

# Unit 3 - Lesson 5

## Preconditions and Postconditions



# Warm Up





# CSA Bingo

 **You should have:**

- a CS Bingo card
- pen / pencil / marker



**CS Bingo**

C

O

D

E

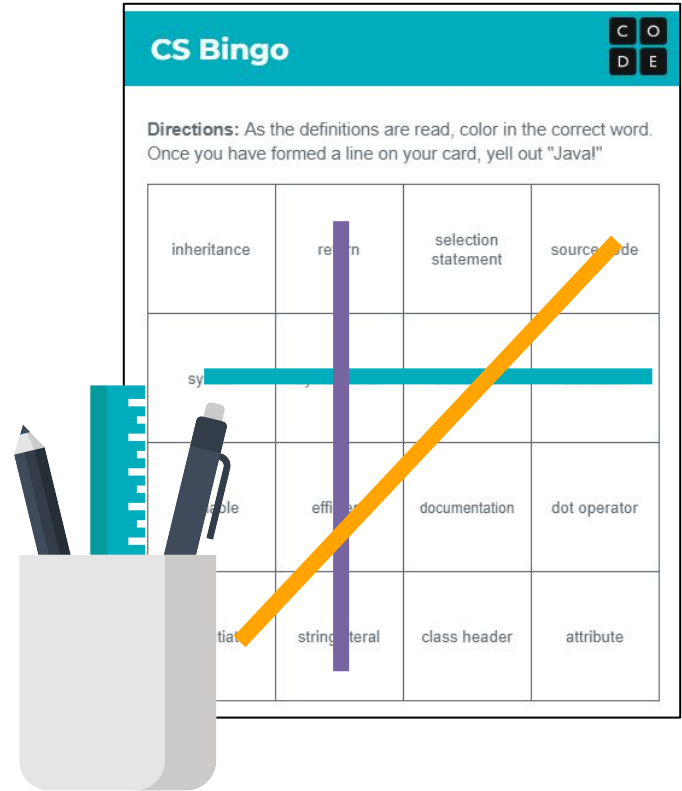
**Directions:** As the definitions are read, color in the correct word. Once you have formed a line on your card, yell out "Java!"

parameterized constructor	primitive type	application program interface (API)	index
encapsulation	refactor	local variable	access modifier
Boolean expression	element	increment	mutator method
instance variable	override	operand	reference type

# CSA Bingo

As the definitions are read,  
**color in the correct word** on  
your **CS Bingo card**.

Once you have **formed a line**  
on your card, yell out **"Java!"**





# CS Bingo!

As the definitions are read,  
**color in the correct word** on  
your **CS Bingo card**.

Once you have **formed a line**  
on your card, yell out **"Java!"**

**CS Bingo**

**Directions:** As the definitions are read, color in the correct word. Once you have formed a line on your card, yell out "Java!"

inheritance	return	selection statement	source code
system	variable	documentation	dot operator
initial	string literal	class header	attribute

# Activity



# Lesson Objectives

By the end of this lesson, you will be able to . . .

- Identify the preconditions and postconditions of an algorithm
- Implement an algorithm to calculate the sum of the values in a one-dimensional (1D) array
- Implement an algorithm to calculate the average of the values in a one-dimensional (1D) array



## Question of the Day

How can I perform calculations on the values stored in a 1D array?



# Investigate and Modify



Navigate to Lesson 5, Level 1

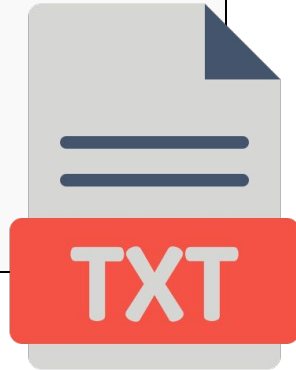


## Do This:

1. Investigate the code
2. Make changes as prompted and observe the results

```
Hours Per Day
number of hours the respondent listens
to music per day

3
1.5
4
2.5
4
5
3
1
6
1
3
8
3
```



A **text file** is a file that contains **letters, numbers,** and/or **symbols** but has **no special formatting**.

We can use text files to save **larger amounts of data** and read the **values** into **1D arrays** in our programs.



```
public double giveDiscount(double price) {  
    double discountedPrice = price / 2;  
    return discountedPrice;  
}
```

## Non-void methods with parameters can:

- Receive values through parameters



```
public double giveDiscount(double price) {  
    double discountedPrice = price / 2;  
    return discountedPrice;  
}
```

## Non-void methods with parameters can:

- Receive values through parameters
- Use those values



```
public double giveDiscount(double price) {  
    double discountedPrice = price / 2;  
    return discountedPrice;  
}
```

## Non-void methods with parameters can:

- Receive values through parameters
- Use those values
- Return a computed value of the specified type



```
/*  
 * Precondition: price is greater than 2.50  
 */  
public double giveDiscount(double price) {  
    double discountedPrice = price / 2;  
    return discountedPrice;  
}
```

A **precondition** is a **condition** that must **always** be **true** just **before** the execution of a code segment.



```
/*  
 * Postcondition: price is half of the original value  
 */  
public double giveDiscount(double price) {  
    double discountedPrice = price / 2;  
    return discountedPrice;  
}
```

A **postcondition** is a **condition** that must **always** be **true** just **after** the execution of a code segment.



HOLD that  
THOUGHT



## Discuss:

- ▶ What **scenarios or tasks** have you encountered that have **preconditions** or **postconditions**?
- ▶ Why do these **matter**?





## Discuss:

Consider this scenario.

- ▶ What would be the **precondition** for this algorithm?
- ▶ What would be the **postcondition**?

### Count Favorite Genres

Count the number of times "Classical" appears as a favorite genre.

```
/*  
 * Precondition: price is greater than 2.50  
 * Postcondition: price is half of the original value  
 */  
public double giveDiscount(double price) {  
    double discountedPrice = price / 2;  
    return discountedPrice;  
}
```

Software engineers **write method code** to **satisfy the postconditions** when **preconditions are met**.

```
/*  
 * Precondition: numberOrdered is less than inventory  
 * Postcondition: newInventory is assigned the inventory remaining  
 */  
public int updateInventory(int inventory, int numberOrdered) {  
    int newInventory = inventory - numberOrdered;  
    return newInventory;  
}
```



**Discuss:** Consider this method. What should we **add or modify** to make sure the **preconditions are met** so we can **satisfy the postconditions**?



```
/*  
 * Precondition: price is greater than 2.50  
 * Postcondition: price is half of the original value  
 */  
public double giveDiscount(double price) {  
    double discountedPrice = price;  
    if (price > 2.5) {  
        discountedPrice = price / 2;  
    }  
    return discountedPrice;  
}
```

**One** way we can do this is to use an **if** statement to **check if price is greater than 2.5**. The **discountedPrice** will **only be updated** if this is true.

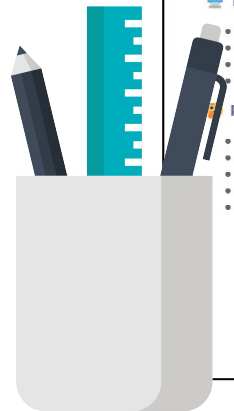




# Writing Algorithms

 **You and your partner should have:**

- Writing Algorithms with Arrays activity guide
- pen / pencil

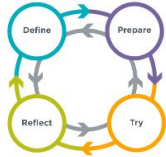


Name(s) \_\_\_\_\_ Period \_\_\_\_\_ Date \_\_\_\_\_

## Activity Guide - Writing Algorithms with Arrays

### Algorithms and the Problem-Solving Process

The **Problem Solving Process** is useful when planning and writing algorithms. This process will help you clarify and break down a problem into manageable steps so you can easily identify the code you need to write for each step.



**Define**

- Read the instructions carefully to make sure you understand the goals.
- Rephrase the problem in your own words.
- Identify any new skills you are being asked to apply.
- If there is starter code, read it to understand what it does.

**Prepare**

- Write out or draw the steps you need to take to solve the problem.
- List what you already know how to do and what you don't yet.
- Explain your algorithm to a classmate.
- Review similar programs that you've written in the past.

**Try**

- Write one small piece at a time.
- Test your program often.
- Use comments to document what your code does.
- Go back to a previous step if you get stuck or don't know whether you've solved the problem.

**Reflect**

- Compare your program to the defined problem to make sure you've solved all aspects of the problem.
- Ask a classmate to try your program and note places where they struggle or show confusion.
- Ask a classmate to read your code to make sure that your documentation is clear and accurate.
- Try to "break" your program to find types of interaction or input that you could handle better.
- Identify changes or improvements you can make next to your program.

1



## Do This:

1. Choose a **scenario**.
2. Identify the **precondition(s)** and **postcondition(s)** for the algorithm.
3. Write **pseudocode** for an algorithm to find the **sum** and the **average** of **all values** in the 1D array.

Name(s) \_\_\_\_\_ Period \_\_\_\_\_ Date \_\_\_\_\_

### Activity Guide - Writing Algorithms with Arrays

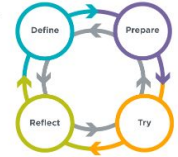


#### Algorithms and the Problem-Solving Process

The **Problem Solving Process** is useful when planning and writing algorithms. This process will help you clarify and break down a problem into manageable steps so you can easily identify the code you need to write for each step.

##### Define

- Read the instructions carefully to make sure you understand the goals.
- Rephrase the problem in your own words.
- Identify any new skills you are being asked to apply.
- If there is starter code, read it to understand what it does.



##### Prepare

- Write out or draw the steps you need to take to solve the problem.
- List what you already know how to do and what you don't yet.
- Explain your algorithm to a classmate.
- Review similar programs that you've written in the past.

##### Try

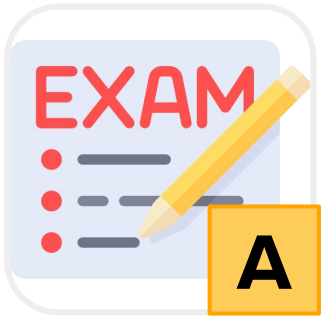
- Write one small piece at a time.
- Test your program often.
- Use comments to document what your code does.
- Go back to a previous step if you get stuck or don't know whether you've solved the problem.

##### Reflect

- Compare your program to the defined problem to make sure you've solved all aspects of the problem.
- Ask a classmate to try your program and note places where they struggle or show confusion.
- Ask a classmate to read your code to make sure that your documentation is clear and accurate.
- Try to "break" your program to find types of interaction or input that you could handle better.
- Identify changes or improvements you can make next to your program.



# Practice



Navigate to Lesson 5, Level 2

 **Do This:**

1. **Level 2** – Check for Understanding
2. **Level 3** – Implement your algorithm to find the sum of all values in a 1D array
3. **Level 4** – Implement your algorithm to find the average of all values in a 1D array

# Wrap Up



# Glows, Grows, Want-to-Knows



**GLOWS**

What was awesome about writing your code?

What is one action you can take to improve your code?

**GROWS**



**WANT TO KNOWS**

What questions do you have about today?





## Today, you learned about . . .

- Identifying the preconditions and postconditions of an algorithm
- How to implement an algorithm to calculate the sum of the values in a one-dimensional (1D) array
- How to implement an algorithm to calculate the average of the values in a one-dimensional (1D) array



## Question of the Day

How can I perform calculations on the values stored in a 1D array?



## Key Vocabulary

- **text file:** a file that contains letters, numbers, and/or symbols but has no special formatting
- **precondition:** a condition that must always be true just before the execution of a code segment
- **postcondition:** a condition that must always be true just after the execution of a code segment