,648 and 2,147,483,647)
- Every Java variable has a *type* and a *name*

Variables: Parking space analogy

- Like a parking space, a variable can store a value (think *vehicle*) until you need it later



Variables: Parking space analogy

- Parking spaces are often labeled so you can find your car later when you need it
- Lets say that **G210** is the *name* of this parking space



Variables: Parking space analogy

- In addition to a *name*, parking spaces can have a *type* that sets size limits



What is the error?

- We are trying to print an integer that is too large for Java's integer size

The screenshot shows the Processing IDE interface. The title bar reads "too_large | Processing 4.0b2". The sketch window contains the following Java code:

```
1 void setup() {  
2     System.out.println(1234567890123456789);  
3 }  
4  
5 void draw() {  
6 }
```

The line `System.out.println(1234567890123456789);` is highlighted in blue, indicating it is the source of the error. A red error message at the bottom of the IDE states: "The literal 1234567890123456789 of type int is out of range". Below this message is a detailed error log table:

Problem	Tab	Line
• The literal 1234567890123456789 of type int is out of range	too_large	2

At the bottom of the IDE, there are tabs for "Console" and "Errors".

Primitive data types

- In Java (unlike Python or JavaScript) variables have *types*
- Each type has size limits
- For this class, you need to know 5 basic (aka “primitive”) types:
 - **int**
 - **float**
 - **double**
 - **boolean**
 - **char**

Primitive data types

- **int** holds a single integer value between -2,147,483,648 and 2,147,483,647
- **float** a decimal value with up to 7 digits
- **double** a decimal value with up to 15 digits
- A **boolean** can only hold values that evaluate to either **true** or **false**
- **char** holds a single letter, digit, space or punctuation mark and must be enclosed in *single quotes*, like this: 'G'

float vs. double

- **float** is short for *floating point*, another name for *decimal point*
- **double** gets its name because of its size, it has about twice as many digits as a **float**

The screenshot shows the Processing 4.0b2 interface with a sketch titled "sketch_220914e". The code compares the output of a `double` variable and a `float` variable:

```
1 double dNum = 5/9.0;
2 System.out.println(dNum);
3 System.out.println((float)dNum);
```

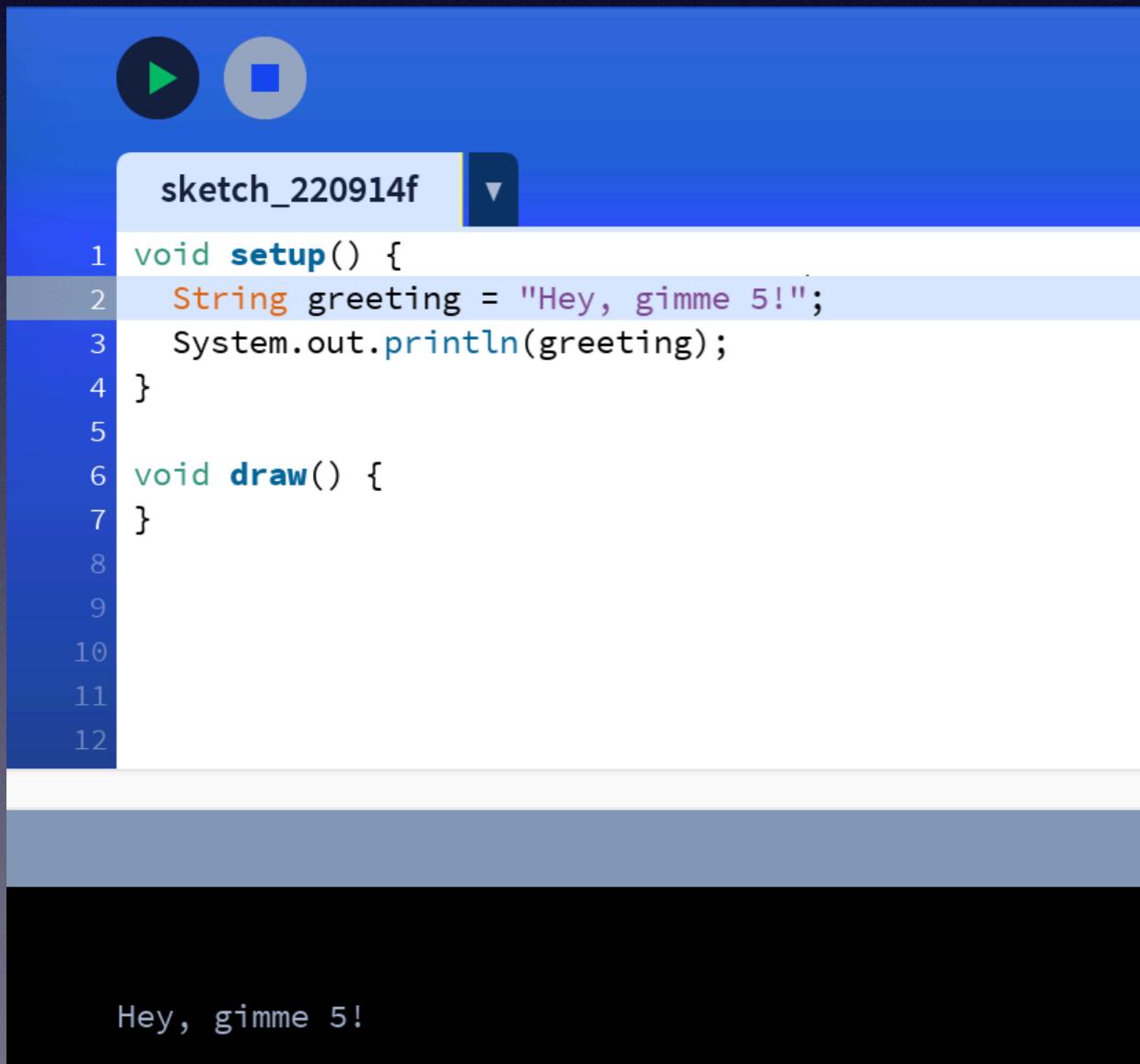
The console output shows two results:

```
0.5555555820465088
0.5555556
```

The "Console" tab is selected at the bottom.

String variables

- A **String** variable can store text with any number of letters, digits, punctuation marks and spaces
- The beginning and end of the text is marked with double quotes ("")



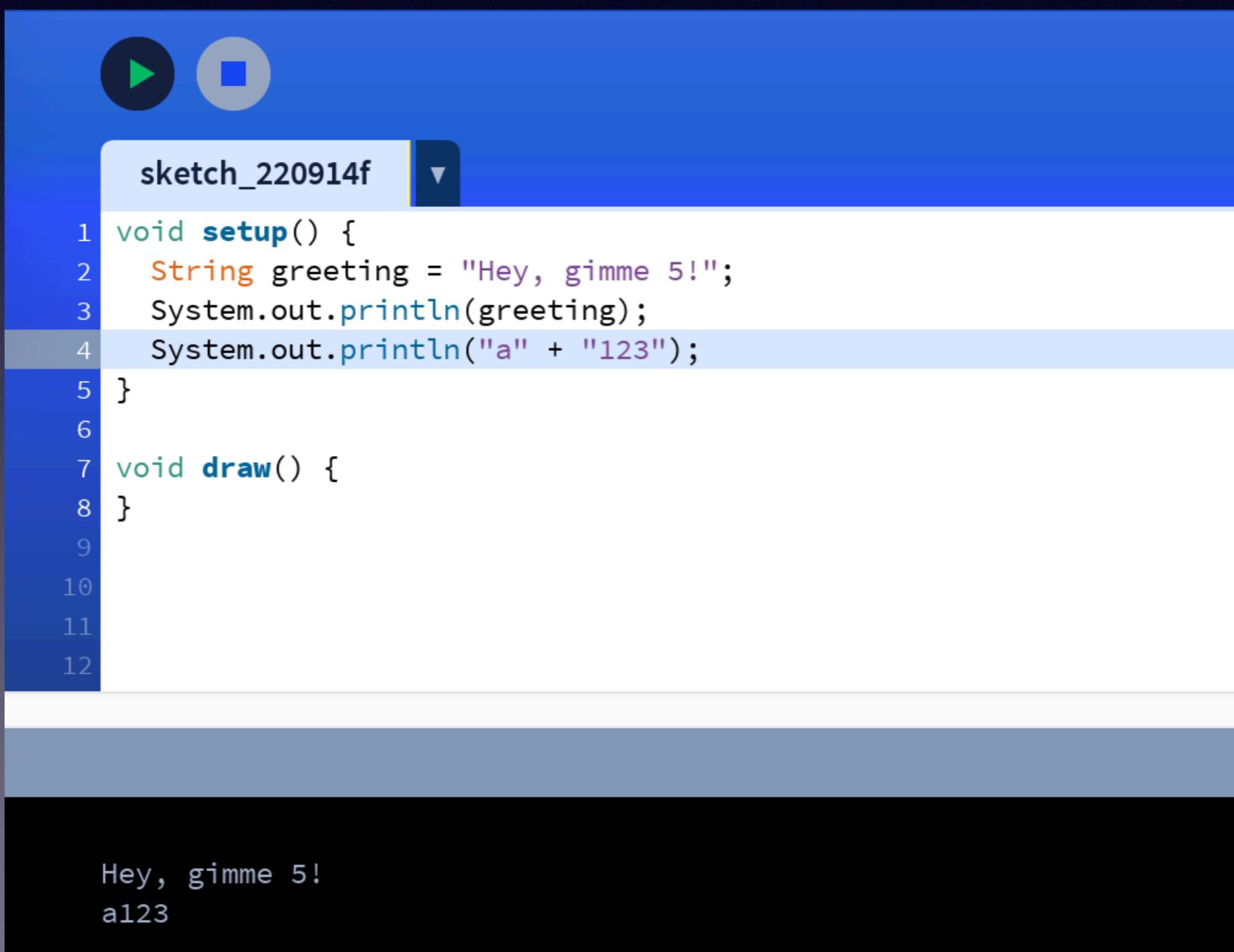
```
sketch_220914f

1 void setup() {
2   String greeting = "Hey, gimme 5!";
3   System.out.println(greeting);
4 }
5
6 void draw() {
7 }
8
9
10
11
12
```

Hey, gimme 5!

+ and Strings

- Using **+** with **Strings** isn't addition arithmetic, it's called *concatenation*
- That's a fancy word that means making bigger **Strings** out of little ones



The screenshot shows the Arduino IDE interface. At the top, there are two buttons: a green play button and a grey square button. Below them is a dropdown menu showing "sketch_220914f". The main area contains the following code:

```
1 void setup() {  
2     String greeting = "Hey, gimme 5!";  
3     System.out.println(greeting);  
4     System.out.println("a" + "123");  
5 }  
6  
7 void draw() {  
8 }  
9  
10  
11  
12
```

Below the code, the output window displays the results of the println statements:

```
Hey, gimme 5!  
a123
```

+ and Strings

- Java executes from left to right so, `1 + 2` is `3`, and `3 + "Hello"` is `"3Hello"`
- `"Hello" + 1` is `"Hello1"`, and `"Hello1" + 2` is `"Hello12"`
- What's the output?

```
System.out.println(1 + 2 + "Hello");
```

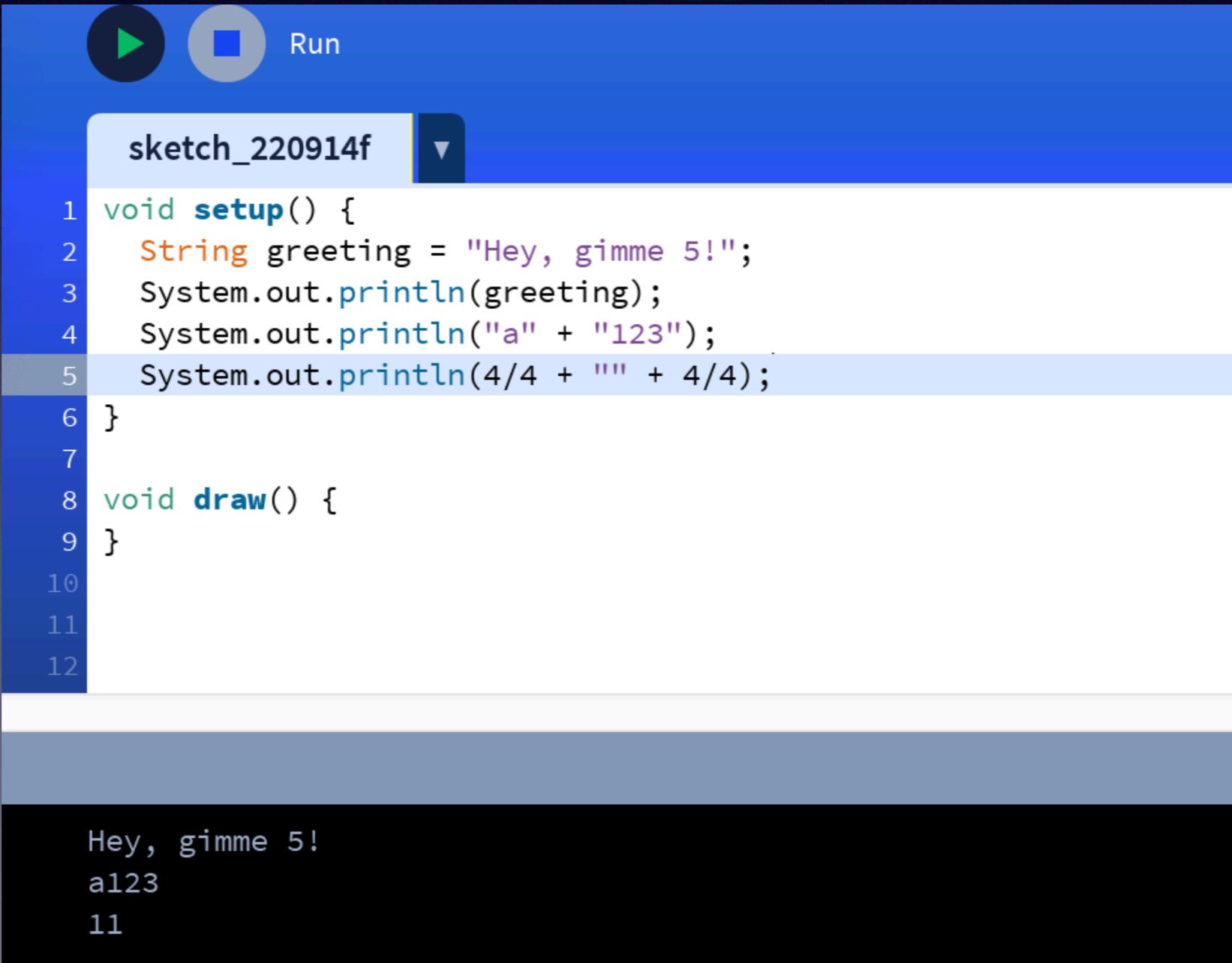
// the output is

```
System.out.println("Hello" + 1 + 2);
```

// the output is

Unique way to make 11 using four 4s

- We could make 11 with four 4s by putting an empty **String** in the middle
- (**String** concatenation is not allowed in the rules of the four 4s challenge though)



```
sketch_220914f

1 void setup() {
2     String greeting = "Hey, gimme 5!";
3     System.out.println(greeting);
4     System.out.println("a" + "123");
5     System.out.println(4/4 + "" + 4/4);
6 }
7
8 void draw() {
9 }
10
11
12
```

Hey, gimme 5!
a123
11

A Java variable declaration

- The Java statement that sets up a variable is called a *declaration*

- Here's an example declaration

```
int num;
```

- The first word of the declaration is the **type**
- The second word is the **name** that the programmer chooses

Variable names and keywords

- You can pretty much choose any name you want for a variable with a few limitations:
 - Variable names cannot
 - start with a number
 - contain a space
 - have a special meaning in Java
 - Java has about 50 reserved words or *keywords* that you are not allowed to use as variable names such as **public**, **class**, **static**, **void**, **int** ...

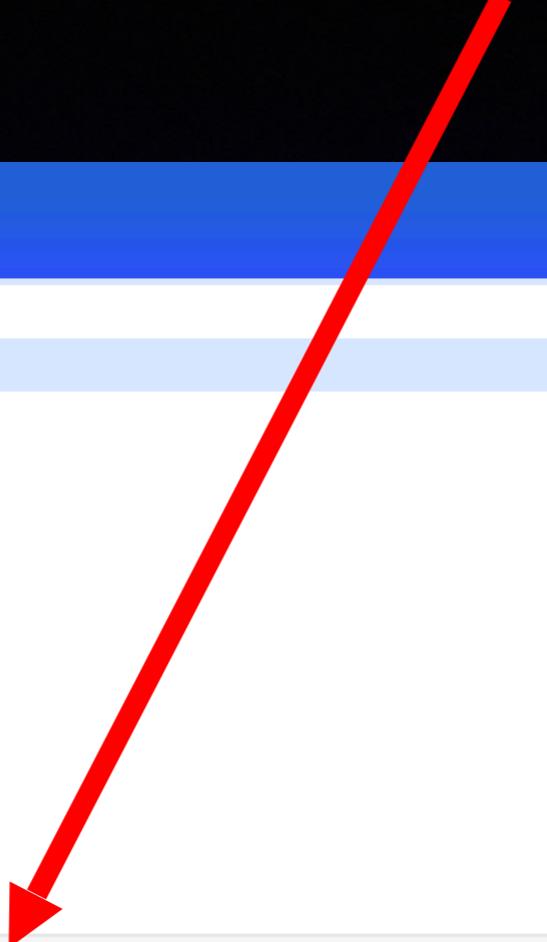
camelCase

- Because a Java variable name can't have spaces, a style called **camelCase** is usually used for variable names with more than one word
- **camelCase** capitalizes the first letter of each word except the first word
- Examples: **firstName**, **numberOfDogs**
- Java variable names are case-sensitive, so **firstName** is not the same as **firstname** or **FirstName**.

What's the error?

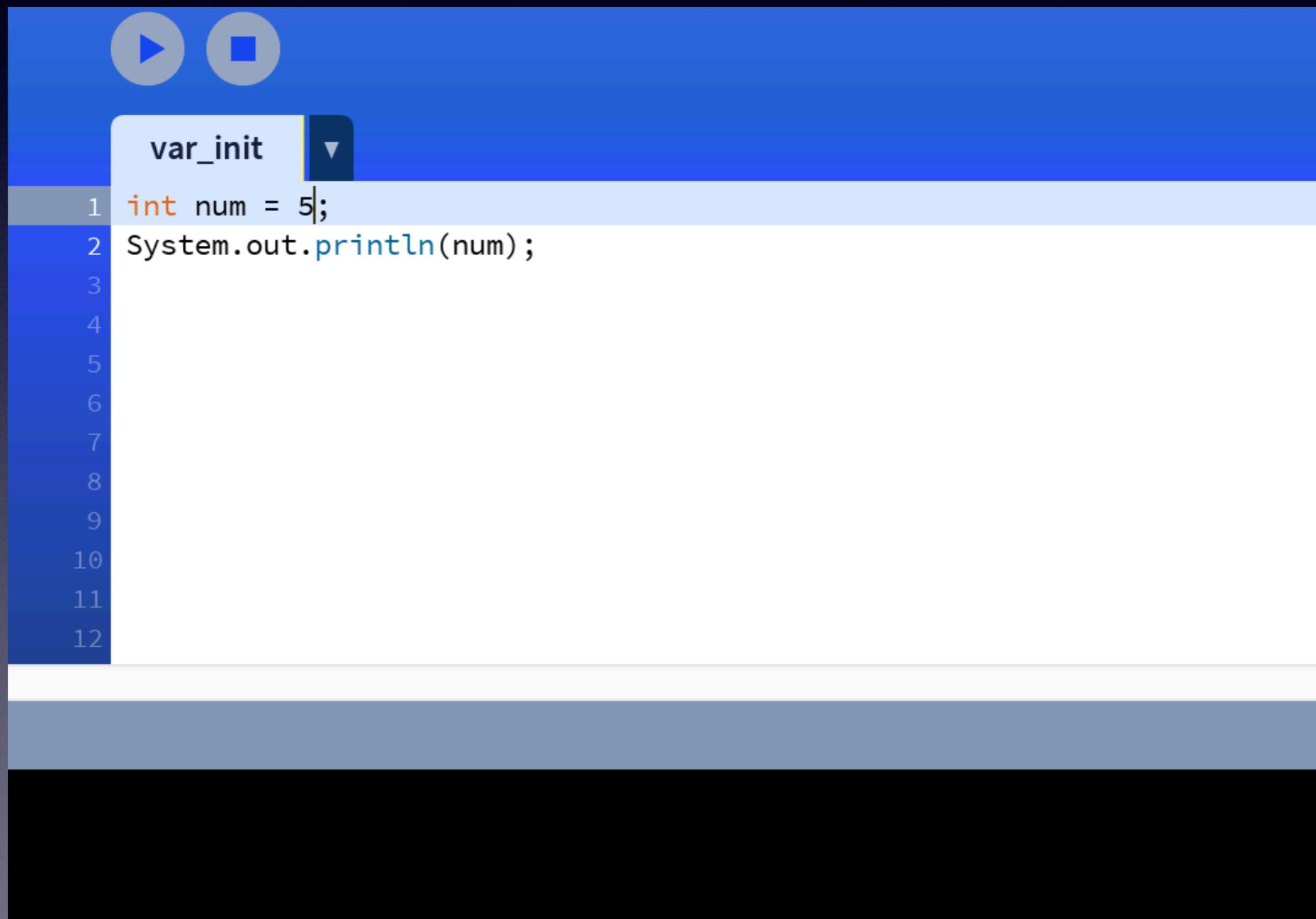
```
var_init ▾  
1 int num;  
2 System.out.println(num);  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12
```

The local variable "num" may not have been initialized



Variable initialization

- After a Java variable is declared, it needs to be *initialized*



The image shows a screenshot of a Java code editor. At the top, there are two circular icons: a play button on the left and a square button on the right. Below them, the code is displayed:

```
var_init
1 int num = 5;
2 System.out.println(num);
3
4
5
6
7
8
9
10
11
12
```

The code consists of two lines: line 1 declares an integer variable 'num' and initializes it to 5; line 2 prints the value of 'num' to the console. Lines 3 through 12 are blank.

Variable initialization

- The English word “initial” means *first*
- We initialize a variable by **assigning** (“setting it equal to”) its *first* value

```
var_init ▾
1 int num ;
2
3 void setup() {
4     num = 17;
5     System.out.println("num = " + num);
6 }
7
8
9
10
11
12
```

num = 17

Declare *and* initialize

- You can **declare** and **initialize** a variable with one line of code:

```
int num = 3;
```

- Just remember that the one line of code is doing two different steps: the **declaration** and the **initialization**

Comments

```
//Single Line
```

```
/*
```

```
Multi  
Line
```

```
*/
```

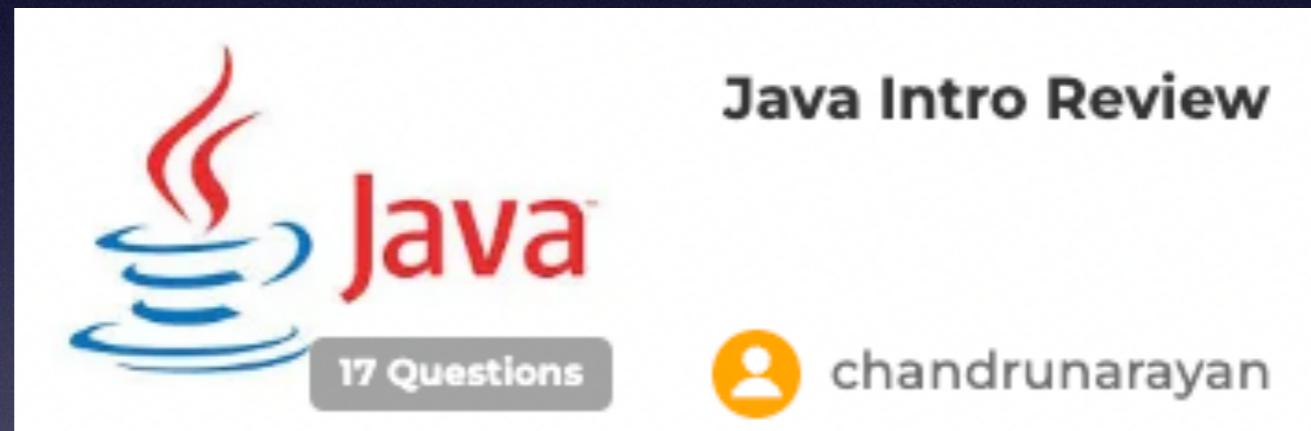
- Tells the computer to ignore some text

Used to:

- Write notes to yourself or other programmers

- Temporarily disable code

Kahoot!



A Kahoot! game card for "Java Intro Review". The card features the Java logo (a steaming coffee cup) and the word "Java" in red. A grey button below the logo says "17 Questions". To the right, the title "Java Intro Review" is displayed above a user profile icon (a yellow circle with a person silhouette) and the name "chandrunarayan".

Java Intro Review

17 Questions

chandrunarayan