[redacted]

Instructions and consent form

CS[redacted] Student, Welcome to the CS Survey!

- Many questions just ask for your opinion - those questions do not have right or wrong answers.
- Please complete the survey on your own.
- Please do not use outside resources (unless directed to).
- Please do not discuss the survey with classmates until they complete it.
- You cannot change your answers after you move on to the next page.

Thank you for doing the survey!

Please read the consent form below or at this link:[redacted]

Do you agree to consent to the research? By selecting 'I agree', you are confirming the three following conditions.
1) I confirm that I have read this consent document and have had the opportunity to ask questions.
2) I confirm that I am 18 or older.
3) I agree to participate in this research study and allow you to use and disclose information about me for this study, as you have explained in this document.
If you agree, you will proceed to the study survey. If you disagree, simply close your browser window.

○ I agree
○ I do not agree

Your instructor will give extra credit for doing the survey. **Please enter your [redacted] if you want to be awarded credit [redacted]**

Writing directions

The next few questions will ask you to write code.

- Please write in **java**
- To make sure your code **compiles**, please check your answer with a compiler (either your own or an online one like https://www.tutorialspoint.com/compile_java_online.php)
- If you need to make the text boxes longer or wider, click and drag the lower right corner
- Do not use the TAB key. Use spaces instead.
- We're sorry tabs don't work in this survey - don't worry about perfectly consistent indentation.

writing: validWord

Write a function that takes a String as input and **returns** a boolean. The first and last lines are provided for you.

- When the input starts with "A", has at least 6 characters, and ends with "b", return true. Otherwise, return false. Assume that word is never null.

Hints:

- `word.startsWith("x")` will return `true` when the `String word` starts with "x".
- `word.endsWith("x")` will return `true` when the `String word` ends with "x".
  - If word is empty (that is, `""`), `startsWith()` and `endsWith()` will return `false`.
- `word.length()` returns the number of characters in the word.
  - If word is empty (that is, `""`), this method will return 0.

Input	Returned
"Absorb"	true
"ABSORB"	false
"abcB"	false
""	false

```
public static boolean validWord(String word) {
}
```

Writing: okNotOk

Write a function that takes two ints as input and **returns** a String. The first and last lines are provided for you.

- When the first input divided by the second is 5 or larger, and the first input is bigger than 10, return the String "Ok".
- Otherwise, return the String “Not Ok”.

Hint: Be careful to avoid an exception caused by dividing by `0`.

Examples:

Input	**Returned**
36, 6	"Ok"
6, 36	"Not Ok"
20, 0	"Not Ok"

```
public static String okNotOk(int num1, int num2) {
```

```
}
```

Writing - atat

Write a function that takes a String as input and **prints** a message. The first and last lines are provided for you.

- The printed message should follow this general template: `"Your word was <input>. <input_with_ending> is length <length>. All done!"`
- The specific ending depends on what the input ends with. Follow the table below.
- Assume the input is never null or empty.
- Write the function so that it would be **easy for someone else to modify**, for example if they wanted to change part of the template.

HINTS

- `word.endsWith("ish")` will return `true` if the String word ends with "ish"
- `word.length()` will return the length of the String word
- `+` will concatenate Strings. For example`:`
  - `String word1 = "hello";`
  - `String word2 = word1 + " there";`
  - word2 is "`hello there`".

	Example Input	Special Ending	Desired Printed Message
Input ends with "ish"	"Fish"	-ishness	"Your word was Fish. Fish-ishness is length 12. All done!"
Input ends with "at"	"Hat"	-atat	"Your word was Hat. Hat-atat is length 8. All done!"
Input ends with "ness"	"Messyness"	-bess	"Your word was Messyness. Messyness-bess is length 14. All done!"
All other inputs	"Bird"	[none]	"Your word was Bird. Bird is length 4. All done!"

```
public static void yourWord(String word) {
}
```

Editing direction

The next few questions will ask you to Edit the code for the style (if there is room for improvement).

- To make sure your **modified code has the same functionality with the original code**, please check your answer with a compiler (either your own or an online one like https://www.tutorialspoint.com/compile_java_online.php)
- If you need to make the text boxes longer or wider, click and drag the lower right corner
- Do not use the TAB key. Use spaces instead.
- We're sorry tabs don't work in this survey - don't worry about perfectly consistent indentation.

Edit: validTarget

Consider the following code block. We want the **code block to have a good style, as an expert would view it. However, we do not want the code functionality to change** (that is, if the new code and the original code were called with the same input, they should have the same output).

```
public static boolean closeEnoughFractions(int num1, int den1, in
   if (den1 != 0) {
      if(den2 != 0) {
         if (Math.abs(num1/den1 - num2/den2) < fudgeFactor) {
            return true;
         }
      }
   }
   return false;
}
```

Do you think the code already has the best style, or can it be improved?

- ○ Yes I can improve it
- ○ No, the code already has the best style

You said you can improve the style. Copy and paste the code into the box bellow and fix its style without changing its functionality. Note: You don't need to add comments.

```
public static boolean closeEnoughFractions(int num1, int den1, in
   if (den1 != 0) {
      if(den2 != 0) {
         if (Math.abs(num1/den1 - num2/den2) < fudgeFactor) {
            return true;
         }
      }
   }
   return false;
}
```

Edit: printMessage

Consider the following code block. We want the **code block to have a good style, as an expert would view it. However, we do not want the code functionality to change** (that is, if the new code and the original code were called with the same input, they should have the same output).

```
public static void printMessage(double grade) {
//grade is always between 1 and 100
    if (grade < 60) {
        System.out.println("You did not pass the test");
    }
    if (grade >= 60) {
        if (grade < 70) {
            System.out.println("Your grade is C, come to office h
        }
        if (grade >= 70 && grade < 80) {
            System.out.println("Your grade is B");
        }
        if (grade >= 80 && grade < 90) {
            System.out.println ("Good work! Your grade is B+ ");
        }
        if(grade >= 90) {
            System.out.println ("Great job! You got an A");
        }
    }
}
```

Do you think the code already has the best style, or can it be improved?

- ○ Yes I can improve it
- ○ No, the code already has the best style

You said you can improve the style. Copy and paste the code into the box bellow and fix its style without changing its functionality. Note: You don't need to add comments.

```
public static void printMessage(double grade) {
//grade is always between 1 and 100
    if (grade < 60) {
        System.out.println("You did not pass the test");
    }
    if (grade >= 60) {
        if (grade < 70) {
            System.out.println("Your grade is C, come to office h
        }
        if (grade >= 70 && grade < 80) {
            System.out.println("Your grade is B");
        }
        if (grade >= 80 && grade < 90) {
            System.out.println ("Good work! Your grade is B+ ");
        }
        if(grade >= 90) {
            System.out.println ("Great job! You got an A");
        }
    }
}
```

Edit: salePrice

Consider the following code block. We want the **code block to have a good style, as an expert would view it. However, we do not want the code functionality to change** (that is, if the new code and the original code were called with the same input, they should have the same output).

```
public static double salePrice(String item, double price, boolean
   double sale = .25;
   double specialSale = .5;
   double over50sale = .35;
   String beginning = "Your item, " + item + ", usually  costs "
   String ending = "you are getting it for ";
   if (item.equals("socks") && coupon) {
      String returnMessage = "";
      returnMessage = beginning;
      returnMessage += "since you have a coupon, " ;
```

```
        returnMessage += ending;
        double discountPrice = price * (1 - specialSale);
        returnMessage += discountPrice;
        System.out.println(returnMessage);
        double amountSaved = price * specialSale;
        return amountSaved;
    } else if (price > 50) {
        String returnMessage = "";
        returnMessage = beginning;
        returnMessage += "since it is over 50, " ;
        returnMessage += ending;
        double discountPrice = price * (1 - over50sale);
        returnMessage += discountPrice;
        System.out.println(returnMessage);
        double amountSaved = price * over50sale;
        return amountSaved;
    } else {
        String returnMessage = "";
        returnMessage = beginning;
        returnMessage += ending;
        double discountPrice = price * (1 - sale);
        returnMessage += discountPrice;
        System.out.println(returnMessage);
        double amountSaved = price * sale;
        return amountSaved;
    }
}
```

Do you think the code already has the best style, or can it be improved?

- ○ Yes I can improve it
- ○ No, the code already has the best style

You said you can improve the style. Copy and paste the code into the box bellow and fix its style without changing its functionality. Note: You don't need to add comments.

```
public static double salePrice(String item, double price, boolean
   double sale = .25;
   double specialSale = .5;
   double over50sale = .35;
   String beginning = "Your item, " + item + ", usually  costs "
   String ending = "you are getting it for ";
   if (item.equals("socks") && coupon) {
      String returnMessage = "";
      returnMessage = beginning;
      returnMessage += "since you have a coupon, " ;
      returnMessage += ending;
      double discountPrice = price * (1 - specialSale);
      returnMessage += discountPrice;
      System.out.println(returnMessage);
      double amountSaved = price * specialSale;
      return amountSaved;
   } else if (price > 50) {
      String returnMessage = "";
      returnMessage = beginning;
      returnMessage += "since it is over 50, " ;
      returnMessage += ending;
      double discountPrice = price * (1 - over50sale);
      returnMessage += discountPrice;
      System.out.println(returnMessage);
      double amountSaved = price * over50sale;
      return amountSaved;
   } else {
      String returnMessage = "";
      returnMessage = beginning;
      returnMessage += ending;
```

```
        double discountPrice = price * (1 - sale);
        returnMessage += discountPrice;
        System.out.println(returnMessage);
        double amountSaved = price * sale;
        return amountSaved;
    }
}
```

countryList read

All of these code samples do the same thing. Which one makes it easiest for YOU to figure out what the code does?
Note: The `get()` function returns null when the key does not exist in the Map.

- ○
```
public static void addVisitedCity(String country, String city, Map<Str
   ArrayList<String> visitedCitiesSequence = visitedCountries.get(coun
   if (!visitedCountries.containsKey(country)) {
      visitedCitiesSequence = new ArrayList();
      visitedCountries.put(country, visitedCitiesSequence);
   }
   visitedCitiesSequence.add(city);
}
```
- ○
```
public static void addVisitedCity(String country, String city, Map<Str
   ArrayList visitedCitiesSequence = visitedCountries.get(country);
   if (!visitedCountries.containsKey(country)) {
       visitedCitiesSequence= new ArrayList();
       visitedCitiesSequence.add(city);
       visitedCountries.put(country, visitedCitiesSequence);
    } else {
       visitedCitiesSequence.add(city);
    }
}
```
- ○
```
public static void addVisitedCity(String country, String city, Map<Str
   if (!visitedCountries.containsKey(country)) {
      visitedCountries.put(country, new ArrayList<String>());
   }
   ArrayList<String> visitedCitiesSequence = visitedCountries.get(coun
   visitedCitiesSequence.add(city);
}
```
- ○
```
public static void addVisitedCity(String country, String city, Map<Str
    if (!visitedCountries.containsKey(country)) {
        ArrayList visitedCitiesSequence = new ArrayList();
        visitedCitiesSequence.add(city);
        visitedCountries.put(country, visitedCitiesSequence);
    } else {
        ArrayList visitedCitiesSequence = visitedCountries.get(country
        visitedCitiesSequence.add(city);
    }
}
```

countryList style

These are the same code samples as the previous question. **They may be in a different order**.

Which one would **an expert** say is the most elegant, idiomatic, and revealing of design intent?
Note: The `get()` function returns null when the key does not exist in the Map.

- ◯
```
public static void addVisitedCity(String country, String city, Map<Str
   ArrayList visitedCitiesSequence = visitedCountries.get(country);
   if (!visitedCountries.containsKey(country)) {
       visitedCitiesSequence= new ArrayList();
       visitedCitiesSequence.add(city);
       visitedCountries.put(country, visitedCitiesSequence);
    } else {
       visitedCitiesSequence.add(city);
    }
}
```

- ◯
```
public static void addVisitedCity(String country, String city, Map<Str
    if (!visitedCountries.containsKey(country)) {
        ArrayList visitedCitiesSequence = new ArrayList();
        visitedCitiesSequence.add(city);
        visitedCountries.put(country, visitedCitiesSequence);
    } else {
        ArrayList visitedCitiesSequence = visitedCountries.get(country
        visitedCitiesSequence.add(city);
    }
}
```

- ◯
```
public static void addVisitedCity(String country, String city, Map<Str
   ArrayList visitedCitiesSequence = visitedCountries.get(country);
   if (!visitedCountries.containsKey(country)) {
      visitedCitiesSequence = new ArrayList();
      visitedCountries.put(country, visitedCitiesSequence);
   }
   visitedCitiesSequence.add(city);
}
```

- ◯
```
public static void addVisitedCity(String country, String city, Map<Str
   if (!visitedCountries.containsKey(country)) {
      visitedCountries.put(country, new ArrayList<String>());
   }
   ArrayList<String> visitedCitiesSequence = visitedCountries.get(coun
   visitedCitiesSequence.add(city);
}
```

- ◯ An expert would say they all have equal style

evenCounter read

All of these code samples do the same thing. Which one makes it easiest for YOU to figure out what the code does?

- ○
```
public static int evenCounter(String numWord) {
   int total = 0;
   for (int i = 0; i < numWord.length(); i++) {
       char curChar = numWord.charAt(i);
       if (!Character.isDigit(curChar)) {
           continue;
       }
       if (Character.getNumericValue(curChar) % 2 == 0) {
           total++;
       }
   }
   return total;
}
```

- ○
```
public static int evenCounter(String numWord) {
   int total = 0;
   for (int i = 0; i < numWord.length(); i++) {
       char curChar = numWord.charAt(i);
       if (Character.isDigit(curChar)) {
           if (Character.getNumericValue(curChar) % 2 == 0) {
               total++;
           }
       }
   }
   return total;
}
```

- ○
```
public static int evenCounter(String numWord) {
   int total = 0;
   for (int i = 0; i < numWord.length(); i++) {
       char curChar = numWord.charAt(i);
       if (Character.isDigit(curChar) &&
           Character.getNumericValue(curChar) % 2 == 0) {
           total++;
       }
   }
   return total;
}
```

evenCounter style

These are the same code samples as the previous question. They all do the same thing, but the options are shuffled. Which one would **an expert** say is the most elegant, idiomatic, and revealing of design intent?

- ○
```
public static int evenCounter(String numWord) {
   int total = 0;
   for (int i = 0; i < numWord.length(); i++) {
       char curChar = numWord.charAt(i);
       if (Character.isDigit(curChar)) {
           if (Character.getNumericValue(curChar) % 2 == 0) {
               total++;
           }
       }
   }
   return total;
}
```

- ○
```
public static int evenCounter(String numWord) {
   int total = 0;
   for (int i = 0; i < numWord.length(); i++) {
       char curChar = numWord.charAt(i);
       if (Character.isDigit(curChar) &&
           Character.getNumericValue(curChar) % 2 == 0) {
           total++;
       }
   }
   return total;
}
```

- ○
```
public static int evenCounter(String numWord) {
   int total = 0;
   for (int i = 0; i < numWord.length(); i++) {
       char curChar = numWord.charAt(i);
       if (!Character.isDigit(curChar)) {
           continue;
       }
       if (Character.getNumericValue(curChar) % 2 == 0) {
           total++;
       }
   }
   return total;
}
```

- ○ An expert would say they all have equal style.

size read

All of these code samples do the same thing. Which one makes it easiest for YOU to figure out what the code does?

○
```
public static String size(int i) {
   String size = "";
   if (i < 10) {
      size = "small";
   } else if (i >= 10 && i < 20) {
      size = "medium";
   } else if (i >= 20) {
      size = "big";
   }
   return "The size is " + size;
}
```

○
```
public static String size(int i) {
   String size = "";
   if (i < 10) {
      size = "small";
   }
   if (i >= 10 && i < 20) {
      size = "medium";
   }
   if (i >= 20) {
      size = "big";
   }
   return "The size is " + size;
}
```

○
```
public static String size(int i) {
   String size = "";
   if (i < 10) {
      size = "small";
   } else if (i < 20) {
      size = "medium";
   } else {
      size = "big";
   }
   return "The size is " + size;
}
```

size style

These are the same code samples as the previous question. They all do the same thing, but the options are shuffled. Which one would **an expert** say is the most elegant, idiomatic, and revealing of design intent?

- ○

```
public static String size(int i) {
   String size = "";
   if (i < 10) {
      size = "small";
   } else if (i >= 10 && i < 20) {
      size = "medium";
   } else if (i >= 20) {
      size = "big";
   }
   return "The size is " + size;
}
```

- ○

```
public static String size(int i) {
   String size = "";
   if (i < 10) {
      size = "small";
   }
   if (i >= 10 && i < 20) {
      size = "medium";
   }
   if (i >= 20) {
      size = "big";
   }
   return "The size is " + size;
}
```

- ○

```
public static String size(int i) {
   String size = "";
   if (i < 10) {
      size = "small";
   } else if (i < 20) {
      size = "medium";
   } else {
      size = "big";
   }
   return "The size is " + size;
}
```

- ○ An expert would say they all have equal style.

guessWord read

All of these code samples do the same thing. Which one makes it easiest for YOU to figure out what the code does?
Note: `lastLetter()` returns the last `char` of the word and `firstLetter()` returns the first `char` of the word. Assume these methods are defined elsewhere.

○
```
public static boolean guessWord(Set<String> validWords, String lastWor
        return validWords.contains(newWord) &&
               lastLetter(lastWord) == firstLetter(newWord) &&
               newWord.contains(String.valueOf(firstLetter(lastWord)))
}
```

○
```
public static boolean guessWord(Set<String> validWords, String lastWor
       if (validWords.contains(newWord)) {
            if (lastLetter(lastWord) == firstLetter(newWord)) {
                if (newWord.contains(String.valueOf(firstLetter(lastWo
                     return true;
                }
            }
       }
       return false;
 }
```

○
```
public static boolean guessWord(Set<String> validWords, String lastWor
        if (validWords.contains(newWord)) {
            return lastLetter(lastWord) == firstLetter(newWord) &&
                   newWord.contains(String.valueOf(firstLetter(lastWor
       }
       return false;
}
```

○
```
public static boolean guessWord(Set<String> validWords, String lastWor
    if (validWords.contains(newWord) &&
        lastLetter(lastWord) == firstLetter(newWord) &&
        newWord.contains(String.valueOf(firstLetter(lastWord)))) {
            return true;
     }
     return false;
}
```

guessWord style

These are the same code samples as the previous question. They all do the same thing, but the options are shuffled. Which one would **an expert** say is the most elegant, idiomatic, and revealing of design intent?
Note: `lastLetter`() returns the last `char` of the word and `firstLetter()` returns the first `char` of the word. Assume these methods are defined elsewhere.

○
```
public static boolean guessWord(Set<String> validWords, String lastWor
        return validWords.contains(newWord) &&
               lastLetter(lastWord) == firstLetter(newWord) &&
               newWord.contains(String.valueOf(firstLetter(lastWord)))
}
```

○
```
public static boolean guessWord(Set<String> validWords, String lastWor
       if (validWords.contains(newWord)) {
            if (lastLetter(lastWord) == firstLetter(newWord)) {
                if (newWord.contains(String.valueOf(firstLetter(lastWo
                     return true;
                }
            }
       }
       return false;
 }
```

○
```
public static boolean guessWord(Set<String> validWords, String lastWor
    if (validWords.contains(newWord) &&
        lastLetter(lastWord) == firstLetter(newWord) &&
        newWord.contains(String.valueOf(firstLetter(lastWord)))) {
            return true;
     }
     return false;
}
```

○
```
public static boolean guessWord(Set<String> validWords, String lastWor
        if (validWords.contains(newWord)) {
            return lastLetter(lastWord) == firstLetter(newWord) &&
                   newWord.contains(String.valueOf(firstLetter(lastWor
       }
       return false;
}
```

○ An expert would say they all have equal style.

comprehension

For this section, please do NOT use any IDE or compiler

Comprehension: addMap1

Consider two Maps. Both are Map<String, List<String>> (for ease of reading, each String printed below is separated with a comma):

Map1: { Canada=[Montreal, Vancouver], USA=[Atlanta, Salt Lake City], Japan=[Tokyo, Tokyo] }

Map2: { Canada=[Montreal, Vancouver], USA=[Atlanta, Los Angeles], Mexico=[Mexico City] }

Which option below best shows the contents of Map2 after executing `mapFunc(Map1, Map2)`? The `mapFunc` function is below.

- ○ Map2: { Canada=[Vancouver, Montreal], USA=[Atlanta] }
- ○ Map2: { USA=[Los Angeles, Salt Lake City], Japan=[Tokyo], Mexico=[Mexico City] }
- ○ Map2: { Canada=[Vancouver, Montreal], USA=[Atlanta, Los Angeles, Salt Lake City], Japan=[Tokyo], Mexico=[Mexico City] }
- ○ Map2: { Canada=[Vancouver, Montreal, Vancouver, Montreal], USA=[Altanta, Los Angeles, Atlanta, Salt Lake City], Japan=[Tokyo, Tokyo], Mexico=[Mexico City] }

Methods of the Map class (used below):

- `keySet()`: returns a Set view of the keys in the map
- `containsKey(K key)`: returns true if key is in the map, and false otherwise
- `put(K key, V value)`: adds the given key:value pair to the map. If key is already present, modifies the map to pair key with value

- `get(K key)`: returns the value associated with key, or null if the key is not present

```
public static void mapFunc(Map<String, ArrayList<String>> map1, M
    for (String country : map1.keySet()) {
        if (!map2.containsKey(country)) {
            map2.put(country, new ArrayList<String>());
        }
        ArrayList<String> cityList = map2.get(country);
        cityList.addAll(map1.get(country));
        Set<String> citiesNoDuplicates = new HashSet<String>(city
        map2.put(country, new ArrayList<String>(citiesNoDuplicate
    }
 }
```

Comprehension: BirthYear1

What is returned by the following code snippet: canVoteThisYear`(1992, 2024)`; See the method below.

- ○ true
- ○ false
- ○ The code throws an error

What is returned by the following code snippet: canVoteThisYear`(1996, 2023)`; See the method below.

- ○ true
- ○ false
- ○ The code throws an error

```
public static boolean canVoteThisYear(int birthYear, int currentY
    return currentYear - birthYear >= 18 &&
           currentYear - birthYear <= 125 &&
           currentYear % 4 == 0;
}
```

Comprehension: greater

Decide if the code below accomplishes this task:
Task: The function counts the lines of code in a Python file, counting any line that is not a comment and is not empty. All comments in the file start with the hash symbol, '#'.
Hints:

- `line.isEmpty()` returns `true` if the line is empty or only has whitespace.
- `line.trim()` removes whitespace from both ends of string.

Does this code accomplish the task?

```
public static int numberOfLines(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
       String line = scanner.nextLine();
       String trimmedLine = line.trim();
       if (trimmedLine.charAt(0) != '#' && !trimmedLine.isEmpty()
           codeLines++;
       }
   }
   return codeLines;
}
```

- ○ Yes
- ○ No

Comprehension: greater 1

This code is the same as the previous question.

```
public static int numberOfLines(Scanner scanner) {
    int codeLines = 0;
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        String trimmedLine = line.trim();
        if (trimmedLine.charAt(0) != '#' && !trimmedLine.isEmpty(
            codeLines++;
        }
    }
    return codeLines;
}
```

What is returned for each input?

What is returned for the following python file?

Note: the first line of the file starts with "`import`" and the last line starts with "`return`".

```
import math
#This method calculates the area of a circle
def circle_area(radius):
   #raise an error if radius is negative
   if radius < 0:
       raise ValueError("Radius cannot be negative")
   return math.pi * radius**2
```

- ○ 7
- ○ 5
- ○ [ ] Something else:
- ○ Throws an exception

What is returned for the following python file?

Note: the first line of the file starts with "`import`" and the last line starts with "`return`".

```
import math
#This method calculates the area of a circle

def circle_area(radius):
   #raise an error if radius is negative
   if radius < 0:
       raise ValueError("Radius cannot be negative")

   return math.pi * radius**2
```

- ○ 7
- ○ 5
- ○ ______ Something else:
- ○ Throws an exception

Comprehension: greater 2

Decide if the code below accomplishes this task:

Task: The function counts the lines of code in a Python file, counting any line that is not a comment and is not empty. All comments in the file start with the hash symbol, '#'.

Hints:

- line.isEmpty() returns true if the line is empty or only has whitespace.
- line.trim() removes whitespace from both ends of string.

```
public static int numberOfLines1(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
       String line = scanner.nextLine();
       String trimmedLine = line.trim();
       if (!trimmedLine.isEmpty() && trimmedLine.charAt(0) != '#'
           codeLines++;
       }
   }
   return codeLines;
}
```

Does this code (numberOfLines1) accomplish the task?

- Yes
- No

```
public static int numberOfLines2(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
       String line = scanner.nextLine();
       String trimmedLine = line.trim();
       if (!trimmedLine.isEmpty()) {
           if (trimmedLine.charAt(0) != '#') {
               codeLines++;
           }
       }
   }
   return codeLines;
}
```

Does this code (numberOfLines2) accomplish the task?

- Yes
- No

```
public static int numberOfLines3(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
       String line = scanner.nextLine();
       String trimmedLine = line.trim();
       if (trimmedLine.charAt(0) != '#') {
           if (!trimmedLine.isEmpty()) {
               codeLines++;
           }
       }
   }
   return codeLines;
}
```

Does this code (numberOfLines3) accomplish the task?

- Yes
- No

Comprehension: greater 2.1

The next three questions will ask you to determine the output of **different code** snippets for the **same input**.

For the function (numberOfLines1), when the scanner reads the following python file what will be returned?

Note: the first line of the file starts with "`import`" and the last line starts with "`return`".

```
import math
#This method calculates the area of a circle

def circle_area(radius):
   #raise an error if radius is negative
   if radius < 0:
       raise ValueError("Radius cannot be negative")

   return math.pi * radius**2

public static int numberOfLines1(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
      String line = scanner.nextLine();
      String trimmedLine = line.trim();
      if (!trimmedLine.isEmpty() && trimmedLine.charAt(0) != '#')
          codeLines++;
      }
   }
   return codeLines;
}
```

- 7
- 5
- ______ Something else:
- Throws an exception

For the function (numberOfLines2), when the scanner reads the following python file what will be returned?

Note: the first line of the file starts with "`import`" and the last line starts with "`return`".

```
import math
#This method calculates the area of a circle

def circle_area(radius):
   #raise an error if radius is negative
   if radius < 0:
       raise ValueError("Radius cannot be negative")

   return math.pi * radius**2
```

```
public static int numberOfLines2(Scanner scanner) {
    int codeLines = 0;
    while (scanner.hasNextLine()) {
       String line = scanner.nextLine();
       String trimmedLine = line.trim();
       if (!trimmedLine.isEmpty()) {
           if (trimmedLine.charAt(0) != '#') {
               codeLines++;
           }
        }
    }
    return codeLines;
}
```

- ○ 7
- ○ 5
- ○ ______ Something else:
- ○ Throws an exception

For the function (numberOfLines3), when the scanner reads the following python file what will be returned?
Note: the first line of the file starts with "import" and the last line starts with "return".

```
import math
#This method calculates the area of a circle

def circle_area(radius):
   #raise an error if radius is negative
   if radius < 0:
       raise ValueError("Radius cannot be negative")

   return math.pi * radius**2
```

```
public static int numberOfLines3(Scanner scanner) {
    int codeLines = 0;
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        String trimmedLine = line.trim();
        if (trimmedLine.charAt(0) != '#') {
            if (!trimmedLine.isEmpty()) {
                codeLines++;
            }
        }
    }
    return codeLines;
}
```

- ○ 7
- ○ 5
- ○ ______ Something else:
- ○ Throws an exception

Comprehension greater what executes

Consider lines 6 to 7 in this code (numberOfLines1). For the code segments in the "Items" bucket below, what order do conditions execute when trimmedLine is Empty? You may put more than one seqment in each bucket to indicate that they execute at the same time.

Note1: One bucket is for code that does not execute at all for the given case (if any).

Note2: Place the segments in buckets based on when they start executing (regardless of if they complete execution successfully or not)

```
public static int numberOfLines1(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
      String line = scanner.nextLine();
      String trimmedLine = line.trim();
      if (!trimmedLine.isEmpty() && trimmedLine.charAt(0) != '#')
          codeLines++;
      }
   }
   return codeLines;
}
```

Items

!trimmedLine.isEmpty()

trimmedLine.charAt(0) != '#'

codeLines++

First	**Second**

Third	**Does not execute at all in this case**

Consider lines 6 to 8 in this code (numberOfLines2). For the code segments in the "Items" bucket below, what order do conditions execute when trimmedLine is Empty? You may put more than one seqment in each bucket to indicate that they execute at the same time.

Note1: One bucket is for code that does not execute at all for the given case (if any).

Note2: Place the segments in buckets based on when they start executing (regardless of if they complete execution successfully or not)

```
public static int numberOfLines2(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
      String line = scanner.nextLine();
      String trimmedLine = line.trim();
      if (!trimmedLine.isEmpty()) {
         if (trimmedLine.charAt(0) != '#') {
          codeLines++;
         }
      }
   }
   return codeLines;
}
```

Items

!trimmedLine.isEmpty()

trimmedLine.charAt(0) != '#'

codeLines++

First	**Second**

Third	**Does not execute at all in this case**

Consider lines 6 to 8 in this code (numberOfLines3). For the code segments in the "Items" bucket below, what order do conditions execute when trimmedLine is Empty? You may put more than one seqment in each bucket to indicate that they execute at the same time.

Note1: One bucket is for code that does not execute at all for the given case (if any).

Note2: Place the segments in buckets based on when they start executing (regardless of if they complete execution successfully or not)

```
public static int numberOfLines3(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
      String line = scanner.nextLine();
      String trimmedLine = line.trim();
      if (trimmedLine.charAt(0) != '#') {
         if (!trimmedLine.isEmpty()) {
          codeLines++;
         }
      }
   }
   return codeLines;
}
```

Items

!trimmedLine.isEmpty()

trimmedLine.charAt(0) != '#'

codeLines++

First

Second

Third

Does not execute at all in this case

Consider lines 6 to 7 in this code (numberOfLines4). For the code segments in the "Items" bucket below, what order do conditions execute when trimmedLine is Empty? You may put more than one seqment in each bucket to indicate that they execute at the same time.

Note1: One bucket is for code that does not execute at all for the given case (if any).

Note2: Place the segments in buckets based on when they start executing (regardless of if they complete execution successfully or not)

```
public static int numberOfLines4(Scanner scanner) {
   int codeLines = 0;
   while (scanner.hasNextLine()) {
      String line = scanner.nextLine();
      String trimmedLine = line.trim();
      if (trimmedLine.charAt(0) != '#' && !trimmedLine.isEmpty())
          codeLines++;
      }
   }
   return codeLines;
}
```

Items

!trimmedLine.isEmpty()

trimmedLine.charAt(0) != '#'

codeLines++

First

Second

Third

Does not execute at all in this case

Comprehension: word1-2

Do the two functions below do the same thing? That is, if they are each given the same input, will they always return the same thing (or throw the same exception)?

- Yes, word1 and word2 do the same things.
- No, word1 and word2 do different things.

```
public static String word1(String word) {
   if (word.endsWith("?")) {
      word = word.substring(0, word.length() - 1);
   }
   if (word.endsWith("!")) {
      word = word.substring(0, word.length() - 1);
   }
   return word;
}

public static String word2(String word) {
   if (word.endsWith("?")) {
      word = word.substring(0, word.length() - 1);
   } else if (word.endsWith("!")) {
      word = word.substring(0, word.length() - 1);
   }
   return word;
}
```

Comprehension: word 1-2 inputs

These are the same functions as the previous question. For the function word1 (below), what is returned for this input: "Question!?"

- "Question!?"
- "Question!"
- "Question"
- Something else:
- Throws an exception

```
public static String word1(String word) {
   if (word.endsWith("?")) {
      word = word.substring(0, word.length() - 1);
   }
   if (word.endsWith("!")) {
      word = word.substring(0, word.length() - 1);
   }
   return word;
}
```

For the function word2 (below), what is returned for this input: "Question!?"

- "Question!?"
- "Question!"
- "Question"
- Something else:
- Throws an exception

```
public static String word2(String word) {
   if (word.endsWith("?")) {
      word = word.substring(0, word.length() - 1);
   } else if (word.endsWith("!")) {
      word = word.substring(0, word.length() - 1);
   }
   return word;
}
```

Comprehension: word 1-2 comparison operations

These are the same functions as the previous question. For the input "Hello?", In which function(s) will this condition be checked: `(word.endsWith("!"))`

- ○ Only in word1
- ○ Only in word2
- ○ In both word1 and word2

```
public static String word1(String word) {
   if (word.endsWith("?")) {
      word = word.substring(0, word.length() - 1);
   }
   if (word.endsWith("!")) {
      word = word.substring(0, word.length() - 1);
   }
   return word;
}
```

```
public static String word2(String word) {
   if (word.endsWith("?")) {
      word = word.substring(0, word.length() - 1);
   } else if (word.endsWith("!")) {
      word = word.substring(0, word.length() - 1);
   }
   return word;
}
```

Comprehension: magic

Do the two functions below do the same thing? That is, if they are each given the same input, will they always return the same thing (or throw the same exception)?

- ○ Yes, magic1 and magic2 do the same things.
- ○ No, magic1 and magic2 do different things.

```
public static String magic1(int i) {
   int out = 0;
   if (i > 100) {
      out = i + 15;
   }
   if (i < 100) {
      out = i - 15;
   }
   if (i == 100) {
      out = i + 37;
   }
   return "Your number is "+ i +" and your magic number is: " + o
}
```

```
public static String magic2(int i) {
   int out = 0;
   if (i > 100) {
      out = i + 15;
   } else if (i < 100) {
      out = i - 15;
   } else {
      out = i + 37;
   }
   return "Your number is "+ i +" and your magic number is: " + o
}
```

Comprehension: magic comparison

These are the same functions as the previous question. Which function below will perform more comparison operations for this input: 100

- ○ magic1
- ○ magic2
- ○ magic1 and magic2 will both perform the same number of comparisons.

Which function below will perform more comparison operations for this input: 110

- ○ magic1
- ○ magic2
- ○ magic1 and magic2 will both perform the same number of comparisons.

```
public static String magic1(int i) {
   int out = 0;
   if (i > 100) {
      out = i + 15;
   }
   if (i < 100) {
      out = i - 15;
   }
   if (i == 100) {
      out = i + 37;
   }
   return "Your number is "+ i +" and your magic number is: " + o
}
```

```
public static String magic2(int i) {
   int out = 0;
   if (i > 100) {
       out = i + 15;
   } else if (i < 100) {
      out = i - 15;
   } else {
      out = i + 37;
   }
   return "Your number is "+ i +" and your magic number is: " + o
}
```

Comprehension: addMap2

Consider carDealership1 and carDealership2. Both are Map<String, ArrayList<String>> (for ease of reading, each String printed below is separated with a comma):

carDealership1: { Toyota=[Camry , Corolla], Honda=[Civic, Accord], Ford=[Mustang, Mustang] }

carDealership2: { Toyota=[Camry , Corolla], Honda=[Civic, Insight], Chevrolet=[Camaro] }

Which option below best shows the contents of carDealership2 after executing `carBrand(carDealership1, carDealership2)`? The carBrand function is below.

- ○ carDealership2: { Toyota=[Camry , Corolla, Camry , Corolla], USA=[Civic, Insight, Civic, Accord], Ford=[Mustang, Mustang], Chevrolet=[Camaro] }
- ○ carDealership2: { Toyota=[Camry , Corolla], Honda=[Civic, Accord, Insight], Ford=[Mustang], Chevrolet=[Camaro] }
- ○ carDealership2: { Toyota=[Camry , Corolla], Honda=[Civic]}
- ○ carDealership2: { Honda=[Accord, Insight], Ford=[Mustang], Chevrolet=[Camaro] }

Methods of the Map class (used below):

- `keySet()`: returns a Set view of the keys in the map
- `containsKey(K key)`: returns true if key is in the map, and false otherwise
- `put(K key, V value)`: adds the given key:value pair to the map. If key is already present, modifies the map to pair key with value
- `get(K key)`: returns the value associated with key, or null if the key is not present

```
public static void carBrand(Map<String, ArrayList<String>> carDea
    for (String brand : carDealership1.keySet()){
        if (!carDealership2.containsKey(brand)) {
            carDealership2.put(brand, new ArrayList<String>());
            ArrayList<String> carModels = carDealership2.get(bran
            carModels.addAll(carDealership1.get(brand));
            Set<String> carsNoDuplicates = new HashSet<String>(ca
            carDealership2.put(brand, new ArrayList<String>(carsN
        } else {
            ArrayList<String> carModels = carDealership2.get(bran
            carModels.addAll(carDealership1.get(brand));
            Set<String> carsNoDuplicates = new HashSet<String>(ca
            carDealership2.put(brand, new ArrayList<String>(carsN
        }
    }
}
```

Comprehension: BirthYear2

What is returned by the following code snippet: `isValid(1963, 2023)`; See the method below.

- true
- false
- The code throws an error

What is returned by the following code snippet: `isValid(1920, 2023)`; See the method below.

- true
- false
- The code throws an error

```
public static boolean isValid(int birthYear, int currentYear) {
    if (birthYear <= currentYear) {
         if (currentYear - birthYear <= 150) {
```

[redacted]

```
                if (birthYear % 2 == 0) {
                        return true;
                }
        }
    }
    return false;
}
```

Demographics

Would you be interested in doing a follow-up interview about style and readability?

- ○ Yes, [redacted] may email me about participating in a follow-up interview. The best email for me is: ______
- ○ No.

Please select the gender you identify with:

- ○ Female
- ○ Male
- ○ ______ Prefer to self-identify:
- ○ Prefer not to say

Of the questions/instructions on the survey, how many did you understand? (we're asking about the clarity of the prompts, not the correctness of your answers)

- ○ All were clear to me
- ○ Most were clear to me
- ○ About half were clear to me
- ○ Less than half were clear to me