Noah Luddy CodingBat Problems and Solutions

Table of Contents

Java

Warmup-1

The parameter weekday is true if it is a weekday, and the parameter vacation is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in.

```
sleepIn(false, false) → true
sleepIn(true, false) → false
sleepIn(false, true) → true
```

```
public boolean sleepIn(boolean weekday, boolean vacation) {
  if (!weekday || vacation) {
    return true;
  }
  return false;
}
```

We have two monkeys, a and b, and the parameters aSmile and bSmile indicate if each is smiling. We are in trouble if they are both smiling or if neither of them is smiling. Return true if we are in trouble.

```
monkeyTrouble(true, true) → true
monkeyTrouble(false, false) → true
monkeyTrouble(true, false) → false
```

```
public boolean monkeyTrouble(boolean aSmile, boolean bSmile) {
  if (aSmile && bSmile) {
    return true;
  }
  else if (!aSmile && !bSmile) {
    return true;
  }
  else {
    return false;
  }
}
```

Given two int values, return their sum. Unless the two values are the same, then return double their sum.

sumDouble(1, 2) → 3
sumDouble(3, 2) → 5
sumDouble(2, 2) → 8

```
public int sumDouble(int a, int b) {
  if (a == b) {
    return (2*(a+b));
  }
  else {
    return (a+b);
  }
}
```

Given an int n, return the absolute difference between n and 21, except return double the absolute difference if n is over 21.

diff21(19) → 2
diff21(10) → 11
diff21(21) → 0

```
public int diff21(int n) {
  if (n > 21) {
    return 2*(Math.abs(n-21));
  }
  else {
    return Math.abs(n-21);
  }
}
```

We have a loud talking parrot. The "hour" parameter is the current hour time in the range 0..23. We are in trouble if the parrot is talking and the hour is before 7 or after 20. Return true if we are in trouble.

parrotTrouble(true, 6) → true
parrotTrouble(true, 7) → false

parrotTrouble(false, 6) → false

```
public boolean parrotTrouble(boolean talking, int hour) {
  return (talking && (hour < 7 || hour > 20));
}
```

Given 2 ints, a and b, return true if one if them is 10 or if their sum is 10.

makes10(9, 10) → true
makes10(9, 9) → false
makes10(1, 9) → true

```
public boolean makes10(int a, int b) {
  if (a == 10 || b == 10 || (a + b) == 10) {
    return true;
  }
  else {
    return false;
  }
}
```

Given an int n, return true if it is within 10 of 100 or 200. Note: Math.abs(num) computes the absolute value of a number.

nearHundred(93) → true
nearHundred(90) → true
nearHundred(89) → false

```
public boolean nearHundred(int n) {
  if ((Math.abs(100-n) <= 10) || Math.abs(200-n) <= 10) {
    return true;
  }
  else {
    return false;
  }
}
```

Given 2 int values, return true if one is negative and one is positive. Except if the parameter "negative" is true, then return true only if both are negative.

posNeg(1, -1, false) → true
posNeg(-1, 1, false) → true
posNeg(-4, -5, true) → true

```
public boolean posNeg(int a, int b, boolean negative) {
  if ((a < 0 && b > 0 && !negative) || (a > 0 && b < 0 && !negative)) {
    return true;
  } else if (a < 0 && b < 0 && negative) {
    return true;
  } else {
    return false;
  }
}
```

Given a string, return a new string where "not " has been added to the front. However, if the string already begins with "not", return the string unchanged. Note: use .equals() to compare 2 strings.

notString("candy") → "not candy"
notString("x") → "not x"
notString("not bad") → "not bad"

```
public String notString(String str) {
  if (str.length() < 3) {
    return "not " + str;
  } else if (str.substring(0, 3).equals("not")) {
    return str;
  } else {
    return "not " + str;
  }
}
```

Given a non-empty string and an int n, return a new string where the char at index n has been removed. The value of n will be a valid index of a char in the original string (i.e. n will be in the range 0..str.length()-1 inclusive).

missingChar("kitten", 1) → "ktten"
missingChar("kitten", 0) → "itten"
missingChar("kitten", 4) → "kittn"

```
public String missingChar(String str, int n) {
  String a = str.substring(0, n);
  String b = str.substring(n+1, str.length());
  return a + b;
}
```

Given a string, return a new string where the first and last chars have been exchanged.

frontBack("code") → "eodc"
frontBack("a") → "a"
frontBack("ab") → "ba"

```
public String frontBack(String str) {
  if (str.length() <= 1) {
    return str;
  } else {
    char a = str.charAt(0);
    char b = str.charAt(str.length()-1);
    String c = str.substring(1, str.length()-1);
    return b + c + a;
  }
}
```

Given a string, we'll say that the front is the first 3 chars of the string. If the string length is less than 3, the front is whatever is there. Return a new string which is 3 copies of the front.

front3("Java") → "JavJavJav"
front3("Chocolate") → "ChoChoCho"
front3("abc") → "abcabcabc"

```
public String front3(String str) {
  if (str.length() < 3) {
    String front = str;
    return front + front + front;
  } else {
    String front = str.substring(0, 3);
    return front + front + front;
  }
}
```

Given a string, take the last char and return a new string with the last char added at the front and back, so "cat" yields "tcatt". The original string will be length 1 or more.

backAround("cat") → "tcatt"
backAround("Hello") → "oHelloo"
backAround("a") → "aaa"

```
public String backAround(String str) {
  char a = str.charAt(str.length()-1);
  return a + str + a;
}
```

Return true if the given non-negative number is a multiple of 3 or a multiple of 5. Use the % "mod" operator -- see Introduction to Mod

or35(3) → true
or35(10) → true
or35(8) → false

```
public boolean or35(int n) {
  if (n % 3 == 0 || n % 5 == 0) {
    return true;
  } else {
    return false;
  }
}
```

Given a string, take the first 2 chars and return the string with the 2 chars added at both the front and back, so "kitten" yields"kikittenki". If the string length is less than 2, use whatever chars are there.

front22("kitten") → "kikittenki"
front22("Ha") → "HaHaHa"
front22("abc") → "ababcab"

```
public String front22(String str) {
  if (str.length() < 2) {
    String front = str;
    return front + str + front;
  } else {
    String front = str.substring(0, 2);
    return front + str + front;
  }
}
```

Given a string, return true if the string starts with "hi" and false otherwise.

startHi("hi there") → true
startHi("hi") → true
startHi("hello hi") → false

```
public boolean startHi(String str) {
  char h = 'h';
  char i = 'i';

  if (str.length() < 2) {
    return false;
  } else if (str.charAt(0) == h && str.charAt(1) == i) {
    return true;
  } else {
    return false;
  }
}
```

Given two temperatures, return true if one is less than 0 and the other is greater than 100.

icyHot(120, -1) → true
icyHot(-1, 120) → true

icyHot(2, 120) → false

```
public boolean icyHot(int temp1, int temp2) {
  if ((temp1 < 0 || temp2 < 0) && (temp1 > 100 || temp2 > 100)) {
    return true;
  } else {
    return false;
  }
}
```

Given 2 int values, return true if either of them is in the range 10..20 inclusive.

in1020(12, 99) → true
in1020(21, 12) → true
in1020(8, 99) → false

```
public boolean in1020(int a, int b) {
  if ((a > 9 && a < 21) || (b > 9 && b < 21)) {
    return true;
  } else {
    return false;
  }
}
```

We'll say that a number is "teen" if it is in the range 13..19 inclusive. Given 3 int values, return true if 1 or more of them are teen.

hasTeen(13, 20, 10) → true
hasTeen(20, 19, 10) → true
hasTeen(20, 10, 13) → true

```
public boolean hasTeen(int a, int b, int c) {
  if ((a > 12 && a < 20) || (b > 12 && b < 20) || (c > 12 && c < 20)) {
    return true;
  } else {
    return false;
  }
}
```

We'll say that a number is "teen" if it is in the range 13..19 inclusive. Given 2 int values, return true if one or the other is teen, but not both.

loneTeen(13, 99) → true
loneTeen(21, 19) → true
loneTeen(13, 13) → false

```
public boolean loneTeen(int a, int b) {
  if ((a > 12 && a < 20) && (b > 12 && b < 20)) {
    return false;
  } else if (((a > 12 && a < 20) && (b < 13 || b > 19)) || ((b > 12 && b <
20) && (a < 13 || a > 19))) {
    return true;
  } else  {
    return false;
  }
}
```

Given a string, if the string "del" appears starting at index 1, return a string where that "del" has been deleted. Otherwise, return the string unchanged.

delDel("adelbc") → "abc"
delDel("adelHello") → "aHello"
delDel("adedbc") → "adedbc"

```
public String delDel(String str) {
  if (str.length() >= 4 && str.substring(1, 4).equals("del")) {
    return str.substring(0,1) + str.substring(4);
  } else {
    return str;
  }
}
```

Return true if the given string begins with "mix", except the 'm' can be anything, so "pix", "9ix" .. all count.

mixStart("mix snacks") → true
mixStart("pix snacks") → true

mixStart("piz snacks") → false

```
public boolean mixStart(String str) {
  if (str.length() < 3) {
    return false;
  } else if (str.substring(1, 3).equals("ix")) {
    return true;
  } else {
    return false;
  }
}
```

Given a string, return a string made of the first 2 chars (if present), however include first char only if it is 'o' and include the second only if it is 'z', so "ozymandias" yields "oz".

startOz("ozymandias") → "oz"
startOz("bzoo") → "z"
startOz("oxx") → "o"

```
public String startOz(String str) {
  if (str.length() < 2) {
    if (str.length() == 0) {
      return "";
    } else if (str.substring(0, 1).equals("o")) {
      return "o";
    } else if (str.substring(1, 2).equals("z")) {
      return "z";
    } else {
      return "";
    }
  } else if (str.substring(0, 2).equals("oz")) {
    return "oz";
  } else if (str.substring(0, 1).equals("o")) {
    return "o";
  } else if (str.substring(1, 2).equals("z")) {
    return "z";
  } else {
    return "";
  }
}
```

Given three int values, a b c, return the largest.

intMax(1, 2, 3) → 3
intMax(1, 3, 2) → 3
intMax(3, 2, 1) → 3

```
public int intMax(int a, int b, int c) {
  return Math.max(a, Math.max(b, c));
}
```

Given 2 int values, return whichever value is nearest to the value 10, or return 0 in the event of a tie. Note that Math.abs(n) returns the absolute value of a number.

close10(8, 13) → 8
close10(13, 8) → 8
close10(13, 7) → 0

```
public int close10(int a, int b) {
  int c = 10-a;
  int d = 10-b;
  if (Math.abs(c) == Math.abs(d)) {
    return 0;
  } else if (Math.abs(c) < Math.abs(d)) {
    return a;
  } else {
    return b;
  }
}
```

Given 2 int values, return true if they are both in the range 30..40 inclusive, or they are both in the range 40..50 inclusive.

in3050(30, 31) → true
in3050(30, 41) → false
in3050(40, 50) → true

```
public boolean in3050(int a, int b) {
  if (((a > 29 && a < 41) && (b > 29 && b < 41)) || ((a > 39 && a < 51) &&
(b > 39 && b < 51))) {
    return true;
  } else {
    return false;
  }
}
```

Given 2 positive int values, return the larger value that is in the range 10..20 inclusive, or return 0 if neither is in that range.

max1020(11, 19) → 19
max1020(19, 11) → 19
max1020(11, 9) → 11

```
public int max1020(int a, int b) {
  if ((a > 9 && a < 21) && (b > 9 && b < 21)) {
    return Math.max(a, b);
  } else if ((a > 9 && a < 21) && (b < 10 || b > 19)) {
    return a;
  } else if ((a > 19 || a < 10) && (b > 9 && b < 21)) {
    return b;
  } else {
    return 0;
  }
}
```

Return true if the given string contains between 1 and 3 'e' chars.

stringE("Hello") → true
stringE("Heelle") → true
stringE("Heelele") → false

```
public boolean stringE(String str) {
  if (str.indexOf("e") > -1) {
     String newStr = str.substring(str.indexOf("e")+1);
     if (newStr.indexOf("e") > -1) {
        String newNewStr = newStr.substring(newStr.indexOf("e")+1);
        if (newNewStr.indexOf("e") > -1) {
          String newNewNewStr =
newNewStr.substring(newNewStr.indexOf("e")+1);
          if (newNewNewStr.indexOf("e") > -1) {
            return false;
          } else {
            return true;
          }
        } else {
          return true;
        }
      } else {
        return true;
      }
  } else {
    return false;
  }
}
```

Given two non-negative int values, return true if they have the same last digit, such as with 27 and 57. Note that the % "mod" operator computes remainders, so 17 % 10 is 7.

lastDigit(7, 17) → true
lastDigit(6, 17) → false
lastDigit(3, 113) → true

```
public boolean lastDigit(int a, int b) {
  if (a % 10 == b % 10) {
    return true;
  } else {
    return false;
  }
}
```

Given a string, return a new string where the last 3 chars are now in upper case. If the string has less than 3 chars, uppercase whatever is there. Note that str.toUpperCase() returns the uppercase version of a string.

endUp("Hello") → "HeLLO"
endUp("hi there") → "hi thERE"
endUp("hi") → "HI"

```
public String endUp(String str) {
  if (str.length() < 3) {
    return str.toUpperCase();
  } else {
    return str.substring(0, str.length()-3) +
str.substring(str.length()-3).toUpperCase();
  }
}
```

Given a non-empty string and an int N, return the string made starting with char 0, and then every Nth char of the string. So if N is 3, use char 0, 3, 6, ... and so on. N is 1 or more.

everyNth("Miracle", 2) → "Mrce"
everyNth("abcdefg", 2) → "aceg"
everyNth("abcdefg", 3) → "adg"

```
public String everyNth(String str, int n) {
  int length = str.length();
  int words = (length+1)/n;
  String result = "";
  for (int i = 0; i < length; i+=n) {
     result += str.substring(i,i+1);
  }
  return result;
}
```

Warmup-2

Given a string and a non-negative int n, return a larger string that is n copies of the original string.

stringTimes("Hi", 2) → "HiHi"
stringTimes("Hi", 3) → "HiHiHi"
stringTimes("Hi", 1) → "Hi"

```
public String stringTimes(String str, int n) {
  String result = "";
  for (int i = n; i>0; i--) {
    result += str;
  }
  return result;
}
```

Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;

frontTimes("Chocolate", 2) → "ChoCho"
frontTimes("Chocolate", 3) → "ChoChoCho"
frontTimes("Abc", 3) → "AbcAbcAbc"

```
public String frontTimes(String str, int n) {
  String result = "";
  String front = "";
  if (str.length() < 3) {
    front = str;
  } else {
    front = str.substring(0, 3);
  }
  for (int i = n; i > 0; i--) {
    result += front;
  }
  return result;
}
```

Count the number of "xx" in the given string. We'll say that overlapping is allowed, so "xxx" contains 2 "xx".

countXX("abcxx") → 1
countXX("xxx") → 2
countXX("xxxx") → 3

```
public int countXX(String str) {
  int count = 0;
  for (int i = 0; i < str.length()-1; i++) {
     if (str.substring(i,i+2).equals("xx")) {
       count++;
     }
  }
  return count;
}
```

Given a string, return true if the first instance of "x" in the string is immediately followed by another "x".

doubleX("axxbb") → true
doubleX("axaxax") → false
doubleX("xxxxx") → true

```
boolean doubleX(String str) {
  boolean x = false;
  if (str.indexOf("x") == -1) {
    x = false;
  } else {
    int firstX = str.indexOf("x");
    if (str.length()-1 == firstX) {
    x = false;
    } else if (str.substring(firstX, firstX+2).equals("xx")) {
      x = true;
    } else {
      x = false;
    }
  }
  return x;
}
```

Given a string, return a new string made of every other char starting with the first, so "Hello" yields "Hlo".

stringBits("Hello") → "Hlo"
stringBits("Hi") → "H"
stringBits("Heeololeo") → "Hello"

```
public String stringBits(String str) {
  String result = "";
  for (int i = 0; i < str.length(); i+=2) {
     result+=str.substring(i,i+1);
  }
  return result;
}
```

Given a non-empty string like "Code" return a string like "CCoCodCode".

stringSplosion("Code") → "CCoCodCode"
stringSplosion("abc") → "aababc"
stringSplosion("ab") → "aab"

```
public String stringSplosion(String str) {
  String result = "";
  for (int i = 1; i < str.length()+1; i++) {
  result += str.substring(0, i);
  }
  return result;
}
```

Given a string, return the count of the number of times that a substring length 2 appears in the string and also as the last 2 chars of the string, so "hixxxhi" yields 1 (we won't count the end substring).

last2("hixxhi") → 1
last2("xaxxaxaxx") → 1
last2("axxxaaxx") → 2

```
public int last2(String str) {
  if (str.length() < 2) {
    return 0;
  }
  String two = str.substring(str.length()-2);
  int count = 0;
  for (int i = 0; i < str.length()-2; i++) {
    if (str.substring(i, i+2).equals(two)) {
      count++;
    }
  }
  return count;
}
```

Given an array of ints, return the number of 9's in the array.

arrayCount9([1, 2, 9]) → 1
arrayCount9([1, 9, 9]) → 2
arrayCount9([1, 9, 9, 3, 9]) → 3

```
public int arrayCount9(int[] nums) {
  int nine = 0;
  for (int i = 0; i < nums.length; i++) {
    if (nums[i] == 9) {
      nine++;
    }
  }
  return nine;
}
```

Given an array of ints, return true if one of the first 4 elements in the array is a 9. The array length may be less than 4.

arrayFront9([1, 2, 9, 3, 4]) → true
arrayFront9([1, 2, 3, 4, 9]) → false
arrayFront9([1, 2, 3, 4, 5]) → false

```
public boolean arrayFront9(int[] nums) {
  boolean nine = false;
  if (nums.length < 5) {
    for (int i = 0; i < nums.length; i++) {
      if (nums[i] == 9) {
        nine = true;
      }
    }
  } else {
    for (int k = 0; k < 4; k++) {
      if (nums[k] == 9) {
        nine = true;
      }
    }
  }
  return nine;
}
```

Given an array of ints, return true if .. 1, 2, 3, .. appears in the array somewhere.

array123([1, 1, 2, 3, 1]) → true
array123([1, 1, 2, 4, 1]) → false
array123([1, 1, 2, 1, 2, 3]) → true

```
public boolean array123(int[] nums) {
  boolean one = false;
  for (int i = 0; i < nums.length-2; i++) {
    if (nums[i] == 1 && nums[i+1] == 2 && nums[i+2] == 3) {
      one = true;
      break;
    }
  }
  return one;
}
```

Given 2 strings, a and b, return the number of the positions where they contain the same length 2 substring. So "xxcaazz" and "xxbaaz" yields 3, since the "xx", "aa", and "az" substrings appear in the same place in both strings.

```
stringMatch("xxcaazz", "xxbaaz") → 3
stringMatch("abc", "abc") → 2
stringMatch("abc", "axc") → 0

public int stringMatch(String a, String b) {
  int count = 0;
  String shorty, longy;
  if (a.length() <= b.length()) {
     shorty = a;
     longy = b;
  } else {
     longy = a;
     shorty = b;
  }
  for (int i = 0; i < shorty.length()-1; i++) {
     if (shorty.substring(i, i+2).equals(longy.substring(i, i+2))) {
        count++;
     }
  }
  return count;
}
```

Given a string, return a version where all the "x" have been removed. Except an "x" at the very start or end should not be removed.

```
stringX("xxHxix") → "xHix"
stringX("abxxxcd") → "abcd"
stringX("xabxxxcdx") → "xabcdx"

public String stringX(String str) {
  String result = "";
  String newStr = str;
  if (str.indexOf("x") == -1 || str.equals("x") || str.equals("xx")) {
     return str;
  }
  while (newStr.indexOf("x") != -1) {
     result += newStr.substring(0, newStr.indexOf("x"));
     newStr = newStr.substring(newStr.indexOf("x") + 1);
  }
  result += newStr;
  if (str.indexOf("x") == 0) {
    result = "x" + result;
  }
  if (str.substring(str.length()-1).equals("x")) {
    result += "x";
  }
  return result;
}
```

Given a string, return a string made of the chars at indexes 0,1, 4,5, 8,9 ... so "kittens" yields "kien".

altPairs("kitten") → "kien"
altPairs("Chocolate") → "Chole"
altPairs("CodingHorror") → "Congrr"

```
public String altPairs(String str) {
  String result = "";
  if (str.length() < 3) {
    return str;
  }
  for (int i = 0; i < str.length()-1; i+=4) {
    result += str.substring(i, i+2);
  }
  if (str.length()%2 != 0 && str.length() > 3) {
    result += str.substring(str.length()-1);
  }
  return result;
}
```

Suppose the string "yak" is unlucky. Given a string, return a version where all the "yak" are removed, but the "a" can be any char. The "yak" strings will not overlap.

stringYak("yakpak") → "pak"
stringYak("pakyak") → "pak"
stringYak("yak123ya") → "123ya"

```
public String stringYak(String str) {
  String result = "";

  for (int i=0; i<str.length(); i++) {
    // Look for i starting a "yak" -- advance i in that case
    if (i+2<str.length() && str.charAt(i)=='y' && str.charAt(i+2)=='k') {
      i =  i + 2;
    } else { // Otherwise do the normal append
      result += str.charAt(i);
    }
  }

  return result;
}

/**
 * First try
public String stringYak(String str) {
  String result = "";
  String newStr = str;
  if (str.length() < 3) {
     return str;
  }
  for (int i = 0; i < newStr.length()-2; i++) {
     if (newStr.charAt(i) == 'y' && newStr.charAt(i+2) == 'k') {
        result += newStr.substring(0, newStr.indexOf("y"));
        newStr = newStr.substring(newStr.indexOf("k") + 1);
        i = 0;
     }
  }
  if (newStr.indexOf("k") < newStr.indexOf("y")) {
     result += newStr.substring(0, newStr.indexOf("k") + 1);
     newStr = newStr.substring(newStr.indexOf("y"));
  }
  while (newStr.indexOf("y") == newStr.indexOf("k")-2) {
     result += newStr.substring(0, newStr.indexOf("y"));
     newStr = newStr.substring(newStr.indexOf("k") + 1);
  }
  result += newStr;
  return result;
}
*/
```

Given an array of ints, return the number of times that two 6's are next to each other in the array. Also count instances where the second "6" is actually a 7.

array667([6, 6, 2]) → 1
array667([6, 6, 2, 6]) → 1
array667([6, 7, 2, 6]) → 1

```
public int array667(int[] nums) {
  int count = 0;
  for (int i = 0; i < nums.length-1; i++) {
     if (nums[i] == 6 && (nums[i+1] == 6 || nums[i+1] == 7)) {
        count++;
     }
  }
  return count;
}
```

Given an array of ints, we'll say that a triple is a value appearing 3 times in a row in the array. Return true if the array does not contain any triples.

noTriples([1, 1, 2, 2, 1]) → true
noTriples([1, 1, 2, 2, 2, 1]) → false
noTriples([1, 1, 1, 2, 2, 2, 1]) → false

```
public boolean noTriples(int[] nums) {
  for (int i = 0; i < nums.length-2; i++) {
     if (nums[i] == nums[i+1] && nums[i+1] == nums[i+2]) {
        return false;
     }
  }
  return true;
}
```

Given an array of ints, return true if it contains a 2, 7, 1 pattern -- a value, followed by the value plus 5, followed by the value minus 1. Additionally the 271 counts even if the "1" differs by 2 or less from the correct value.

has271([1, 2, 7, 1]) → true
has271([1, 2, 8, 1]) → false
has271([2, 7, 1]) → true

```
public boolean has271(int[] nums) {
  for (int i = 0; i < nums.length-2; i++) {
     if (nums[i] == nums[i+1]-5 && Math.abs(nums[i+2]-(nums[i]-1)) <= 2)
{
        return true;
     }
  }
  return false;
}
```

Logic-1

When squirrels get together for a party, they like to have cigars. A squirrel party is successful when the number of cigars is between 40 and 60, inclusive. Unless it is the weekend, in which case there is no upper bound on the number of cigars. Return true if the party with the given values is successful, or false otherwise.

cigarParty(30, false) → false
cigarParty(50, false) → true
cigarParty(70, true) → true

```
public boolean cigarParty(int cigars, boolean isWeekend) {
  if ((cigars > 39 && cigars < 61) && !isWeekend) {
    return true;
  } else if (cigars > 39 && isWeekend) {
    return true;
  } else {
    return false;
  }
}
```

You and your date are trying to get a table at a restaurant. The parameter "you" is the stylishness of your clothes, in the range 0..10, and "date" is the stylishness of your date's clothes. The result getting the table is encoded as an int value with 0=no, 1=maybe, 2=yes. If either of you is very stylish, 8 or more, then the result is 2 (yes). With the exception that if either of you has style of 2 or less, then the result is 0 (no). Otherwise the result is 1 (maybe).

dateFashion(5, 10) → 2
dateFashion(5, 2) → 0
dateFashion(5, 5) → 1

```
public int dateFashion(int you, int date) {
  if (you < 3 || date < 3) {
    return 0;
  } else if (you > 7 || date > 7){
    return 2;
  } else {
    return 1;
  }
}
```

The squirrels in Palo Alto spend most of the day playing. In particular, they play if the temperature is between 60 and 90 (inclusive). Unless it is summer, then the upper limit is 100 instead of 90. Given an int temperature and a boolean isSummer, return true if the squirrels play and false otherwise.

squirrelPlay(70, false) → true
squirrelPlay(95, false) → false
squirrelPlay(95, true) → true

```
public boolean squirrelPlay(int temp, boolean isSummer) {
  if (isSummer && (temp > 59 && temp < 101)) {
    return true;
  } else if (!isSummer && (temp > 59 && temp < 91)) {
    return true;
  } else {
    return false;
  }
}
```

You are driving a little too fast, and a police officer stops you. Write code to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday -- on that day, your speed can be 5 higher in all cases.

caughtSpeeding(60, false) → 0
caughtSpeeding(65, false) → 1
caughtSpeeding(65, true) → 0

```
public int caughtSpeeding(int speed, boolean isBirthday) {
  if (isBirthday && speed > 85) {
    return 2;
  } else if (isBirthday && speed > 65) {
    return 1;
  } else if (isBirthday && speed < 66) {
    return 0;
  } else if (!isBirthday && speed > 80) {
    return 2;
  } else if (!isBirthday && speed > 60) {
    return 1;
  } else if (!isBirthday && speed < 61) {
    return 0;
  } else {
    return 5;  // To complete statement
  }
}
```

Given 2 ints, a and b, return their sum. However, sums in the range 10..19 inclusive, are forbidden, so in that case just return 20.

sortaSum(3, 4) → 7
sortaSum(9, 4) → 20
sortaSum(10, 11) → 21

```
public int sortaSum(int a, int b) {
  if (a + b > 9 && a + b < 20) {
    return 20;
  } else {
    return a + b;
  }
}
```

Given a day of the week encoded as 0=Sun, 1=Mon, 2=Tue, ...6=Sat, and a boolean indicating if we are on vacation, return a string of the form "7:00" indicating when the alarm clock should ring. Weekdays, the alarm should be "7:00" and on the weekend it should be "10:00". Unless we are on vacation -- then on weekdays it should be "10:00" and weekends it should be "off".

alarmClock(1, false) → "7:00"
alarmClock(5, false) → "7:00"

alarmClock(0, false) → "10:00"

```
public String alarmClock(int day, boolean vacation) {
  if (vacation && (day != 0 && day != 6)) {
    return "10:00";
  } else if (vacation && (day == 0 || day == 6)) {
    return "off";
  } else if (!vacation && (day != 0 && day != 6)) {
    return "7:00";
  } else if (!vacation && (day == 0 || day == 6)) {
    return "10:00";
  } else {
    return "";
  }
}
```

The number 6 is a truly great number. Given two int values, a and b, return true if either one is 6. Or if their sum or difference is 6. Note: the function Math.abs(num) computes the absolute value of a number.

love6(6, 4) → true
love6(4, 5) → false
love6(1, 5) → true

```
public boolean love6(int a, int b) {
  if (a == 6 || b == 6 || Math.abs(a-b) == 6 || a+b == 6) {
    return true;
  } else {
    return false;
  }
}
```

Given a number n, return true if n is in the range 1..10, inclusive. Unless "outsideMode" is true, in which case return true if the number is less or equal to 1, or greater or equal to 10.

in1To10(5, false) → true
in1To10(11, false) → false
in1To10(11, true) → true

```
public boolean in1To10(int n, boolean outsideMode) {
  if (outsideMode && (n <= 1 || n >= 10)) {
    return true;
  } else if (!outsideMode && (n >= 1 && n <= 10)) {
    return true;
  } else {
    return false;
  }
}
```

We'll say a number is special if it is a multiple of 11 or if it is one more than a multiple of 11. Return true if the given non-negative number is special. Use the % "mod" operator -- see Introduction to Mod

specialEleven(22) → true
specialEleven(23) → true
specialEleven(24) → false

```
public boolean specialEleven(int n) {
  if ((n % 11 == 0) || (n % 11 == 1)) {
    return true;
  } else {
    return false;
  }
}
```

Return true if the given non-negative number is 1 or 2 more than a multiple of 20. See also: Introduction to Mod

more20(20) → false
more20(21) → true
more20(22) → true

```
public boolean more20(int n) {
  if (n % 20 == 1 || n % 20 == 2) {
    return true;
  } else {
    return false;
  }
}
```

Return true if the given non-negative number is a multiple of 3 or 5, but not both. Use the % "mod" operator -- see Introduction to Mod

old35(3) → true
old35(10) → true
old35(15) → false

```
public boolean old35(int n) {
  if ((n % 3 == 0 || n % 5 == 0) && n % 15 != 0) {
    return true;
  } else {
    return false;
  }
}
```

Return true if the given non-negative number is 1 or 2 **less** than a multiple of 20. So for example 38 and 39 return true, but 40 returns false. See also: Introduction to Mod

less20(18) → true
less20(19) → true
less20(20) → false

```
public boolean less20(int n) {
  if (n % 20 == 19 || n % 20 == 18) {
    return true;
  } else {
    return false;
  }
}
```

Given a non-negative number "num", return true if num is within 2 of a multiple of 10. Note: (a % b) is the remainder of dividing a by b, so (7 % 5) is 2. See also: Introduction to Mod

nearTen(12) → true

nearTen(17) → false
nearTen(19) → true

```
public boolean nearTen(int num) {
  if (num % 10 == 8 || num % 10 == 9 || num % 10 == 0 || num % 10
== 1 || num % 10 == 2) {
    return true;
  } else {
    return false;
  }
}
```

Given 2 ints, a and b, return their sum. However, "teen" values in the range 13..19 inclusive, are extra lucky. So if either value is a teen, just return 19.

teenSum(3, 4) → 7
teenSum(10, 13) → 19
teenSum(13, 2) → 19

```
public int teenSum(int a, int b) {
  if ((a > 12 && a < 20) || (b > 12 && b < 20)) {
    return 19;
  } else {
    return a + b;
  }
}
```

Your cell phone rings. Return true if you should answer it. Normally you answer, except in the morning you only answer if it is your mom calling. In all cases, if you are asleep, you do not answer.

answerCell(false, false, false) → true
answerCell(false, false, true) → false
answerCell(true, false, false) → false

```
public boolean answerCell(boolean isMorning, boolean isMom, boolean
isAsleep) {
  if (isAsleep) {
    return false;
  } else if (isMorning && isMom) {
    return true;
  } else if (isMorning && !isMom) {
    return false;
  } else {
    return true;
  }
}
```

We are having a party with amounts of tea and candy. Return the int outcome of the party encoded as 0=bad, 1=good, or 2=great. A party is good (1) if both tea and candy are at least 5. However, if either tea or candy is at least double the amount of the other one, the party is great (2). However, in all cases, if either tea or candy is less than 5, the party is always bad (0).

teaParty(6, 8) → 1
teaParty(3, 8) → 0
teaParty(20, 6) → 2

```
public int teaParty(int tea, int candy) {
  if (tea < 5 || candy < 5) {
    return 0;
  } else if (tea >= 2*candy || candy >= 2*tea) {
    return 2;
  } else {
    return 1;
  }
}
```

Given a string str, if the string starts with "f" return "Fizz". If the string ends with "b" return "Buzz". If both the "f" and "b" conditions are true, return "FizzBuzz". In all other cases, return the string unchanged. (See also: FizzBuzz Code)

fizzString("fig") → "Fizz"
fizzString("dib") → "Buzz"
fizzString("fib") → "FizzBuzz"

```
public String fizzString(String str) {
  if (str.substring(0, 1).equals("f") &&
str.substring(str.length()-1).equals("b")) {
    return "FizzBuzz";
  } else if (str.substring(0, 1).equals("f")) {
    return "Fizz";
  } else if (str.substring(str.length()-1).equals("b")) {
    return "Buzz";
  } else {
    return str;
  }
}
```

Given an int n, return the string form of the number followed by "!". So the int 6 yields "6!". Except if the number is divisible by 3 use "Fizz" instead of the number, and if the number is divisible by 5 use "Buzz", and if divisible by both 3 and 5, use "FizzBuzz". Note: the % "mod" operator computes the remainder after division, so 23 % 10 yields 3. What will the remainder be when one number divides evenly into another? (See also: FizzBuzz Code and Introduction to Mod)

fizzString2(1) → "1!"
fizzString2(2) → "2!"
fizzString2(3) → "Fizz!"

```
public String fizzString2(int n) {
  if (n%3 == 0 && n%5 == 0) {
    return "FizzBuzz!";
  } else if (n%3 == 0) {
    return "Fizz!";
  } else if (n%5 == 0) {
    return "Buzz!";
  } else {
    return n + "!";
  }
}
```

Given three ints, a b c, return true if it is possible to add two of the ints to get the third.

twoAsOne(1, 2, 3) → true
twoAsOne(3, 1, 2) → true
twoAsOne(3, 2, 2) → false

```
public boolean twoAsOne(int a, int b, int c) {
  if (a+b == c || b+c == a || a+c == b) {
    return true;
  } else {
    return false;
  }
}
```

Given three ints, a b c, return true if b is greater than a, and c is greater than b. However, with the exception that if "bOk" is true, b does not need to be greater than a.

inOrder(1, 2, 4, false) → true
inOrder(1, 2, 1, false) → false
inOrder(1, 1, 2, true) → true

```
public boolean inOrder(int a, int b, int c, boolean bOk) {
  if (b > a && c > b && !bOk) {
    return true;
  } else if (c > b && bOk) {
    return true;
  } else {
    return false;
  }
}
```

Given three ints, a b c, return true if they are in strict increasing order, such as 2 5 11, or 5 6 7, but not 6 5 7 or 5 5 7. However, with the exception that if "equalOk" is true, equality is allowed, such as 5 5 7 or 5 5 5.

inOrderEqual(2, 5, 11, false) → true
inOrderEqual(5, 7, 6, false) → false
inOrderEqual(5, 5, 7, true) → true

```
public boolean inOrderEqual(int a, int b, int c, boolean equalOk) {
  if (!equalOk && a<b && b<c) {
    return true;
  } else if (equalOk && a<=b && b<=c) {
    return true;
  } else {
    return false;
  }
}
```

Given three ints, a b c, return true if two or more of them have the same rightmost digit. The ints are non-negative. Note: the % "mod" operator computes the remainder, e.g. 17 % 10 is 7.

lastDigit(23, 19, 13) → true
lastDigit(23, 19, 12) → false
lastDigit(23, 19, 3) → true

```
public boolean lastDigit(int a, int b, int c) {
  if (a%10 == b%10 || b%10 == c%10 || a%10 == c%10) {
    return true;
  } else {
    return false;
  }
}
```

Given three ints, a b c, return true if one of them is 10 or more less than one of the others.

lessBy10(1, 7, 11) → true
lessBy10(1, 7, 10) → false
lessBy10(11, 1, 7) → true

```
public boolean lessBy10(int a, int b, int c) {
  if (Math.abs(a-b)>9 || Math.abs(b-c)>9 || Math.abs(a-c)>9) {
    return true;
  } else {
    return false;
  }
}
```

Return the sum of two 6-sided dice rolls, each in the range 1..6. However, if noDoubles is true, if the two dice show the same value, increment one die to the next value, wrapping around to 1 if its value was 6.

withoutDoubles(2, 3, true) → 5
withoutDoubles(3, 3, true) → 7
withoutDoubles(3, 3, false) → 6

```
public int withoutDoubles(int die1, int die2, boolean noDoubles) {
  if (noDoubles && die1==die2 && die1 == 6) {
    return 7;
  } else if (noDoubles && die1==die2) {
    return die1 + die2 + 1;
  } else {
    return die1 + die2;
  }
}
```

Given two int values, return whichever value is larger. However if the two values have the same remainder when divided by 5, then the return the smaller value. However, in all cases, if the two values are the same, return 0. Note: the % "mod" operator computes the remainder, e.g. 7 % 5 is 2.

maxMod5(2, 3) → 3
maxMod5(6, 2) → 6
maxMod5(3, 2) → 3

```
public int maxMod5(int a, int b) {
  if (a == b) {
    return 0;
  } else if (a%5 == b%5) {
    return Math.min(a, b);
  } else {
    return Math.max(a, b);
  }
}
```

You have a red lottery ticket showing ints a, b, and c, each of which is 0, 1, or 2. If they are all the value 2, the result is 10. Otherwise if they are all the same, the result is 5. Otherwise so long as both b and c are different from a, the result is 1. Otherwise the result is 0.

redTicket(2, 2, 2) → 10
redTicket(2, 2, 1) → 0
redTicket(0, 0, 0) → 5

```
public int redTicket(int a, int b, int c) {
  if (a == b && b == c && a == 2) {
    return 10;
  } else if (a == b && b == c) {
    return 5;
  } else if (a!=b && a!=c) {
    return 1;
  } else {
    return 0;
  }
}
```

You have a green lottery ticket, with ints a, b, and c on it. If the numbers are all different from each other, the result is 0. If all of the numbers are the same, the result is 20. If two of the numbers are the same, the result is 10.

greenTicket(1, 2, 3) → 0
greenTicket(2, 2, 2) → 20
greenTicket(1, 1, 2) → 10

```
public int greenTicket(int a, int b, int c) {
  if (a == b && b == c) {
    return 20;
  } else if (a == b || b == c || a == c) {
    return 10;
  } else {
    return 0;
  }
}
```

You have a blue lottery ticket, with ints a, b, and c on it. This makes three pairs, which we'll call ab, bc, and ac. Consider the sum of the numbers in each pair. If any pair sums to exactly 10, the result is 10. Otherwise if the ab sum is exactly 10 more than either bc or ac sums, the result is 5. Otherwise the result is 0.

blueTicket(9, 1, 0) → 10
blueTicket(9, 2, 0) → 0
blueTicket(6, 1, 4) → 10

```
public int blueTicket(int a, int b, int c) {
  if (a+b==10 || b+c==10 || a+c==10) {
    return 10;
  } else if (a+b == b+c+10 || a+b == a+c+10) {
    return 5;
  } else {
    return 0;
  }
}
```

Given two ints, each in the range 10..99, return true if there is a digit that appears in both numbers, such as the 2 in 12 and 23. (Note: division, e.g. n/10, gives the left digit while the % "mod" n%10 gives the right digit.)

shareDigit(12, 23) → true
shareDigit(12, 43) → false
shareDigit(12, 44) → false

```
public boolean shareDigit(int a, int b) {
  if (a/10 == b/10 || a/10 == b%10 || a%10 == b/10 || a%10 == b%10) {
    return true;
  } else {
    return false;
  }
}
```

Given 2 non-negative ints, a and b, return their sum, so long as the sum has the same number of digits as a. If the sum has more digits than a, just return a without b. (Note: one way to compute the number of digits of a non-negative int n is to convert it to a string with String.valueOf(n) and then check the length of the string.)

sumLimit(2, 3) → 5
sumLimit(8, 3) → 8
sumLimit(8, 1) → 9

```
public int sumLimit(int a, int b) {
  if (String.valueOf(a+b).length() == String.valueOf(a).length()) {
    return a + b;
  } else {
    return a;
  }
}
```

Logic-2

We want to make a row of bricks that is **goal** inches long. We have a number of small bricks (1 inch each) and big bricks (5 inches each). Return true if it is possible to make the goal by choosing from the given bricks. This is a little harder than it looks and can be done without any loops. See also: Introduction to MakeBricks

makeBricks(3, 1, 8) → true
makeBricks(3, 1, 9) → false
makeBricks(3, 2, 10) → true

```
public boolean makeBricks(int small, int big, int goal) {
  /**
   * This can all be condensed!
  if (small + 5*big < goal)  return false;  // not enough
  if (goal % 5 > small)  return false;  // not enough small
  return true;
  */
  return ((small + 5*big >= goal) && (goal % 5 <= small));
}

/**
 * First attempt
 * Not sure why it failed exactly
public boolean makeBricks(int small, int big, int goal) {
  if (small + 5*big < goal) {
    return false;
  } else {
    if (goal <= 5*big) {
      if (goal%5 == 0) {
        return true;
      }
      goal -= small;
      return goal % 5 == 0;
    }
    return true;
  }
}
*/
```

Given 3 int values, a b c, return their sum. However, if one of the values is the same as another of the values, it does not count towards the sum.

```
loneSum(1, 2, 3) → 6
loneSum(3, 2, 3) → 2
loneSum(3, 3, 3) → 0
```

```
public int loneSum(int a, int b, int c) {
  if (a == b && b == c) {
    return 0;
  } else if (a == b) {
    return c;
  } else if (b == c) {
    return a;
  } else if (a == c) {
    return b;
  } else {
    return a + b + c;
  }
}
```

Given 3 int values, a b c, return their sum. However, if one of the values is 13 then it does not count towards the sum and values to its right do not count. So for example, if b is 13, then both b and c do not count.

```
luckySum(1, 2, 3) → 6
luckySum(1, 2, 13) → 3
luckySum(1, 13, 3) → 1
```

```
public int luckySum(int a, int b, int c) {
  if (a == 13) {
    return 0;
  } else if (b == 13) {
    return a;
  } else if (c == 13) {
    return a + b;
  } else {
    return a + b + c;
  }
}
```

Given 3 int values, a b c, return their sum. However, if any of the values is a teen -- in the range 13..19 inclusive -- then that value counts as 0, except 15 and 16 do not count as a teens. Write a separate helper "public int fixTeen(int n) {"that takes in an int value and returns that value fixed for the teen rule. In this way, you avoid repeating the teen code 3 times (i.e. "decomposition"). Define the helper below and at the same indent level as the main noTeenSum().

noTeenSum(1, 2, 3) → 6
noTeenSum(2, 13, 1) → 3
noTeenSum(2, 1, 14) → 3

```
public int noTeenSum(int a, int b, int c) {
  return fixTeen(a) + fixTeen(b) + fixTeen(c);
}

public int fixTeen(int n) {
  if (n > 12 && n < 20 && n != 15 && n != 16) {
    return 0;
  } else {
    return n;
  }
}
```

For this problem, we'll round an int value up to the next multiple of 10 if its rightmost digit is 5 or more, so 15 rounds up to 20. Alternately, round down to the previous multiple of 10 if its rightmost digit is less than 5, so 12 rounds down to 10. Given 3 ints, a b c, return the sum of their rounded values. To avoid code repetition, write a separate helper "public int round10(int num) {" and call it 3 times. Write the helper entirely below and at the same indent level as roundSum().

roundSum(16, 17, 18) → 60
roundSum(12, 13, 14) → 30

```
roundSum(6, 4, 4) → 10

public int roundSum(int a, int b, int c) {
  return round10(a) + round10(b) + round10(c);
}

public int round10(int n) {
  if (n % 10 > 4) {
    return n + (10 - (n % 10));
  } else if (n % 10 < 5) {
    return n - (n % 10);
  } else if (n % 10 == 0) {
    return n;
  } else {
    return 525600;
  }
}
```

Given three ints, a b c, return true if one of b or c is "close" (differing from a by at most 1), while the other is "far", differing from both other values by 2 or more. Note: Math.abs(num) computes the absolute value of a number.

```
closeFar(1, 2, 10) → true
closeFar(1, 2, 3) → false
closeFar(4, 1, 3) → true

public boolean closeFar(int a, int b, int c) {
  if (((b == a+1 || b == a-1 || b == a) && ((c > a+1 || c < a-1) && (c >
b+1 || c < b-1)) || ((c == a+1 || c == a-1 || c == a) && ((b > a+1 || b <
a-1) && (b > c+1 || b < c-1))))) {
    return true;
  } else {
    return false;
  }
}
```

Given 2 int values greater than 0, return whichever value is nearest to 21 without going over. Return 0 if they both go over.

blackjack(19, 21) → 21
blackjack(21, 19) → 21
blackjack(19, 22) → 19

```
public int blackjack(int a, int b) {
  if (a > 21 && b < 22) {
    return b;
  } else if (b > 21 && a < 22) {
    return a;
  } else if (a > 21 && b > 21) {
    return 0;
  } else if (a >= b) {
    return  a;
  } else if (b > a) {
    return b;
  } else {
    return 525600;
  }
}
```

Given three ints, a b c, one of them is small, one is medium and one is large. Return true if the three values are evenly spaced, so the difference between small and medium is the same as the difference between medium and large.

evenlySpaced(2, 4, 6) → true
evenlySpaced(4, 6, 2) → true
evenlySpaced(4, 6, 3) → false

```
public boolean evenlySpaced(int a, int b, int c) {
  int la, med, sm;
  // I think there are six possibilities, assuming no two numbers are the
same
  // Note: I know it must be tempting to copy and paste a lot, but that may
lead to errors. Besides, this was good logic practice.
  if (a > b && b > c) {
    la = a;
    med = b;
    sm = c;
  } else if (a > c && c > b) {
    la = a;
    med = c;
    sm = b;
  } else if (b > a && a > c) {
    la = b;
    med = a;
    sm = c;
  } else if (b > c && c > a) {
    la = b;
    med = c;
    sm = a;
  } else if (c > a && a > b) {
    la = c;
    med = a;
    sm = b;
  } else {  // (c > b && b > a)
    la = c;
    med = b;
    sm = a;
  }
  return (la-med) == (med-sm);
}
```

We want make a package of **goal** kilos of chocolate. We have small bars (1 kilo each) and big bars (5 kilos each). Return the number of small bars to use, assuming we always use big bars before small bars. Return -1 if it can't be done.

makeChocolate(4, 1, 9) → 4
makeChocolate(4, 1, 10) → -1
makeChocolate(4, 1, 7) → 2

```
public int makeChocolate(int small, int big, int goal) {
  if (small + 5*big < goal)  return -1;  // not enough
  if (goal % 5 > small)  return -1;  // not enough small
  // Two cases - goal%5 and goal-5*big
  if (5*big < goal) {  // Note: < or <= works here (since %5 catches the
<=)
```

```
    return goal-(5*big);
  }
  return goal%5;
}
```

String-1

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

```
public String helloName(String name) {
  return "Hello " + name + "!";
}
```

Given two strings, a and b, return the result of putting them together in the order abba, e.g. "Hi" and "Bye" returns "HiByeByeHi".

makeAbba("Hi", "Bye") → "HiByeByeHi"
makeAbba("Yo", "Alice") → "YoAliceAliceYo"
makeAbba("What", "Up") → "WhatUpUpWhat"

```
public String makeAbba(String a, String b) {
  return a + b + b + a;
}
```

The web is built with HTML strings like "<i>Yay</i>" which draws Yay as italic text. In this example, the "i" tag makes <i> and </i> which surround the word "Yay". Given tag and word strings, create the HTML string with tags around the word, e.g. "<i>Yay</i>".

makeTags("i", "Yay") → "<i>Yay</i>"
makeTags("i", "Hello") → "<i>Hello</i>"
makeTags("cite", "Yay") → "<cite>Yay</cite>"

```
public String makeTags(String tag, String word) {
  return "<" + tag + ">" + word + "</" + tag + ">";
}
```

Given an "out" string length 4, such as "<<>>", and a word, return a new string where the word is in the middle of the out string, e.g. "<<word>>".

Note: use str.substring(i, j) to extract the String starting at index i and going up to but not including index j.

makeOutWord("<<>>", "Yay") → "<<Yay>>"
makeOutWord("<<>>", "WooHoo") → "<<WooHoo>>"
makeOutWord("[[]]", "word") → "[[word]]"

```
public String makeOutWord(String out, String word) {
  return out.substring(0, 2) + word + out.substring(2, 4);
}
```

Given a string, return a new string made of 3 copies of the last 2 chars of the original string. The string length will be at least 2.

extraEnd("Hello") → "lololo"
extraEnd("ab") → "ababab"
extraEnd("Hi") → "HiHiHi"

```
public String extraEnd(String str) {
  return str.substring(str.length()-2) + str.substring(str.length()-2) +
str.substring(str.length()-2);
}
```

Given a string, return the string made of its first two chars, so the String "Hello" yields "He". If the string is shorter than length 2, return whatever there is, so "X" yields "X", and the empty string "" yields the empty string "". Note that str.length() returns the length of a string.

firstTwo("Hello") → "He"
firstTwo("abcdefg") → "ab"
firstTwo("ab") → "ab"

```
public String firstTwo(String str) {
  if (str.length() < 2) {
  return str;
  } else {
  return str.substring(0, 2);
  }
}
```

Given a string of even length, return the first half. So the string "WooHoo" yields "Woo".

firstHalf("WooHoo") → "Woo"
firstHalf("HelloThere") → "Hello"
firstHalf("abcdef") → "abc"

```
public String firstHalf(String str) {
  return str.substring(0, str.length()/2);
}
```

Given a string, return a version without the first and last char, so "Hello" yields "ell". The string length will be at least 2.

withoutEnd("Hello") → "ell"
withoutEnd("java") → "av"
withoutEnd("coding") → "odin"

```
public String withoutEnd(String str) {
  return str.substring(1, str.length()-1);
}
```

Given 2 strings, a and b, return a string of the form short+long+short, with the shorter string on the outside and the longer string on the inside. The strings will not be the same length, but they may be empty (length 0).

comboString("Hello", "hi") → "hiHellohi"
comboString("hi", "Hello") → "hiHellohi"
comboString("aaa", "b") → "baaab"

```
public String comboString(String a, String b) {
  if (a.length() < b.length()) {
    return a + b + a;
  } else {
    return b + a + b;
  }
}
```

Given 2 strings, return their concatenation, except omit the first char of each. The strings will be at least length 1.

nonStart("Hello", "There") → "ellohere"
nonStart("java", "code") → "avaode"
nonStart("shotl", "java") → "hotlava"

```
public String nonStart(String a, String b) {
  return a.substring(1) + b.substring(1);
}
```

Given a string, return a "rotated left 2" version where the first 2 chars are moved to the end. The string length will be at least 2.

left2("Hello") → "lloHe"
left2("java") → "vaja"
left2("Hi") → "Hi"

```
public String left2(String str) {
  return str.substring(2) + str.substring(0, 2);
}
```

Given a string, return a "rotated right 2" version where the last 2 chars are moved to the start. The string length will be at least 2.

right2("Hello") → "loHel"
right2("java") → "vaja"
right2("Hi") → "Hi"

```
public String right2(String str) {
  return str.substring(str.length()-2) + str.substring(0, str.length()-2);
}
```

Given a string, return a string length 1 from its front, unless **front** is false, in which case return a string length 1 from its back. The string will be non-empty.

theEnd("Hello", true) → "H"
theEnd("Hello", false) → "o"

theEnd("oh", true) → "o"

```
public String theEnd(String str, boolean front) {
  if (front) {
    return str.substring(0, 1);
  } else {
    return str.substring(str.length()-1);
  }
}
```

Given a string, return a version without both the first and last char of the string. The string may be any length, including 0.

withouEnd2("Hello") → "ell"
withouEnd2("abc") → "b"
withouEnd2("ab") → ""

```
public String withouEnd2(String str) {
  if (str.length() < 3) {
    return "";
  } else {
    return str.substring(1, str.length()-1);
  }
}
```

Given a string of even length, return a string made of the middle two chars, so the string "string" yields "ri". The string length will be at least 2.

middleTwo("string") → "ri"
middleTwo("code") → "od"
middleTwo("Practice") → "ct"

```
public String middleTwo(String str) {
  return str.substring((str.length()/2)-1, str.length()/2+1);
}
```

Given a string, return true if it ends in "ly".

endsLy("oddly") → true

endsLy("y") → false
endsLy("oddy") → false

```
public boolean endsLy(String str) {
  if (str.length() < 2) {
    return false;
  } else if (str.substring(str.length()-2).equals("ly")) {
    return true;
  } else {
    return false;
  }
}
```

Given a string and an int n, return a string made of the first and last n chars from the string. The string length will be at least n.

nTwice("Hello", 2) → "Helo"
nTwice("Chocolate", 3) → "Choate"
nTwice("Chocolate", 1) → "Ce"

```
public String nTwice(String str, int n) {
  return str.substring(0, n) + str.substring(str.length()-n);
}
```

Given a string and an index, return a string length 2 starting at the given index. If the index is too big or too small to define a string length 2, use the first 2 chars. The string length will be at least 2.

twoChar("java", 0) → "ja"
twoChar("java", 2) → "va"
twoChar("java", 3) → "ja"

```
public String twoChar(String str, int index) {
  if (index < 0 || (index+2) > str.length()) {
    return str.substring(0, 2);
  } else {
    return str.substring(index, (index+2));
  }
}
```

Given a string of odd length, return the string length 3 from its middle, so "Candy" yields "and". The string length will be at least 3.

```
middleThree("Candy") → "and"
middleThree("and") → "and"
middleThree("solving") → "lvi"

public String middleThree(String str) {
  return str.substring((str.length()/2)-1, (str.length()/2)+2);
}
```

Given a string, return true if "bad" appears starting at index 0 or 1 in the string, such as with "badxxx" or "xbadxx" but not "xxbadxx". The string may be any length, including 0. Note: use .equals() to compare 2 strings.

```
hasBad("badxx") → true
hasBad("xbadxx") → true
hasBad("xxbadxx") → false

public boolean hasBad(String str) {
  if (str.length() < 3) {
    return false;
  } else if (str.substring(0, 3).equals("bad")){
    return true;
  } else if (str.length() < 4) {
    return false;
  } else if (str.substring(1, 4).equals("bad")) {
    return true;
  } else {
    return false;
  }
}
```

Given a string, return a string length 2 made of its first 2 chars. If the string length is less than 2, use '@' for the missing chars.

```
atFirst("hello") → "he"
atFirst("hi") → "hi"
```

atFirst("h") → "h@"

```
public String atFirst(String str) {
  if (str.length() == 0) {
    return "@@";
  } else if (str.length() == 1) {
    return str.substring(0, 1) + "@";
  } else {
    return str.substring(0, 2);
  }
}
```

Given 2 strings, a and b, return a new string made of the first char of a and the last char of b, so "yo" and "java" yields "ya". If either string is length 0, use '@' for its missing char.

lastChars("last", "chars") → "ls"
lastChars("yo", "java") → "ya"
lastChars("hi", "") → "h@"

```
public String lastChars(String a, String b) {
  if (a.length() == 0 && b.length() == 0) {
    return "@@";
  } else if (a.length() == 0) {
    return "@" + b.substring(b.length()-1);
  } else if (b.length() == 0) {
    return a.substring(0, 1) + "@";
  } else {
    return a.substring(0, 1) + b.substring(b.length()-1);
  }
}
```

Given two strings, append them together (known as "concatenation") and return the result. However, if the concatenation creates a double-char, then omit one of the chars, so "abc" and "cat" yields "abcat".

conCat("abc", "cat") → "abcat"
conCat("dog", "cat") → "dogcat"
conCat("abc", "") → "abc"

```
public String conCat(String a, String b) {
  if (a.length() == 0 || b.length() == 0) {
    return a + b;
  } else if (a.substring(a.length()-1).equals(b.substring(0, 1))) {
    return a.substring(0, a.length()-1) + b;
  } else {
    return a + b;
  }
}
```

Given a string of any length, return a new string where the last 2 chars, if present, are swapped, so "coding" yields "codign".

```
lastTwo("coding") → "codign"
lastTwo("cat") → "cta"
lastTwo("ab") → "ba"
```

```
public String lastTwo(String str) {
  if (str.length() < 2) {
    return str;
  } else {
    return str.substring(0, str.length()-2) + str.substring(str.length()-1) +
str.substring(str.length()-2, str.length()-1);
  }
}
```

Given a string, if the string begins with "red" or "blue" return that color string, otherwise return the empty string.

```
seeColor("redxx") → "red"
seeColor("xxred") → ""
seeColor("blueTimes") → "blue"
```

```
public String seeColor(String str) {
  if (str.length() < 3) {
    return "";
  } else if (str.substring(0, 3).equals("red")) {
    return "red";
  } else if (str.length() < 4) {
    return "";
  } else if (str.substring(0, 4).equals("blue")) {
    return "blue";
  } else {
    return "";
  }
}
```

Given a string, return true if the first 2 chars in the string also appear at the end of the string, such as with "edited".

```
frontAgain("edited") → true
frontAgain("edit") → false
frontAgain("ed") → true

public boolean frontAgain(String str) {
  if (str.length() < 2) {
    return false;
  } else if (str.substring(0, 2).equals(str.substring(str.length()-2))) {
    return true;
  } else {
    return false;
  }
}
```

Given two strings, append them together (known as "concatenation") and return the result. However, if the strings are different lengths, omit chars from the longer string so it is the same length as the shorter string. So "Hello" and "Hi" yield "loHi". The strings may be any length.

```
minCat("Hello", "Hi") → "loHi"
minCat("Hello", "java") → "ellojava"
minCat("java", "Hello") → "javaello"

public String minCat(String a, String b) {
  if (a.length() > b.length()) {
    String newA = a.substring(a.length()-b.length());
    return newA + b;
  } else if (a.length() < b.length()) {
    String newB = b.substring(b.length()-a.length());
    return a + newB;
  } else {
    return a + b;
  }
}
```

Given a string, return a new string made of 3 copies of the first 2 chars of the original string. The string may be any length. If there are fewer than 2 chars, use whatever is there.

extraFront("Hello") → "HeHeHe"
extraFront("ab") → "ababab"
extraFront("H") → "HHH"

```
public String extraFront(String str) {
  if (str.length() < 2) {
    return str + str + str;
  } else {
    return str.substring(0,2) + str.substring(0,2) + str.substring(0,2);
  }
}
```

Given a string, if a length 2 substring appears at both its beginning and end, return a string without the substring at the beginning, so "HelloHe" yields "lloHe". The substring may overlap with itself, so "Hi" yields "". Otherwise, return the original string unchanged.

without2("HelloHe") → "lloHe"
without2("HelloHi") → "HelloHi"
without2("Hi") → ""

```
public String without2(String str) {
  if (str.length() < 2) {
    return str;
  } else if (str.length() == 2) {
    return "";
  } else {
    if (str.substring(0,2).equals(str.substring(str.length()-2))) {
      return str.substring(2);
    } else {
      return str;
    }
  }
}
```

Given a string, return a version without the first 2 chars. Except keep the first char if it is 'a' and keep the second char if it is 'b'. The string may be any length. Harder than it looks.

deFront("Hello") → "llo"
deFront("java") → "va"
deFront("away") → "aay"

```
public String deFront(String str) {
  String newStr = str;
  if (str.length() < 2) {
    return str;
  } else {
     if (str.substring(0,1).equals("a")) {
        if (str.substring(1,2).equals("b")) {
           return str;
        } else {
           return "a" + str.substring(2);
        }
     } else if (str.substring(1,2).equals("b")) {
        return str.substring(1);
     } else {
        return str.substring(2);
     }
  }
}
```

Given a string and a second "word" string, we'll say that the word matches the string if it appears at the front of the string, except its first char does not need to match exactly. On a match, return the front of the string, or otherwise return the empty string. So, so with the string "hippo" the word "hi" returns "hi" and "xip" returns "hip". The word will be at least length 1.

startWord("hippo", "hi") → "hi"
startWord("hippo", "xip") → "hip"
startWord("hippo", "i") → "h"

```
public String startWord(String str, String word) {
  if (str.length() == 0) {
     return "";
  } else if (word.length() == 1) {
     return str.substring(0,1);
  } else if (str.length() == 1) {
     return "";
  } else {
     if (word.substring(1).equals(str.substring(1,word.length()))) {
        return str.substring(0,word.length());
     } else {
        return "";
     }
  }
}
```

Given a string, if the first or last chars are 'x', return the string without those 'x' chars, and otherwise return the string unchanged.

withoutX("xHix") → "Hi"
withoutX("xHi") → "Hi"
withoutX("Hxix") → "Hxi"

```
public String withoutX(String str) {
  if (str.length() == 0) {
     return str;
  } else if (str.equals("x")) {
     return "";
  } else if (str.substring(0,1).equals("x")) {
     if (str.substring(str.length()-1).equals("x")) {
        return str.substring(1,str.length()-1);
     } else {
      return str.substring(1);
    }
  } else if (str.substring(str.length()-1).equals("x")) {
    return str.substring(0,str.length()-1);
  } else {
    return str;
  }
}
```

Given a string, if one or both of the first 2 chars is 'x', return the string without those 'x' chars, and otherwise return the string unchanged. This is a little harder than it looks.

withoutX2("xHi") → "Hi"
withoutX2("Hxi") → "Hi"
withoutX2("Hi") → "Hi"

```
public String withoutX2(String str) {
  if (str.length() == 0) {
    return str;
  } else if (str.equals("x") || str.equals("xx")) {
    return "";
  } else if (str.substring(0,1).equals("x")) {
    if (str.substring(1,2).equals("x")) {
      return str.substring(2);
    } else {
      return str.substring(1);
    }
  } else if (str.substring(1,2).equals("x")) {
    return str.substring(0,1) + str.substring(2);
  } else {
    return str;
  }
}
```

String-2

Given a string, return a string where for every char in the original, there are two chars.

doubleChar("The") → "TThhee"
doubleChar("AAbb") → "AAAAbbbb"
doubleChar("Hi-There") → "HHii--TThheerree"

```
public String doubleChar(String str) {
  String result = "";
  for (int i = 0; i < str.length(); i++) {
     result += str.substring(i,i+1);
     result += str.substring(i,i+1);
  }
  return result;
}
```

Return the number of times that the string "hi" appears anywhere in the given string.

countHi("abc hi ho") → 1
countHi("ABChi hi") → 2
countHi("hihi") → 2

```
public int countHi(String str) {
  String hi = "hi";
  int count = 0;
  for (int i = 0; i<str.length()-1; i++) {
    if (str.substring(i, i+2).equals(hi)) {
      count++;
    }
  }
  return count;
}
```

Return true if the string "cat" and "dog" appear the same number of times in the given string.

```
catDog("catdog") → true
catDog("catcat") → false
catDog("1cat1cadodog") → true

public boolean catDog(String str) {
  int dogCount = 0;
  int catCount = 0;
  for (int i = 0; i<str.length()-2; i++) {
    if (str.substring(i, i+3).equals("cat")) {
      catCount++;
    } else if (str.substring(i, i+3).equals("dog")) {
      dogCount++;
    }
  }
  if (catCount == dogCount) {
    return true;
  } else {
    return false;
  }
}
```

Return the number of times that the string "code" appears anywhere in the given string, except we'll accept any letter for the 'd', so "cope" and "cooe" count.

```
countCode("aaacodebbb") → 1
countCode("codexxcode") → 2
countCode("cozexxcope") → 2

public int countCode(String str) {
  int codeCount = 0;
  int i = 0;
  int k;
  while (i<str.length()-3) {
    k = i+3;
    if (str.substring(i, i+2).equals("co") && str.substring(k,
k+1).equals("e")){
    codeCount++;
    }
  i++;
  }
  return codeCount;
}
```

Given two strings, return true if either of the strings appears at the very end of the other string, ignoring upper/lower case differences (in other words, the computation should not be "case sensitive"). Note: str.toLowerCase() returns the lowercase version of a string.

endOther("Hiabc", "abc") → true
endOther("AbC", "HiaBc") → true
endOther("abc", "abXabc") → true

```
public boolean endOther(String a, String b) {
  int aLength = a.length();
  int bLength = b.length();
  if (aLength > bLength) {
     if (a.substring(Math.abs(aLength-bLength)).equalsIgnoreCase(b)) {
        return true;
     } else {
        return false;
     }
  } else if (aLength < bLength) {
     if (b.substring(Math.abs(aLength-bLength)).equalsIgnoreCase(a)) {
        return true;
     } else {
        return false;
     }
  } else if (a.equals(b)) {
     return true;
  } else {
     return false;
  }
}
```

Return true if the given string contains an appearance of "xyz" where the xyz is not directly preceeded by a period (.). So "xxyz" counts but "x.xyz" does not.

xyzThere("abcxyz") → true
xyzThere("abc.xyz") → false
xyzThere("xyz.abc") → true

```
public boolean xyzThere(String str) {
  for (int i = 0; i < str.length()-2; i++) {
     if (str.substring(i,i+3).equals("xyz")) {
        if (i == 0 || str.charAt(i-1) != '.') {
           return true;
        }
     }
  }
  return false;
}
```

Return true if the given string contains a "bob" string, but where the middle 'o' char can be any char.

bobThere("abcbob") → true
bobThere("b9b") → true
bobThere("bac") → false

```
public boolean bobThere(String str) {
  boolean bob = false;
  for (int i=0; i<str.length()-2; i++) {
    if (str.charAt(i) == 'b' && str.charAt(i+2) == 'b') {
      bob = true;
    }
  }
  return bob;
}
```

We'll say that a String is xy-balanced if for all the 'x' chars in the string, there exists a 'y' char somewhere later in the string. So "xxy" is balanced, but "xyx" is not. One 'y' can balance multiple 'x's. Return true if the given string is xy-balanced.

xyBalance("aaxbby") → true
xyBalance("aaxbb") → false
xyBalance("yaaxbb") → false

```
public boolean xyBalance(String str) {
  boolean xy = true;
  for (int i = 0; i < str.length(); i++) {
     if (str.substring(i,i+1).equals("x")) {
        for (int k = i; k < str.length(); k++) {
           if (str.substring(k, k+1).equals("y")) {
              xy = true;
              break;
           } else {
              xy = false;
           }
        }
     }
  }
  return xy;
}
```

Given two strings, **a** and **b**, create a bigger string made of the first char of a, the first char of b, the second char of a, the second char of b, and so on. Any leftover chars go at the end of the result.

mixString("abc", "xyz") → "axbycz"
mixString("Hi", "There") → "HTihere"
mixString("xxxx", "There") → "xTxhxexre"

```
public String mixString(String a, String b) {
  int aIndex = 0;
  int bIndex = 0;
  String result = "";
  for (int i = 0; i < a.length()+b.length(); i++) {
    if (i%2 == 0) { // a
      if (aIndex < a.length()) {
```

```
      result += a.substring(aIndex, aIndex+1);
      aIndex++;
     } else {
      result += b.substring(bIndex); // rest of b
      return result; // need early return to ensure no extra letters get added
     }
    } else { // b
     if (bIndex < b.length()) {
      result += b.substring(bIndex, bIndex+1);
      bIndex++;
     } else {
      result += a.substring(aIndex); // rest of a
      return result; // need early return to ensure no extra letters get added
     }
    }
  }
  return result;
}
```

Given a string and an int n, return a string made of n repetitions of the last n characters of the string. You may assume that n is between 0 and the length of the string, inclusive.

repeatEnd("Hello", 3) → "llollollo"
repeatEnd("Hello", 2) → "lolo"
repeatEnd("Hello", 1) → "o"

```
public String repeatEnd(String str, int n) {
  String sub = str.substring(str.length()-n), result = "";
  for (int i = n; i > 0; i--) {
     result += sub;
  }
  return result;
}
```

Given a string and an int n, return a string made of the first n characters of the string, followed by the first n-1 characters of the string, and so on. You may assume that n is between 0 and the length of the string, inclusive (i.e. n >= 0 and n <= str.length()).

repeatFront("Chocolate", 4) → "ChocChoChC"
repeatFront("Chocolate", 3) → "ChoChC"
repeatFront("Ice Cream", 2) → "IcI"

```
public String repeatFront(String str, int n) {
  String newStr = "";
  while (n>0) {
    newStr += str.substring(0, n);
    n--;
  }
  return newStr;
}
```

Given two strings, **word** and a separator **sep**, return a big string made of **count** occurrences of the word, separated by the separator string.

repeatSeparator("Word", "X", 3) → "WordXWordXWord"
repeatSeparator("This", "And", 2) → "ThisAndThis"
repeatSeparator("This", "And", 1) → "This"

```
public String repeatSeparator(String word, String sep, int count) {
  String str = "";
  if (count == 0) {
    return str;
  }
  while (count>1) {
    str += (word + sep);
    count--;
  }
  return str + word;
}
```

Given a string, consider the prefix string made of the first N chars of the string. Does that prefix string appear somewhere else in the string? Assume that the string is not empty and that N is in the range 1..str.length().

prefixAgain("abXYabc", 1) → true
prefixAgain("abXYabc", 2) → true

prefixAgain("abXYabc", 3) → false

```
public boolean prefixAgain(String str, int n) {
  String pre = "";
  for (int i = 0; i < n; i++) {
    pre += str.substring(i, i+1);
  }
  return (str.substring(n).indexOf(pre) != -1); // Very sleek, very nice
}
```

Given a string, does "xyz" appear in the middle of the string? To define middle, we'll say that the number of chars to the left and right of the "xyz" must differ by at most one. This problem is harder than it looks.

xyzMiddle("AAxyzBB") → true
xyzMiddle("AxyzBB") → true
xyzMiddle("AxyzBBB") → false

```
/**
 * My solution, which doesn't work for the "other tests" (it works for
everything else)
 * I don't know why...
 *
public boolean xyzMiddle(String str) {
  int num = numberXYZ(str);
  if (str.length() < 3 || num == 0) {
    return false;
  } else if (num == 1) {
    int startIndex = str.indexOf("xyz");
    return (Math.abs((str.length()-startIndex-3)-(startIndex)) <= 1);
  } else {
    int midXYZ = (num/2)+1;
    int i = 0;
    int index = 0;
    while (i < midXYZ) {
      if (str.substring(index, index+3).equals("xyz")) {
        i++;
      }
      index++;
    }
    int midIndex = index-1;
    return (Math.abs((str.length()-midIndex-3)-(midIndex)) <= 1);
  }
}

public int numberXYZ(String str) {
  if (str.indexOf("xyz") == -1) {
    return 0;
  } else {
    return 1 + numberXYZ(str.substring(str.indexOf("xyz")+3));
  }
}
*/

// This solution is from
http://stackoverflow.com/questions/30346726/codingbat-xyzmiddle-fails-in-other-tests
public boolean xyzMiddle(String str) {
  if (str.indexOf("xyz") < 0)  return false;
  int l = str.length();
  int m = l / 2;
  if (l % 2 != 0) {
    if (!(str.substring((m - 1), m + 2).equals("xyz")))  return false;
  } else {
    if (str.charAt(m) != 'y' && str.charAt(m - 1) != 'y')  return false;
  }
  return true;
}
```

A sandwich is two pieces of bread with something in between. Return the string that is between the first and last appearance of "bread" in the given string, or return the empty string "" if there are not two pieces of bread.

getSandwich("breadjambread") → "jam"
getSandwich("xxbreadjambreadyy") → "jam"
getSandwich("xxbreadyy") → ""

```
public String getSandwich(String str) {
  String result = "";
  int start = 0;
  int end = 0;
  for (int i = 0; i < str.length()-4; i++) {
     if (str.substring(i,i+5).equals("bread")) {
        start = i+5;
        break;
     }
  }
  for (int k = 0; k < str.length()-4; k++) {
     if (str.substring(k,k+5).equals("bread")) {
        end = k;
     }
  }
  if (start == 0 || start == end+5) {
     return result;
  } else {
    return str.substring(start,end);
  }
}
```

Returns true if for every '*' (star) in the string, if there are chars both immediately before and after the star, they are the same.

sameStarChar("xy*yzz") → true
sameStarChar("xy*zzz") → false
sameStarChar("*xa*az") → true

```
public boolean sameStarChar(String str) {
  for (int i = 1; i < str.length()-1; i++) {
    if (str.charAt(i) == '*' && (str.charAt(i-1) != str.charAt(i+1))) {
      return false;
    }
  }
  return true;
}
```

Given a string, compute a new string by moving the first char to come after the next two chars, so "abc" yields "bca". Repeat this process for each subsequent group of 3 chars, so "abcdef" yields "bcaefd". Ignore any group of fewer than 3 chars at the end.

oneTwo("abc") → "bca"
oneTwo("tca") → "cat"
oneTwo("tcagdo") → "catdog"

```
public String oneTwo(String str) {
  String result = "";
  if (str.length() < 3) {
    return result;
  } else {
    int i = 0;
    while (i < str.length()-2) {
      result += str.substring(i+1,i+3) + str.substring(i,i+1);
      i += 3;
    }
  }
  return result;
}
```

Look for patterns like "zip" and "zap" in the string -- length-3, starting with 'z' and ending with 'p'. Return a string where for all such words, the middle letter is gone, so "zipXzap" yields "zpXzp".

zipZap("zipXzap") → "zpXzp"
zipZap("zopzop") → "zpzp"
zipZap("zzzopzop") → "zzzpzp"

```
public String zipZap(String str) {
  String result = "";
  if (str.length() < 3) {
    return str;
  }
  for (int i = 0; i < str.length(); i++) {
    if (str.charAt(i) == 'z' && str.charAt(i+2) == 'p') {
      result += "zp";
      i += 2;
    } else {
      result += str.substring(i, i+1);
    }
  }
  return result;
}
```

Return a version of the given string, where for every star (*) in the string the star and the chars immediately to its left and right are gone. So "ab*cd" yields "ad" and "ab**cd" also yields "ad".

starOut("ab*cd") → "ad"
starOut("ab**cd") → "ad"
starOut("sm*eilly") → "silly"

```
public String starOut(String str) {
  if (str.equals("*")) {
    return "";
  }
  ArrayList<String> myStr = new ArrayList<String>();
  for (int i = 0; i < str.length(); i++) {
    myStr.add(str.substring(i, i+1));
  }

  for (int k = myStr.size()-1; k >= 0; k--) {
    if (myStr.get(k).equals("*")) {
      if (k == 0) {
        myStr.remove(k);
        myStr.remove(k);
      } else if (k == myStr.size()-1) {
        myStr.remove(k-1);
        myStr.remove(k-1);
        k -= 2;
      } else if (myStr.get(k-1).equals("*")) {
        if (myStr.get(k-2).equals("*")) { // kind of shady as if there is "****",
code will not work, but that is unlikely
          myStr.remove(k-3);
          myStr.remove(k-3);
          myStr.remove(k-3);
          myStr.remove(k-3);
          myStr.remove(k-3);
          k -= 4;
        } else {
          myStr.remove(k-2);
          myStr.remove(k-2);
          myStr.remove(k-2);
          myStr.remove(k-2);
          k -= 3;
        }
      } else {
        myStr.remove(k-1);
        myStr.remove(k-1);
        myStr.remove(k-1);
        k -= 2;
      }
    }
  }

  String result = "";
  for (String s : myStr) {
    result += s;
  }
  return result;
}
```

Given a string and a non-empty **word** string, return a version of the original String where all chars have been replaced by pluses ("+"), except for appearances of the word string which are preserved unchanged.

plusOut("12xy34", "xy") → "++xy++"
plusOut("12xy34", "1") → "1+++++"
plusOut("12xy34xyabcxy", "xy") → "++xy++xy+++xy"

```
public String plusOut(String str, String word) {
  String result = "";
  for (int i = 0; i <= str.length()-word.length(); i++) {
     if (str.substring(i, i + word.length()).equals(word)) {
        result += word;
        i += word.length()-1;
```

```
    } else {
      result += "+";
    }
  }
  if (!str.substring(str.length()-word.length()).equals(word) && word.length()
!= 1) {
    result += "+";
  }
  if (!str.substring(str.length()-word.length()).equals(word) && word.length()
== 3) {
    result += "+";
  }
  return result;
}
```

Given a string and a non-empty **word** string, return a string made of each char just before and just after every appearance of the word in the string. Ignore cases where there is no char before or after the word, and a char may be included twice if it is between two words.

wordEnds("abcXY123XYijk", "XY") → "c13i"
wordEnds("XY123XY", "XY") → "13"
wordEnds("XY1XY", "XY") → "11"

```
public String wordEnds(String str, String word) {
  if (str.equals(word)) {
    return "";
  }
  String result = "";
  for (int i = 0; i <= str.length()-word.length(); i++) {
    if (str.substring(i, i+word.length()).equals(word)) {
      if (i == 0) {
        result += str.substring(word.length(), word.length()+1); // really
i+word.length(), but i = 0
      } else if (i == str.length()-word.length()) {
        result += str.substring(i-1, i);
      } else {
        result += str.substring(i-1, i);
        result += str.substring(i+word.length(), i+word.length()+1);
      }
    }
  }
  return result;
}
```

String-3

Given a string, count the number of words ending in 'y' or 'z' -- so the 'y' in "heavy" and the 'z' in "fez" count, but not the 'y' in "yellow" (not case sensitive). We'll say that a y or z is at the end of a word if there is not an alphabetic letter immediately following it. (Note: Character.isLetter(char) tests if a char is an alphabetic letter.)

countYZ("fez day") → 2
countYZ("day fez") → 2
countYZ("day fyyyz") → 2

```
public int countYZ(String str) {
  int count = 0;
  for (int i = 0; i < str.length()-1; i++) {
    if ((str.charAt(i) == 'y' || str.charAt(i) == 'z' || str.charAt(i) == 'Y' ||
str.charAt(i) == 'Z') && !Character.isLetter(str.charAt(i+1))) {
      count++;
    }
  }
  if (str.charAt(str.length()-1) == 'y' || str.charAt(str.length()-1) == 'z' ||
str.charAt(str.length()-1) == 'Y' || str.charAt(str.length()-1) == 'Z') {
      count++;
    }
  return count;
}
```

Given two strings, **base** and **remove**, return a version of the base string where all instances of the remove string have been removed (not case sensitive). You may assume that the remove string is length 1 or more. Remove only non-overlapping instances, so with "xxx" removing "xx" leaves "x".

withoutString("Hello there", "llo") → "He there"
withoutString("Hello there", "e") → "Hllo thr"

withoutString("Hello there", "x") → "Hello there"

```
public String withoutString(String base, String remove) {
  if (base.length() == 0)   return "";

  // Testing if entire array full of remove
  boolean full = true;
  for (int k = 0; k < base.length()-remove.length(); k++) {
    if (!base.substring(k, k+remove.length()).equals(remove)) {
      full = false;
    } else {
      k += remove.length()-1;
    }
  }
  if
(!base.substring(base.length()-remove.length()).equalsIgnoreCase(remove)
) {
    full = false;
  } else if (base.length()%2 != 0 && remove.length() == 2) {
    full = false;
  }
  if (full)   return "";

  String result = "";
  for (int i = 0; i < base.length()-remove.length(); i++) {
    if (!base.substring(i, i+remove.length()).equalsIgnoreCase(remove)) {
      result += base.substring(i, i+1);
    } else {
      i += remove.length()-1;
    }
  }
  if (base.substring(base.length()-remove.length()-1,
base.length()-1).equalsIgnoreCase(remove)) {
    result += base.substring(base.length()-1);
  } else if
(!base.substring(base.length()-remove.length()).equalsIgnoreCase(remove)
) {
    result += base.substring(base.length()-remove.length());
  }
  return result;
}
```

Given a string, return true if the number of appearances of "is" anywhere in the string is equal to the number of appearances of "not" anywhere in the string (case sensitive).

equalIsNot("This is not") → false
equalIsNot("This is notnot") → true
equalIsNot("noisxxnotyynotxisi") → true

```
public boolean equalIsNot(String str) {
  return (numberIs(str) == numberNot(str));
}

public int numberIs(String is) {
  if (is.indexOf("is") == -1) {
    return 0;
  } else {
    return 1 + numberIs(is.substring(is.indexOf("is")+2));
  }
}

public int numberNot(String not) {
  if (not.indexOf("not") == -1) {
    return 0;
  } else {
    return 1 + numberNot(not.substring(not.indexOf("not")+3));
  }
}
```

We'll say that a lowercase 'g' in a string is "happy" if there is another 'g' immediately to its left or right. Return true if all the g's in the given string are happy.

gHappy("xxggxx") → true
gHappy("xxgxx") → false
gHappy("xxggyygxx") → false

```
public boolean gHappy(String str) {
  if (str.length() < 2) {
    if (str.length() < 1 || !str.equals("g")) {
      return true;
    }
    return false;
  }

  if (str.charAt(0) == 'g' && str.charAt(1) != 'g') {
    return false;
  } else if (str.charAt(str.length()-1) == 'g' && str.charAt(str.length()-2) !=
'g') {
    return false;
  } else {
    for (int i = 1; i < str.length()-1; i++) {
      if (str.charAt(i) == 'g') {
        if (str.charAt(i-1) != 'g' && str.charAt(i+1) != 'g') { // DeMorgan! I was
thinking (!(str.charAt(i-1) == 'g' || str.charAt(i+1) == 'g')
          return false;
        }
      }
    }
  }
  return true;
}
```

We'll say that a "triple" in a string is a char appearing three times in a row. Return the number of triples in the given string. The triples may overlap.

countTriple("abcXXXabc") → 1
countTriple("xxxabyyyycd") → 3
countTriple("a") → 0

```
public int countTriple(String str) {
  int count = 0;
  for (int i = 0; i < str.length()-2; i++) {
    if (str.charAt(i) == str.charAt(i+1) && str.charAt(i+1) ==
str.charAt(i+2)) {
      count++;
    }
  }
  return count;
}
```

Given a string, return the sum of the digits 0-9 that appear in the string, ignoring all other characters. Return 0 if there are no digits in the string. (Note: Character.isDigit(char) tests if a char is one of the chars '0', '1', .. '9'. Integer.parseInt(string) converts a string to an int.)

sumDigits("aa1bc2d3") → 6
sumDigits("aa11b33") → 8
sumDigits("Chocolate") → 0

```
public int sumDigits(String str) {
  int sum = 0;
  for (int i = 0; i < str.length(); i++) {
    if (Character.isDigit(str.charAt(i))) {
      //sum += str.charAt(i); <--- Can't use because of ASCII values!
      sum += Integer.parseInt(str.substring(i, i+1));
    }
  }
  return sum;
}
```

Given a string, return the longest substring that appears at both the beginning and end of the string without overlapping. For example, sameEnds("abXab") is "ab".

sameEnds("abXYab") → "ab"
sameEnds("xx") → "x"
sameEnds("xxx") → "x"

```
public String sameEnds(String str) {
  String sub = "";
  int backIndex = str.length()/2;
  // Different for odd and even lengthed Strings
  if (str.length()%2 != 0) {
    backIndex++;
  }
  for (int i = str.length()/2; i >= 0; i--) {
    if (str.substring(0, i).equals(str.substring(backIndex))) { // I feel like
substring(0, i) should return an error when i = 0
      sub = str.substring(0, i);
      return sub;
    }
    backIndex++;
  }
  return sub;
}
```

Given a string, look for a mirror image (backwards) string at both the beginning and end of the given string. In other words, zero or more characters at the very begining of the given string, and at the very end of the string in reverse order (possibly overlapping). For example, the string "abXYZba" has the mirror end "ab".

mirrorEnds("abXYZba") → "ab"
mirrorEnds("abca") → "a"
mirrorEnds("aba") → "aba"

```
public String mirrorEnds(String str) {
  String sub = "";
  int backIndex = 0;
  /**
   * Curious...
   * For mirrorEnds, the length of str doesn't matter (in terms of odd or
even)
   * I guess it makes sense because in this problem, we don't care about
overlapping
   * It does make sense!
   *
  // Different for odd and even lengthed Strings
  if (str.length()%2 != 0) {
    backIndex++;
  }
  */
  for (int i = str.length(); i > 0; i--) {
    if (str.substring(0, i).equals(reverse(str.substring(backIndex)))) {
      sub = str.substring(0, i);
      return sub;
    }
    backIndex++;
  }
  return sub;
}

public String reverse(String rev) {
  String result = "";
  for (int i = rev.length(); i > 0; i--) {
    result += rev.substring(i-1, i);
  }
  return result;
}
```

Given a string, return the length of the largest "block" in the string. A block is a run of adjacent chars that are the same.

maxBlock("hoopla") → 2
maxBlock("abbCCCddBBBxx") → 3
maxBlock("") → 0

```
public int maxBlock(String str) {
  if (str.equals(""))  return 0;

  // I imported the code below from "RandomIntegers", a problem where we
found the longest run in an array
  int currLen = 0;
  int maxLen = 0;
  //int start = 0; // Don't need, but is nice (start is the starting index of
longest the block)

  for (int j = 0; j < str.length()-1; j++) {
    if (str.charAt(j) == str.charAt(j+1)) {
      currLen++;
      if (currLen >= maxLen) { // The >= accounts for same length runs
        maxLen = currLen;
      //start = j - currLen + 1; // Don't need, but is nice
     }
   } else {
     currLen = 0;
   }
  }
  return maxLen+1;
}
```

Given a string, return the sum of the numbers appearing in the string, ignoring all other characters. A number is a series of 1 or more digit chars in a row. (Note: Character.isDigit(char) tests if a char is one of the chars '0', '1', .. '9'. Integer.parseInt(string) converts a string to an int.)

sumNumbers("abc123xyz") → 123
sumNumbers("aa11b33") → 44
sumNumbers("7 11") → 18

```
public int sumNumbers(String str) {
  int sum = 0;
  for (int i = 0; i < str.length(); i++) {
    if (Character.isDigit(str.charAt(i))) {
      int temp = i+1;
      String tempNum = str.substring(i, i+1);
      while (temp < str.length() && Character.isDigit(str.charAt(temp))) {
        tempNum += str.substring(temp, temp+1);
        temp++;
      }
      sum += Integer.parseInt(tempNum);
      i = temp;
    }
  }
  return sum;
}
```

Given a string, return a string where every appearance of the lowercase word "is" has been replaced with "is not". The word "is" should not be immediately preceeded or followed by a letter -- so for example the "is" in "this" does not count. (Note: Character.isLetter(char) tests if a char is a letter.)

notReplace("is test") → "is not test"
notReplace("is-is") → "is not-is not"
notReplace("This is right") → "This is not right"

```
public String notReplace(String str) {
  if (str.equals("is"))   return "is not";
  if (str.length() < 3)   return str;
  String result = "";
  int temp = 1;
  if (str.substring(0, 2).equals("is")) {
    if (!Character.isLetter(str.charAt(2))) {
      result += "is not";
    } else {
      result += "is";
    }
    temp = 2;
  } else {
    result += str.substring(0, 1);
  }
  for (int i = temp; i < str.length()-2; i++) {
    if (str.substring(i, i+2).equals("is")) {
      if (!Character.isLetter(str.charAt(i-1)) &&
!Character.isLetter(str.charAt(i+2))) {
        result += "is not";
      } else {
        result += "is";
      }
      i++;
    } else {
      result += str.substring(i, i+1);
    }
  }
  // Still need to account for last two letters
  if (str.substring(str.length()-2).equals("is")) {
    if (!Character.isLetter(str.charAt(str.length()-3))) {
      result += "is not";
    } else {
      result += "is";
    }
  } else {
    result += str.substring(str.length()-2);
  }
  return result;
}
```

Array-1

Given an array of ints, return true if 6 appears as either the first or last element in the array. The array will be length 1 or more.

firstLast6([1, 2, 6]) → true
firstLast6([6, 1, 2, 3]) → true
firstLast6([13, 6, 1, 2, 3]) → false

```
public boolean firstLast6(int[] nums) {
  if (nums[0]==6 || nums[nums.length-1]==6) {
    return true;
  } else {
    return false;
  }
}
```

Given an array of ints, return true if the array is length 1 or more, and the first element and the last element are equal.

sameFirstLast([1, 2, 3]) → false
sameFirstLast([1, 2, 3, 1]) → true
sameFirstLast([1, 2, 1]) → true

```
public boolean sameFirstLast(int[] nums) {
  if (nums.length == 0) {
    return false;
  } else if (nums[0] == nums[nums.length-1]) {
    return true;
  } else {
    return false;
  }
}
```

Return an int array length 3 containing the first 3 digits of pi, {3, 1, 4}.

makePi() → [3, 1, 4]

```
public int[] makePi() {
  int[] pi = {3,1,4};
  return pi;
}
```

Given 2 arrays of ints, a and b, return true if they have the same first element or they have the same last element. Both arrays will be length 1 or more.

commonEnd([1, 2, 3], [7, 3]) → true
commonEnd([1, 2, 3], [7, 3, 2]) → false
commonEnd([1, 2, 3], [1, 3]) → true

```
public boolean commonEnd(int[] a, int[] b) {
  if (a[0] == b[0] || a[a.length-1] == b[b.length-1]) {
    return true;
  } else {
    return false;
  }
}
```

Given an array of ints length 3, return the sum of all the elements.

sum3([1, 2, 3]) → 6
sum3([5, 11, 2]) → 18
sum3([7, 0, 0]) → 7

```
public int sum3(int[] nums) {
  return nums[0]+nums[1]+nums[2];
}
```

Given an array of ints length 3, return an array with the elements "rotated left" so {1, 2, 3} yields {2, 3, 1}.

rotateLeft3([1, 2, 3]) → [2, 3, 1]
rotateLeft3([5, 11, 9]) → [11, 9, 5]
rotateLeft3([7, 0, 0]) → [0, 0, 7]

```
public int[] rotateLeft3(int[] nums) {
  int[] left = {nums[1],nums[2],nums[0]};
  return left;
}
```

Given an array of ints length 3, return a new array with the elements in reverse order, so {1, 2, 3} becomes {3, 2, 1}.

reverse3([1, 2, 3]) → [3, 2, 1]
reverse3([5, 11, 9]) → [9, 11, 5]
reverse3([7, 0, 0]) → [0, 0, 7]

```
public int[] reverse3(int[] nums) {
  int[] reverse = {nums[2], nums[1], nums[0]};
  return reverse;
}
```

Given an array of ints length 3, figure out which is larger between the first and last elements in the array, and set all the other elements to be that value. Return the changed array.

maxEnd3([1, 2, 3]) → [3, 3, 3]
maxEnd3([11, 5, 9]) → [11, 11, 11]
maxEnd3([2, 11, 3]) → [3, 3, 3]

```
public int[] maxEnd3(int[] nums) {
  int max = Math.max(nums[0],nums[2]);
  int[] same = {max, max, max};
  return same;
}
```

Given an array of ints, return the sum of the first 2 elements in the array. If the array length is less than 2, just sum up the elements that exist, returning 0 if the array is length 0.

sum2([1, 2, 3]) → 3
sum2([1, 1]) → 2
sum2([1, 1, 1, 1]) → 2

```
public int sum2(int[] nums) {
  if (nums.length == 0) {
    return 0;
  } else if (nums.length == 1) {
    return nums[0];
  } else {
    return nums[0] + nums[1];
  }
}
```

Given 2 int arrays, a and b, each length 3, return a new array length 2 containing their middle elements.

middleWay([1, 2, 3], [4, 5, 6]) → [2, 5]
middleWay([7, 7, 7], [3, 8, 0]) → [7, 8]
middleWay([5, 2, 9], [1, 4, 5]) → [2, 4]

```
public int[] middleWay(int[] a, int[] b) {
  int[] middle = {a[1],b[1]};
  return middle;
}
```

Given an array of ints, return a new array length 2 containing the first and last elements from the original array. The original array will be length 1 or more.

makeEnds([1, 2, 3]) → [1, 3]
makeEnds([1, 2, 3, 4]) → [1, 4]
makeEnds([7, 4, 6, 2]) → [7, 2]

```
public int[] makeEnds(int[] nums) {
  int[] nums1 = {nums[0], nums[nums.length-1]};
  return nums1;
}
```

Given an int array length 2, return true if it contains a 2 or a 3.

has23([2, 5]) → true

has23([4, 3]) → true
has23([4, 5]) → false

```
public boolean has23(int[] nums) {
  if (nums[0] == 2 || nums[0] == 3 || nums[1] == 2 || nums[1] == 3) {
    return true;
  } else {
    return false;
  }
}
```

Given an int array length 2, return true if it does not contain a 2 or 3.

no23([4, 5]) → true
no23([4, 2]) → false
no23([3, 5]) → false

```
public boolean no23(int[] nums) {
  if (nums[0] == 2 || nums[0] == 3 || nums[1] == 2 || nums[1] == 3) {
    return false;
  } else {
    return true;
  }
}
```

Given an int array, return a new array with double the length where its last element is the same as the original array, and all the other elements are 0. The original array will be length 1 or more. Note: by default, a new int array contains all 0's.

makeLast([4, 5, 6]) → [0, 0, 0, 0, 0, 6]
makeLast([1, 2]) → [0, 0, 0, 2]
makeLast([3]) → [0, 3]

```
public int[] makeLast(int[] nums) {
  int[] nums1 = new int[2*nums.length];
  nums1[nums1.length-1] = nums[nums.length-1];
  return nums1;
}
```

Given an int array, return true if the array contains 2 twice, or 3 twice. The array will be length 0, 1, or 2.

double23([2, 2]) → true
double23([3, 3]) → true
double23([2, 3]) → false

```
public boolean double23(int[] nums) {
  if (nums.length != 2) {
    return false;
  } else if ((nums[0] == 2 && nums[1] == 2) || (nums[0] == 3 && nums[1]
== 3)) {
    return true;
  } else {
    return false;
  }
}
```

Given an int array length 3, if there is a 2 in the array immediately followed by a 3, set the 3 element to 0. Return the changed array.

fix23([1, 2, 3]) → [1, 2, 0]
fix23([2, 3, 5]) → [2, 0, 5]
fix23([1, 2, 1]) → [1, 2, 1]

```
public int[] fix23(int[] nums) {
  if (nums[0] == 2) {
     if (nums[1] == 3) {
        nums[1] = 0;
     } else if (nums[1] == 2) {
     nums[2] = 0;
     }
        return nums;
  } else if (nums[1] == 2) {
     if (nums[2] == 3) {
        nums[2] = 0;
     }
        return nums;
  } else {
  return nums;
  }
}
```

Start with 2 int arrays, a and b, of any length. Return how many of the arrays have 1 as their first element.

start1([1, 2, 3], [1, 3]) → 2
start1([7, 2, 3], [1]) → 1
start1([1, 2], []) → 1

```
public int start1(int[] a, int[] b) {
  if (a.length == 0 && b.length == 0) {
    return 0;
  } else if (a.length == 0 && b[0] == 1) {
    return 1;
  } else if (a[0] == 1 && b.length == 0) {
    return 1;
  } else if (a[0] == 1 && b[0] == 1) {
    return 2;
  } else if ((a[0] != 1 && b.length == 0) || (b[0] != 1 && a.length == 0)) {
    return 0;
  } else if ((a[0] == 1 && b[0] != 1) || (a[0] != 1 && b[0] == 1)) {
    return 1;
  } else {
    return 0;
  }
}
```

Start with 2 int arrays, a and b, each length 2. Consider the sum of the values in each array. Return the array which has the largest sum. In event of a tie, return a.

biggerTwo([1, 2], [3, 4]) → [3, 4]
biggerTwo([3, 4], [1, 2]) → [3, 4]
biggerTwo([1, 1], [1, 2]) → [1, 2]

```
public int[] biggerTwo(int[] a, int[] b) {
  if ((a[0] + a[1]) == (b[0] + b[1])) {
    return a;
  } else if ((a[0] + a[1]) > (b[0] + b[1])) {
    return a;
  } else {
    return b;
  }
}
```

Given an array of ints of even length, return a new array length 2 containing the middle two elements from the original array. The original array will be length 2 or more.

makeMiddle([1, 2, 3, 4]) → [2, 3]
makeMiddle([7, 1, 2, 3, 4, 9]) → [2, 3]
makeMiddle([1, 2]) → [1, 2]

```
public int[] makeMiddle(int[] nums) {
  int first = nums[(int)((nums.length-1)/2)];
  int second = nums[(int)((nums.length)/2)];
  int[] nums1 = {first, second};
  return nums1;
}
```

Given 2 int arrays, each length 2, return a new array length 4 containing all their elements.

plusTwo([1, 2], [3, 4]) → [1, 2, 3, 4]
plusTwo([4, 4], [2, 2]) → [4, 4, 2, 2]
plusTwo([9, 2], [3, 4]) → [9, 2, 3, 4]

```
public int[] plusTwo(int[] a, int[] b) {
  int first = a[0];
  int second = a[1];
  int third = b[0];
  int fourth = b[1];
  int[] nums = {first,second,third,fourth};
  return nums;
}
```

Given an array of ints, swap the first and last elements in the array. Return the modified array. The array length will be at least 1.

swapEnds([1, 2, 3, 4]) → [4, 2, 3, 1]
swapEnds([1, 2, 3]) → [3, 2, 1]
swapEnds([8, 6, 7, 9, 5]) → [5, 6, 7, 9, 8]

```
public int[] swapEnds(int[] nums) {
  int first = nums[0];
  int last = nums[nums.length-1];
  nums[0] = last;
  nums[nums.length-1] = first;
  return nums;
}
```

Given an array of ints of odd length, return a new array length 3 containing the elements from the middle of the array. The array length will be at least 3.

midThree([1, 2, 3, 4, 5]) → [2, 3, 4]
midThree([8, 6, 7, 5, 3, 0, 9]) → [7, 5, 3]
midThree([1, 2, 3]) → [1, 2, 3]

```
public int[] midThree(int[] nums) {
  int[] three = {nums[nums.length/2-1], nums[nums.length/2],
nums[nums.length/2+1]};
  return three;
}
```

Given an array of ints of odd length, look at the first, last, and middle values in the array and return the largest. The array length will be a least 1.

maxTriple([1, 2, 3]) → 3
maxTriple([1, 5, 3]) → 5
maxTriple([5, 2, 3]) → 5

```
public int maxTriple(int[] nums) {
  int max = nums[0];
  if (nums[nums.length/2] > max) {
     max = nums[nums.length/2];
     if (nums[nums.length-1] > max) {
        max = nums[nums.length-1];
     }
  } else if (nums[nums.length-1] > max) {
     max = nums[nums.length-1];
  }
  return max;
}
```

Given an int array of any length, return a new array of its first 2 elements. If the array is smaller than length 2, use whatever elements are present.

frontPiece([1, 2, 3]) → [1, 2]
frontPiece([1, 2]) → [1, 2]
frontPiece([1]) → [1]

```
public int[] frontPiece(int[] nums) {
  if (nums.length < 2) {
     return nums;
  } else {
     int[] two = {nums[0], nums[1]};
     return two;
  }
}
```

We'll say that a 1 immediately followed by a 3 in an array is an "unlucky" 1. Return true if the given array contains an unlucky 1 in the first 2 or last 2 positions in the array.

unlucky1([1, 3, 4, 5]) → true
unlucky1([2, 1, 3, 4, 5]) → true
unlucky1([1, 1, 1]) → false

```
public boolean unlucky1(int[] nums) {
  boolean unlucky = false;
  if (nums.length < 2) {
     unlucky = false;
  } else if (nums[0]==1 && nums[1]!=1) {
     if (nums[1]==3) {
       unlucky = true;
     } else if (nums[nums.length-2]==1) {
       if (nums[nums.length-1]==3) {
         unlucky = true;
       }
     }
  } else if (nums[1]==1 && nums.length>2) {
     if (nums[2]==3) {
       unlucky = true;
     } else if (nums[nums.length-2]==1) {
       if (nums[nums.length-1]==3) {
         unlucky = true;
       }
     }
  } else if (nums[nums.length-2]==1) {
     if (nums[nums.length-1]==3) {
       unlucky = true;
     }
  } else {
     unlucky = false;
  }
  return unlucky;
}
```

Given 2 int arrays, a and b, return a new array length 2 containing, as much as will fit, the elements from a followed by the elements from b. The arrays may be any length, including 0, but there will be 2 or more elements available between the 2 arrays.

make2([4, 5], [1, 2, 3]) → [4, 5]
make2([4], [1, 2, 3]) → [4, 1]
make2([], [1, 2]) → [1, 2]

```
public int[] make2(int[] a, int[] b) {
  if (a.length==0) {
    int[] arr = {b[0], b[1]};
    return arr;
  } else if (b.length==0) {
    int[] arr = {a[0], a[1]};
    return arr;
  } else if (a.length==1 || b.length==1) {
    int[] arr = {a[0], b[0]};
    return arr;
  } else {
    int[] arr = {a[0], a[1]};
    return arr;
  }
}
```

Given 2 int arrays, a and b, of any length, return a new array with the first element of each array. If either array is length 0, ignore that array.

front11([1, 2, 3], [7, 9, 8]) → [1, 7]
front11([1], [2]) → [1, 2]
front11([1, 7], []) → [1]

```
public int[] front11(int[] a, int[] b) {
  if (a.length==0 && b.length==0) {
    int[] arr = {};
    return arr;
  } else if (a.length==0) {
    int[] arr = {b[0]};
    return arr;
  } else if (b.length==0) {
    int[] arr = {a[0]};
    return arr;
  } else {
    int[] arr = {a[0], b[0]};
    return arr;
  }
}
```

Array-2

Return the number of even ints in the given array. Note: the % "mod" operator computes the remainder, e.g. 5 % 2 is 1.

countEvens([2, 1, 2, 3, 4]) → 3
countEvens([2, 2, 0]) → 3
countEvens([1, 3, 5]) → 0

```
public int countEvens(int[] nums) {
  int k = 0;
  for (int i = 0; i < nums.length; i++) {
  if (nums[i] % 2 == 0) {
  k++;
  } else {}
  }
  return k;
}
```

Given an array length 1 or more of ints, return the difference between the largest and smallest values in the array. Note: the built-in Math.min(v1, v2) and Math.max(v1, v2) methods return the smaller or larger of two values.

bigDiff([10, 3, 5, 6]) → 7
bigDiff([7, 2, 10, 9]) → 8
bigDiff([2, 10, 7, 2]) → 8

```
public int bigDiff(int[] nums) {
  int max = nums[0];
  int min = nums[0];
  for (int i = 1; i<nums.length; i++) {
  max = Math.max(nums[i],max);
  min = Math.min(nums[i],min);
  }
  return max-min;
}
```

Return the "centered" average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division to produce the final average. You may assume that the array is length 3 or more.

centeredAverage([1, 2, 3, 4, 100]) → 3
centeredAverage([1, 1, 5, 5, 10, 8, 7]) → 5
centeredAverage([-10, -4, -2, -4, -2, 0]) → -3

```
public int centeredAverage(int[] nums) {
  if (allSameNum(nums)) {
    return nums[0];
  }

  int max = Integer.MIN_VALUE;
  int min = Integer.MAX_VALUE;
  for (int i : nums) {
    if (i > max) {
      max = i;
    }
    if (i < min) {
      min = i;
    }
  }

  int[] myNewNums = new int[nums.length-2];
  boolean pastMin = false;
  boolean pastMax = false;
  int index = 0;
  for (int k = 0; k < nums.length; k++) {
    if (nums[k] == min && !pastMin) {
      pastMin = true;
    } else if (nums[k] == min && pastMin) {
      myNewNums[index] = nums[k];
      index++;
    } else if (nums[k] == max && !pastMax) {
      pastMax = true;
    } else if (nums[k] == max && pastMax) {
      myNewNums[index] = nums[k];
      index++;
    } else {
      myNewNums[index] = nums[k];
      index++;
    }
  }

  int sum = 0;
  for (int j : myNewNums) {
    sum += j;
  }
  return sum/myNewNums.length;
}

public boolean allSameNum(int[] same) {
  for (int i = 0; i < same.length-1; i++) {
    if (same[i] != same[i+1]) {
      return false;
    }
  }
  return true;
}
```

Return the sum of the numbers in the array, returning 0 for an empty array. Except the number 13 is very unlucky, so it does not count and numbers that come immediately after a 13 also do not count.

sum13([1, 2, 2, 1]) → 6
sum13([1, 1]) → 2
sum13([1, 2, 2, 1, 13]) → 6

```
public int sum13(int[] nums) {
  int sum = 0;
  for (int i = 0; i<nums.length; i++) {
    if(nums[i]!=13) {
      sum += nums[i];
    } else {
      i++;
    }
  }
  return sum;
}
```

Return the sum of the numbers in the array, except ignore sections of numbers starting with a 6 and extending to the next 7 (every 6 will be followed by at least one 7). Return 0 for no numbers.

sum67([1, 2, 2]) → 5
sum67([1, 2, 2, 6, 99, 99, 7]) → 5
sum67([1, 1, 6, 7, 2]) → 4

```
public int sum67(int[] nums) {
  if (nums.length < 1) {
    return 0;
  }

  int sum = 0;
  for (int i = 0; i < nums.length; i++) {
    if (nums[i] != 6) {
      sum += nums[i];
    } else {
      for (int k = i+1; k < nums.length; k++) {
        if (nums[k] == 7) {
          i = k; // Really k+1-1, but y'know...substring rules
          break;
        }
      }
    }
  }
  return sum;
}
```

Given an array of ints, return true if the array contains a 2 next to a 2 somewhere.

has22([1, 2, 2]) → true
has22([1, 2, 1, 2]) → false

has22([2, 1, 2]) → false

```
public boolean has22(int[] nums) {
  boolean two = false;
  for (int i = 0; i < nums.length-1; i++) {
     if (nums[i] == 2 && nums[i+1] == 2) {
        two = true;
        break;
     } else {
        two = false;
     }
  }
  return two;
}
```

Given an array of ints, return true if the array contains no 1's and no 3's.

lucky13([0, 2, 4]) → true
lucky13([1, 2, 3]) → false
lucky13([1, 2, 4]) → false

```
public boolean lucky13(int[] nums) {
  boolean lucky = true;
  for (int i = 0; i < nums.length; i++) {
     if (nums[i] == 1 || nums[i] == 3) {
        lucky = false;;
        break;
     } else {
        lucky = true;
     }
  }
  return lucky;
}
```

Given an array of ints, return true if the sum of all the 2's in the array is exactly 8.

sum28([2, 3, 2, 2, 4, 2]) → true
sum28([2, 3, 2, 2, 4, 2, 2]) → false

sum28([1, 2, 3, 4]) → false

```
public boolean sum28(int[] nums) {
  int count2 = 0;
  for (int i = 0; i < nums.length; i++) {
     if (nums[i] == 2) {
        count2++;
     }
  }
  if (count2 == 4) {
     return true;
  } else {
     return false;
  }
}
```

Given an array of ints, return true if the number of 1's is greater than the number of 4's

more14([1, 4, 1]) → true
more14([1, 4, 1, 4]) → false
more14([1, 1]) → true

```
public boolean more14(int[] nums) {
  int count1 = 0, count4 = 0;
  for (int i = 0; i < nums.length; i++) {
     if (nums[i] == 1) {
        count1++;
     } else if (nums[i] == 4) {
        count4++;
     }
  }
  if (count1>count4) {
     return true;
  } else {
     return false;
  }
}
```

Given a number n, create and return a new int array of length n, containing the numbers 0, 1, 2, ... n-1. The given n may be 0, in which case just return a length 0 array. You do not need a separate if-statement for the length-0 case; the for-loop should naturally execute 0 times in that case, so it just works. The syntax to make a new int array is: new int[desired_length]
(See also: FizzBuzz Code)

fizzArray(4) → [0, 1, 2, 3]
fizzArray(1) → [0]
fizzArray(10) → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
public int[] fizzArray(int n) {
  int[] fizz = new int[n];
  for (int i = 0; i < fizz.length; i++) {
     fizz[i] = i;
  }
  return fizz;
}
```

Given an array of ints, return true if every element is a 1 or a 4.

only14([1, 4, 1, 4]) → true
only14([1, 4, 2, 4]) → false
only14([1, 1]) → true

```
public boolean only14(int[] nums) {
  int count14 = 0;
  for (int i = 0; i < nums.length; i++) {
     if (nums[i] == 1 || nums[i] == 4) {
        count14++;
     }
  }
  if (count14 == nums.length) {
     return true;
  } else {
     return false;
  }
}
```

Given a number n, create and return a new string array of length n, containing the strings "0", "1" "2" .. through n-1. N may be 0, in which case just return a length 0 array. Note: String.valueOf(xxx) will make the String form of most types. The syntax to make a new string array is: new String[desired_length] (See also: FizzBuzz Code)

```
fizzArray2(4) → ["0", "1", "2", "3"]
fizzArray2(10) → ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
fizzArray2(2) → ["0", "1"]

public String[] fizzArray2(int n) {
  String[] nums = new String[n];
  for (int i = 0; i < nums.length; i++) {
     nums[i] = String.valueOf(i);
  }
  return nums;
}
```

Given an array of ints, return true if it contains no 1's or it contains no 4's.

```
no14([1, 2, 3]) → true
no14([1, 2, 3, 4]) → false
no14([2, 3, 4]) → true

public boolean no14(int[] nums) {
  // To make up for XOR
  // If nums contains no 1's or no 4's, XOR returns false
  if (!containsOneOrFour(nums)) {
    return true;
  }
  return containsOne(nums)^containsFour(nums); // XOR to the rescue!
  // If nums contains a 1 or a 4, XOR will return false, as intended
  // XOR Table below
//  X  Y      Output
//  0  0      0      // First statment (!containsOneOrFour) changes this output
to true (1)
//  0  1      1
//  1  0      1
//  1  1      0
}

public boolean containsOne(int[] one) {
  for (int i : one) {
    if (i == 1) {
      return true;
    }
  }
  return false;
}

public boolean containsFour(int[] four) {
  for (int i : four) {
    if (i == 4) {
      return true;
    }
  }
  return false;
}

public boolean containsOneOrFour(int[] both) {
  for (int i : both) {
    if (i == 1 || i == 4) {
      return true;
    }
  }
  return false;
}
```

We'll say that a value is "everywhere" in an array if for every pair of adjacent elements in the array, at least one of the pair is that value. Return true if the given value is everywhere in the array.

isEverywhere([1, 2, 1, 3], 1) → true
isEverywhere([1, 2, 1, 3], 2) → false
isEverywhere([1, 2, 1, 3, 4], 1) → false

```
public boolean isEverywhere(int[] nums, int val) {
  for (int i = 0; i < nums.length-1; i ++) {
    if (!(nums[i] == val || nums[i+1] == val)) {
      return false;
    }
  }
  return true;
}
```

Given an array of ints, return true if the array contains a 2 next to a 2 or a 4 next to a 4, but not both.

either24([1, 2, 2]) → true
either24([4, 4, 1]) → true
either24([4, 4, 1, 2, 2]) → false

```
public boolean either24(int[] nums) {
  return containsTwo(nums)^containsFour(nums);
}

public boolean containsTwo(int[] nums) {
  for (int i = 0; i < nums.length-1; i++) {
    if (nums[i] == 2 && nums[i+1] == 2) {
      return true;
    }
  }
  return false;
}

public boolean containsFour(int[] nums) {
  for (int i = 0; i < nums.length-1; i++) {
    if (nums[i] == 4 && nums[i+1] == 4) {
      return true;
    }
  }
  return false;
}
```

Given arrays nums1 and nums2 of the same length, for every element in nums1, consider the corresponding element in nums2 (at the same index).

Return the count of the number of times that the two elements differ by 2 or less, but are not equal.

matchUp([1, 2, 3], [2, 3, 10]) → 2
matchUp([1, 2, 3], [2, 3, 5]) → 3
matchUp([1, 2, 3], [2, 3, 3]) → 2

```
public int matchUp(int[] nums1, int[] nums2) {
  int count = 0;
  for (int i = 0; i < nums1.length; i++) {
    if (differ(nums1[i], nums2[i])) {
      count++;
    }
  }
  return count;
}

public boolean differ(int a, int b) {
  if (a == b) {
    return false;
  } else if (Math.abs(a-b) <= 2) {
    return true;
  } else {
    return false;
  }
}
```

Given an array of ints, return true if the array contains two 7's next to each other, or there are two 7's separated by one element, such as with {7, 1, 7}.

has77([1, 7, 7]) → true
has77([1, 7, 1, 7]) → true
has77([1, 7, 1, 1, 7]) → false

```
public boolean has77(int[] nums) {
  for (int i = 0; i < nums.length-1; i++) {
    if (nums[i] == 7 && nums[i+1] == 7) {
      return true;
    }
  }
  for (int k = 0; k < nums.length-2; k++) {
    if (nums[k] == 7 && nums[k+2] == 7) {
      return true;
    }
  }
  return false;
}
```

Given an array of ints, return true if there is a 1 in the array with a 2 somewhere later in the array.

```
has12([1, 3, 2]) → true
has12([3, 1, 2]) → true
has12([3, 1, 4, 5, 2]) → true
```

```
public boolean has12(int[] nums) {
  for (int i = 0; i < nums.length; i++) {
    if (nums[i] == 1) {
      for (int k = i+1; k < nums.length; k++) {
        if (nums[k] == 2) {
          return true;
        }
      }
    }
  }
  return false;
}
```

Given an array of ints, return true if the array contains either 3 even or 3 odd values all next to each other.

```
modThree([2, 1, 3, 5]) → true
modThree([2, 1, 2, 5]) → false
modThree([2, 4, 2, 5]) → true
```

```
public boolean modThree(int[] nums) {
  for (int i = 0; i < nums.length-2; i++) {
    if (nums[i]%2 == 0 && nums[i+1]%2 == 0 && nums[i+2]%2 == 0) {
      return true;
    } else if (nums[i]%2 != 0 && nums[i+1]%2 != 0 && nums[i+2]%2 != 0)
{
      return true;
    }
  }
  return false;
}
```

Given an array of ints, return true if the value 3 appears in the array exactly 3 times, and no 3's are next to each other.

haveThree([3, 1, 3, 1, 3]) → true
haveThree([3, 1, 3, 3]) → false
haveThree([3, 4, 3, 3, 4]) → false

```
public boolean haveThree(int[] nums) {
  for (int i = 0; i < nums.length-1; i++) {
    if (nums[i] == 3 && nums[i+1] == 3) {
      return false; // early check
    }
  }

  int count = 0;
  for (int k : nums) {
    if (k == 3) {
      count++;
    }
  }

  return (count == 3);
}

/**
 * Original Attempt
 * I'm not sure why it didn't work for the "other tests"
 * It could be an internal website issue
```

```
public boolean haveThree(int[] nums) {
  int countThree = 0;
  for (int i = 0; i < nums.length-1; i++) {
    if (nums[i] == 3 && nums[i+1] == 3) {
      return false;
    } else if (nums[i] == 3) {
      countThree++;
    }
  }
  if (nums[nums.length-1] == 3) { // Loop didn't catch last element
    countThree++;
  }
  return (countThree == 3);
}
*/
```

Given an array of ints, return true if every 2 that appears in the array is next to another 2.

twoTwo([4, 2, 2, 3]) → true
twoTwo([2, 2, 4]) → true
twoTwo([2, 2, 4, 2]) → false

```
public boolean twoTwo(int[] nums) {
  if (nums.length < 1) {
    return true;
  }
  if (nums[0] == 2 && nums.length == 1) {
    return false;
  }
  if (nums[0] != 2 && nums.length == 1) {
    return true;
  }
  if (nums[0] == 2 && nums[1] != 2) {
    return false;
  }
  for (int i = 1; i < nums.length-1; i++) {
    if (nums[i] == 2 && nums[i-1] != 2 && nums[i+1] != 2) {
      return false;
    }
  }
  if (nums[nums.length-2] != 2 && nums[nums.length-1] == 2) {
    return false;
  }
  return true;
}
```

Return true if the group of N numbers at the start and end of the array are the same. For example, with {5, 6, 45, 99, 13, 5, 6}, the ends are the same for n=0 and n=2, and false for n=1 and n=3. You may assume that n is in the range 0..nums.length inclusive.

sameEnds([5, 6, 45, 99, 13, 5, 6], 1) → false
sameEnds([5, 6, 45, 99, 13, 5, 6], 2) → true
sameEnds([5, 6, 45, 99, 13, 5, 6], 3) → false

```
public boolean sameEnds(int[] nums, int len) {
  int[] temp = new int[len];
  for (int i = 0; i < len; i++) {
    temp[i] = nums[i];
  }

  int count = nums.length-len;
  for (int i = 0; i < temp.length; i++) {
    if (temp[i] != nums[count]) {
      return false;
    }
    count++;
  }
  return true;
}
```

Return true if the array contains, somewhere, three increasing adjacent numbers like .... 4, 5, 6, ... or 23, 24, 25.

tripleUp([1, 4, 5, 6, 2]) → true
tripleUp([1, 2, 3]) → true
tripleUp([1, 2, 4]) → false

```
public boolean tripleUp(int[] nums) {
  for (int i = 0; i < nums.length-2; i++) {
    if (nums[i] == nums[i+1]-1 && nums[i+1] == nums[i+2]-1) {
      return true;
    }
  }
  return false;
}
```

Given **start** and **end** numbers, return a new array containing the sequence of integers from start up to but not including end, so start=5 and end=10 yields {5, 6, 7, 8, 9}. The end number will be greater or equal to the start number. Note that a length-0 array is valid. (See also: FizzBuzz Code)

fizzArray3(5, 10) → [5, 6, 7, 8, 9]
fizzArray3(11, 18) → [11, 12, 13, 14, 15, 16, 17]
fizzArray3(1, 3) → [1, 2]

```
public int[] fizzArray3(int start, int end) {
  int[] result = new int[end-start];
  int index = 0;
  // result.length could be (and probably is less than start)
  for (int i = start; i < end; i++) {
    result[index] = i;
    index++;
  }
  return result;
}
```

Return an array that is "left shifted" by one -- so {6, 2, 5, 3} returns {2, 5, 3, 6}. You may modify and return the given array, or return a new array.

shiftLeft([6, 2, 5, 3]) → [2, 5, 3, 6]
shiftLeft([1, 2]) → [2, 1]
shiftLeft([1]) → [1]

```
public int[] shiftLeft(int[] nums) {
  if (nums.length < 1) {
    return new int[0];
  }
  int[] result = new int[nums.length];
  for (int i = 1; i < result.length; i++) {
    result[i-1] = nums[i];
  }
  result[result.length-1] = nums[0];
  return result;
}
```

For each multiple of 10 in the given array, change all the values following it to be that multiple of 10, until encountering another multiple of 10. So {2, 10, 3, 4, 20, 5} yields {2, 10, 10, 10, 20, 20}.

tenRun([2, 10, 3, 4, 20, 5]) → [2, 10, 10, 10, 20, 20]
tenRun([10, 1, 20, 2]) → [10, 10, 20, 20]
tenRun([10, 1, 9, 20]) → [10, 10, 10, 20]

```
public int[] tenRun(int[] nums) {
  for (int i = 0; i < nums.length-1; i++) {
    if (nums[i]%10 == 0) {
      int now = i+1;
      while (now < nums.length && nums[now]%10 != 0) {
        nums[now] = nums[i];
        now++;
      }
      i = now-1;
    }
  }
  return nums;
}
```

Given a non-empty array of ints, return a new array containing the elements from the original array that come before the first 4 in the original array. The original array will contain at least one 4. Note that it is valid in java to create an array of length 0.

pre4([1, 2, 4, 1]) → [1, 2]
pre4([3, 1, 4]) → [3, 1]

```
pre4([1, 4, 4]) → [1]

public int[] pre4(int[] nums) {
  int four = 0;
  for (int i = 0; i < nums.length; i++) {
    if (nums[i] == 4) {
      four = i;
      break;
    }
  }

  int[] result = new int[four];
  for (int i = 0; i < result.length; i++) {
    result[i] = nums[i];
  }
  return result;
}
```

Given a non-empty array of ints, return a new array containing the elements from the original array that come after the last 4 in the original array. The original array will contain at least one 4. Note that it is valid in java to create an array of length 0.

```
post4([2, 4, 1, 2]) → [1, 2]
post4([4, 1, 4, 2]) → [2]
post4([4, 4, 1, 2, 3]) → [1, 2, 3]

public int[] post4(int[] nums) {
  int four = 0;
  // Need to start at the end in case there is more than one four
  for (int i = nums.length-1; i > 0; i--) {
    if (nums[i] == 4) {
      four = i;
      break;
    }
  }

  int[] result = new int[nums.length-four-1];
  int index = 0;
  // Again, you can't use result.length!
  for (int i = four+1; i < nums.length; i++) {
    result[index] = nums[i];
    index++;
  }
  return result;
}
```

We'll say that an element in an array is "alone" if there are values before and after it, and those values are different from it. Return a version of the given array where every instance of the given value which is alone is replaced by whichever value to its left or right is larger.

notAlone([1, 2, 3], 2) → [1, 3, 3]
notAlone([1, 2, 3, 2, 5, 2], 2) → [1, 3, 3, 5, 5, 2]
notAlone([3, 4], 3) → [3, 4]

```
public int[] notAlone(int[] nums, int val) {
  for (int i = 1; i < nums.length-1; i++) {
    if (nums[i] == val && nums[i-1] != val && nums[i+1] != val) {
      if (nums[i-1] > nums[i+1]) {
        nums[i] = nums[i-1];
      } else {
        nums[i] = nums[i+1];
      }
    }
  }
  return nums;
}
```

Return an array that contains the exact same numbers as the given array, but rearranged so that all the zeros are grouped at the start of the array. The order of the non-zero numbers does not matter. So {1, 0, 0, 1} becomes {0 ,0, 1, 1}. You may modify and return the given array or make a new array.

zeroFront([1, 0, 0, 1]) → [0, 0, 1, 1]
zeroFront([0, 1, 1, 0, 1]) → [0, 0, 1, 1, 1]
zeroFront([1, 0]) → [0, 1]

```
public int[] zeroFront(int[] nums) {
  int[] result = new int[nums.length];
  int count = 0; // Count of zeros
  for (int i = 0; i < nums.length; i++) {
    if (nums[i] == 0) {
      count++;
    }
  }

  int[] nonZero = new int[nums.length-count];
  int index = 0;
  for (int k = 0; k < nums.length; k++) {
    if (nums[k] != 0) {
      nonZero[index] = nums[k];
      index++;
    }
  }

  for (int j = 0; j < nonZero.length; j++) {
    result[j + count] = nonZero[j];
  }
  return result;
}
```

Return a version of the given array where all the 10's have been removed. The remaining elements should shift left towards the start of the array as needed, and the empty spaces a the end of the array should be 0. So {1, 10, 10, 2} yields {1, 2, 0, 0}. You may modify and return the given array or make a new array.

```
withoutTen([1, 10, 10, 2]) → [1, 2, 0, 0]
withoutTen([10, 2, 10]) → [2, 0, 0]
withoutTen([1, 99, 10]) → [1, 99, 0]
```

```
public int[] withoutTen(int[] nums) {
  int[] result = new int[nums.length];
  int index = 0;
  for (int i = 0; i < nums.length; i++) {
    if (nums[i] != 10) {
      result[index] = nums[i];
      index++;
    }
  }
  return result;
}
```

Return a version of the given array where each zero value in the array is replaced by the largest odd value to the right of the zero in the array. If there is no odd value to the right of the zero, leave the zero as a zero.

```
zeroMax([0, 5, 0, 3]) → [5, 5, 3, 3]
zeroMax([0, 4, 0, 3]) → [3, 4, 3, 3]
zeroMax([0, 1, 0]) → [1, 1, 0]
```

```
public int[] zeroMax(int[] nums) {
  for (int i = 0; i < nums.length; i++) {
    if (nums[i] == 0) {
      int max = -1;
      for (int k = i+1; k < nums.length; k++) {
        if (nums[k]%2 != 0 && nums[k] > max) {
          max = nums[k];
        }
      }
      if (max != -1) {
        nums[i] = max;
      }
    }
  }
  return nums;
}
```

Return an array that contains the exact same numbers as the given array, but rearranged so that all the even numbers come before all the odd numbers. Other than that, the numbers can be in any order. You may modify and return the given array, or make a new array.

```
evenOdd([1, 0, 1, 0, 0, 1, 1]) → [0, 0, 0, 1, 1, 1, 1]
```

evenOdd([3, 3, 2]) → [2, 3, 3]
evenOdd([2, 2, 2]) → [2, 2, 2]

```
public int[] evenOdd(int[] nums) {
  int[] result = new int[nums.length];
  int index = 0;
  for (int i = 0; i < nums.length; i++) {
    if (nums[i]%2 == 0) {
      result[index] = nums[i];
      index++;
    }
  }
  for (int k = 0; k < nums.length; k++) {
    if (nums[k]%2 != 0) {
      result[index] = nums[k];
      index++;
    }
  }
  return result;
}
```

This is slightly more difficult version of the famous FizzBuzz problem which is sometimes given as a first problem for job interviews. (See also: FizzBuzz Code.) Consider the series of numbers beginning at **start**and running up to but not including **end**, so for example start=1 and end=5 gives the series 1, 2, 3, 4. Return a new String[] array containing the string form of these numbers, except for multiples of 3, use "Fizz" instead of the number, for multiples of 5 use "Buzz", and for multiples of both 3 and 5 use "FizzBuzz". In Java, String.valueOf(xxx) will make the String form of an int or other type. This version is a little more complicated than the usual version since you have to allocate and index into an array instead of just printing, and we vary the start/end instead of just always doing 1..100.

fizzBuzz(1, 6) → ["1", "2", "Fizz", "4", "Buzz"]
fizzBuzz(1, 8) → ["1", "2", "Fizz", "4", "Buzz", "Fizz", "7"]
fizzBuzz(1, 11) → ["1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz"]

```
public String[] fizzBuzz(int start, int end) {
  String[] result = new String[end-start];
  // Need an index!
  int index = 0;
  for (int i = start; i < end; i++) {
    if (i%3 == 0 && i%5 == 0) {
      result[index] = "FizzBuzz";
    } else if (i%3 == 0) {
      result[index] = "Fizz";
    } else if (i%5 == 0) {
      result[index] = "Buzz";
    } else {
      result[index] = String.valueOf(i);
    }
    index++;
  }
  return result;
}
```

Array-3

Consider the leftmost and righmost appearances of some value in an array. We'll say that the "span" is the number of elements between the two inclusive. A single value has a span of 1. Returns the largest span found in the given array. (Efficiency is not a priority.)

maxSpan([1, 2, 1, 1, 3]) → 4
maxSpan([1, 4, 2, 1, 4, 1, 4]) → 6
maxSpan([1, 4, 2, 1, 4, 4, 4]) → 6

```
public int maxSpan(int[] nums) {
  if (nums.length < 1)   return 0;
  int max = 1;
  int curr = 1;
  for (int i = 0; i < nums.length; i++) {
    for (int k = nums.length-1; k > i; k--) {
      if (nums[k] == nums[i]) {
        curr = k - i + 1;
        if (curr >= max) {
          max = curr;
        }
      }
    }
  }
  return max;
}
```

Return an array that contains exactly the same numbers as the given array, but rearranged so that every 3 is immediately followed by a 4. Do not move the 3's, but every other number may move. The array contains the same number of 3's and 4's, every 3 has a number after it that is not a 3 or 4, and a 3 appears in the array before any 4.

fix34([1, 3, 1, 4]) → [1, 3, 4, 1]
fix34([1, 3, 1, 4, 4, 3, 1]) → [1, 3, 4, 1, 1, 3, 4]
fix34([3, 2, 2, 4]) → [3, 4, 2, 2]

```
public int[] fix34(int[] nums) {
  int[] myArr = nums;
  int index = 0;
  for (int y = 0; y < nums.length; y++) {
    if (nums[y] != 3 && nums[y] != 4) {
      myArr[index] = nums[y];
      index++;
    } else if (nums[y] == 3) {
      if (y != nums.length-2) {
        for (int i = y; i < nums.length; i++) {
          if (nums[i] == 4) {
            myArr[index] = 3;
            int temp = myArr[index+1];
            myArr[index+1] = 4;
            myArr[i] = temp;
            index += 2;
            break;
          }
        }
      } else {
        int temp = myArr[y+1];
        myArr[y+1] = 4;
        myArr[y-1] = temp;
      }
    }
  }
  return myArr;
}
```

(This is a slightly harder version of the fix34 problem.) Return an array that contains exactly the same numbers as the given array, but rearranged so that every 4 is immediately followed by a 5. Do not move the 4's, but every other number may move. The array contains the same number of 4's and 5's, and every 4 has a number after it that is not a 4. In this version, 5's may appear anywhere in the original array.

fix45([5, 4, 9, 4, 9, 5]) → [9, 4, 5, 4, 5, 9]
fix45([1, 4, 1, 5]) → [1, 4, 5, 1]
fix45([1, 4, 1, 5, 5, 4, 1]) → [1, 4, 5, 1, 1, 4, 5]

```
// Got it almost all right on the first try!
public int[] fix45(int[] nums) {
  int five = nextFive(nums, 0); // index of next five
  int[] myArr = nums;

  // Checking for 545
  boolean fucked = false;
  if (nums.length > 2 && five < nums.length/2 && nums[five+2] == 5)
fucked = true;

  for (int i = 0; i < nums.length-1; i++) {
    if (myArr[i] == 4) {
      int temp = myArr[i+1];
      myArr[i+1] = myArr[five]; // = 5
      myArr[five] = temp;
      // Note: nums and myArr seem interchangeable in the next if statement
      if (i > 0 && five == i-1 && (!(nums[i-1] == 5 && nums[i+1] == 5))) {
        five = nextFive(nums, i+2);
      } else {
        five = nextFive(myArr, five+1); // NOT nextFive(myArr,
five+1)......Actually....I think I need both!
      }
    }
  }

  // Correcting for 545
  if (fucked) {
    int myFive = nextFive(nums, 0);
    int temp = myArr[myFive+2];
    myArr[myFive] = temp;
    myArr[myFive+2] = 5;
  }
  return myArr;
}

public int nextFive(int[] five, int start) {
  for (int i = start; i < five.length; i++) {
    if (five[i] == 5)   return i;
  }
  return -2;
}
```

Given a non-empty array, return true if there is a place to split the array so that the sum of the numbers on one side is equal to the sum of the numbers on the other side.

canBalance([1, 1, 1, 2, 1]) → true
canBalance([2, 1, 1, 2, 1]) → false
canBalance([10, 10]) → true

```
/**
 * First Time Compile, Baby!!!!!!!!
 */
public boolean canBalance(int[] nums) {
  int sum1 = 0, sum2 = 0;
  for (int i = 0; i < nums.length; i++) {
    for (int k = 0; k < i+1; k++) {
      sum1 += nums[k];
    }
    for (int h = i+1; h < nums.length; h++) {
      sum2 += nums[h];
    }
    if (sum1 == sum2)  return true;
    sum1 = 0;
    sum2 = 0;
  }
  return false;
}
```

Given two arrays of ints sorted in increasing order, **outer** and **inner**, return true if all of the numbers in inner appear in outer. The best solution makes only a single "linear" pass of both arrays, taking advantage of the fact that both arrays are already in sorted order.

linearIn([1, 2, 4, 6], [2, 4]) → true
linearIn([1, 2, 4, 6], [2, 3, 4]) → false
linearIn([1, 2, 4, 4, 6], [2, 4]) → true

```
public boolean linearIn(int[] outer, int[] inner) {
  //if (inner.length < 1) return true; // Don't need this line if temp is
instantiated to true
  boolean temp = true;
  for (int i : inner) {
    temp = false;
    for (int k : outer) {
      if (i == k) {
        temp = true;
        break;
      }
    }
    if (!temp) return false; // outer array didn't have inner element
  }
  return temp;
}
```

Given n>=0, create an array length n*n with the following pattern, shown here for n=3 : {0, 0, 1,   0, 2, 1,   3, 2, 1} (spaces added to show the 3 groups).

squareUp(3) → [0, 0, 1, 0, 2, 1, 3, 2, 1]
squareUp(2) → [0, 1, 2, 1]
squareUp(4) → [0, 0, 0, 1, 0, 0, 2, 1, 0, 3, 2, 1, 4, 3, 2, 1]

```
public int[] squareUp(int n) {
  int[] result = new int[n*n];
  int index = result.length-1;
  int count = 0;
  // These for loops are so sweet
  for (int k = n; k >= 0; k--) {
    for (int i = 1; i <= k; i++) {
      result[index] = i;
      index--;
    }
    index -= count;
    count++;
  }
  return result;
}
```

Given n>=0, create an array with the pattern {1,   1, 2,   1, 2, 3,   ... 1, 2, 3 .. n} (spaces added to show the grouping). Note that the length of the array will be 1 + 2 + 3 ... + n, which is known to sum to exactly n*(n + 1)/2.

seriesUp(3) → [1, 1, 2, 1, 2, 3]
seriesUp(4) → [1, 1, 2, 1, 2, 3, 1, 2, 3, 4]
seriesUp(2) → [1, 1, 2]

```
public int[] seriesUp(int n) {
  int[] result = new int[n*(n+1)/2];
  int index = 0;
  for (int i = 1; i <= n; i++) {
    for (int k = 1; k <= i; k++) {
      result[index] = k;
      index++;
    }
  }
  return result;
}
```

We'll say that a "mirror" section in an array is a group of contiguous elements such that somewhere in the array, the same group appears in reverse order. For example, the largest mirror section in {1, 2, 3, 8, 9, 3, 2, 1} is length 3 (the {1, 2, 3} part). Return the size of the largest mirror section found in the given array.

maxMirror([1, 2, 3, 8, 9, 3, 2, 1]) → 3
maxMirror([1, 2, 1, 4]) → 3
maxMirror([7, 1, 2, 9, 7, 2, 1]) → 2

```
public int maxMirror(int[] nums) {
  int[] rev = rev(nums);
  /**
   * It works without this!
   *
  if (Arrays.equals(nums, rev)) {
    return nums.length;
  }
  */
  // Ohhhhhhhhhh! Just reverse the array and find the longest similar run
between them!
  // Start at index 0, then 1, then 2, etc...
  // Then increase length by 1, 2, 3, etc...

  /**
   * Below was a great idea, but it only works for runs at the same relative
index
   *
  // This is a lot like RandomIntegers!
  // Imported code below
  int currLen = 0;
  int maxLen = 0;

  for (int j = 0; j < nums.length-1; j++) {
    if (nums[j] == rev[j]) {
      currLen++;
      if (currLen >= maxLen) { // The >= accounts for same length runs
        maxLen = currLen;
      }
    } else {
      currLen = 0;
    }
  }
  return maxLen+1;
}
*/

  // I think at least two for loops are needed -- No, four for loops! -- No, six!
  int maxLen = 0;
  int[] temp1, temp2; // have to use Arrays.equals(int[] x, int[] y)
  for (int i = 0; i < nums.length; i++) { // starting index
    for (int k = i; k < nums.length; k++) {
      temp1 = new int[k-i+1];
      int index1 = 0;
      for (int j = i; j <= k; j++) {
        temp1[index1] = nums[j];
        index1++;
      }
      // At this point, I have captured every run in the original array, as an
array!
      // Now, we do the same thing for the reversed array!
      for (int a = 0; a < rev.length; a++) {
        for (int b = a; b < rev.length; b++) {
          temp2 = new int[b-a+1];
          int index2 = 0;
          for (int c = a; c <= b; c++) {
            temp2[index2] = rev[c];
            index2++;
          }
          // Now we have two separate arrays we can compare
          if (Arrays.equals(temp1, temp2) && temp1.length >= maxLen) {
            maxLen = temp1.length;
          }
          // Note: This algorithm is not the most efficient as it wastes a lot of
time (relatively) in comparing arrays of different length (obviously not
equal). This could be smoothened out by making temp2 only the length of
temp1 and filling it as such.
        }
      }
    }
  }
  return maxLen;
}

public int[] rev(int[] arr) {
  int[] result = new int[arr.length];
  int index = result.length-1;
  for (int i : arr) {
    result[index] = i;
    index--;
  }
  return result;
}
```

Say that a "clump" in an array is a series of 2 or more adjacent elements of the same value. Return the number of clumps in the given array.

countClumps([1, 2, 2, 3, 4, 4]) → 2
countClumps([1, 1, 2, 1, 1]) → 2
countClumps([1, 1, 1, 1, 1]) → 1

```
public int countClumps(int[] nums) {
  int count = 0;
  int currLen = 1;

  for (int j = 0; j < nums.length-1; j++) {
    if (nums[j] == nums[j+1]) {
      currLen++;
      if (currLen < 3) {
        count++;
      }
    } else {
      currLen = 1; // Be sure to reset value back to original (1). It's not
always 0! (No factorial joke intended)
    }
  }
  return count;
}
```

ArrayList-1

(From http://codingbat.com/home/tlsmith2@wsfcs.k12.nc.us)

Return the sum of all the numbers at even positions in the array list.

sumEvenIndexes([1, 2, 3, 4]) → 4
sumEvenIndexes([8, 16, 10]) → 18
sumEvenIndexes([5, 0, -10]) → -5

```
public int sumEvenIndexesB(ArrayList<Integer> numsList) {
  int sum = 0;
  for (int i = 0; i < numsList.size(); i+=2) {
    sum += numsList.get(i);
  }
  return sum;
}
```

Return the count of all the even numbers in the array list. Hint: The % operator may be useful.

evenCount([1, 2, 3, 4]) → 2
evenCount([8, 16, 10]) → 3
evenCount([5, 0, 22]) → 2

```
public int evenCountB(ArrayList<Integer> numsList) {
  int count = 0;
  for (int i : numsList) {
    if (i % 2 == 0) {
      count ++;
    }
  }
  return count;
}
```

Return the sum of the numbers in the array list that are odd and have an even index, or the number is even and its index is odd.

fairlyOddNums([1, 2, 3, 4]) → 10

fairlyOddNums([8, 16, 10]) → 16
fairlyOddNums([5, 0, 13, 51]) → 18

```
public int fairlyOddNumsB(ArrayList<Integer> numsList) {
  int sum = 0;
  for (int i = 0; i < numsList.size(); i+=2) { // Even index (odd num)
    if (numsList.get(i)%2 != 0) {
      sum += numsList.get(i);
    }
  }
  for (int k = 1; k < numsList.size(); k+=2) { // Odd index (even num)
    if (numsList.get(k)%2 == 0) {
      sum += numsList.get(k);
    }
  }
  return sum;
}
```

Return the sum of the numbers in the array list. Except the number 7 is yucky, so it does not count and numbers that come immediately after a 7 also do not count.

yucky7([1, 4]) → 5
yucky7([1, 2, 7, 3]) → 3
yucky7([1, 7, 8, 5, 7]) → 6

```
public int yucky7b(ArrayList<Integer> numsList) {
  int sum = 0;
  for (int i = 0; i < numsList.size(); i++) {
    if (numsList.get(i) != 7) {
      sum += numsList.get(i);
    } else {
      i++;
      for (int k = i; k < numsList.size(); k++) {
        if (numsList.get(k) == 7) {
          i++;
        } else {
          break;
        }
      }
    }
  }
  return sum;
}
```

An array list contains a list of animals. If the animal is a cat (i.e. the animal's descrition contains the word "cat" or "Cat"), then add it to a new array list. Return the new array list of cats.

catty(["buffalo", "dog"]) → []
catty(["bobcat", "siamese cat", "catbird"]) → ["bobcat", "siamese cat", "catbird"]
catty(["Cat", "frog", "mouse"]) → ["Cat"]

```
public ArrayList<String> cattyB(ArrayList<String> animalsList) {
  ArrayList<String> cats = new ArrayList<String>();
  for (String s : animalsList) {
    if (s.indexOf("cat") >= 0 || s.indexOf("Cat") >= 0) {
      cats.add(s);
    }
  }
  return cats;
}
```

An array list contains the number of votes in each district for a given political candidate named Jones. There are only two candidates in the race and his opponent is named Sims. Positive numbers indicate that Jones won by that number of votes and negative numbers indicate that Sims won by that number of votes. The winner of the race is the candidate that has the most total votes from all the districts. Jones wins if there is a tie since he is the incumbent. Return true if Jones won and false if Sims won.

winRace([20, -15]) → true
winRace([20, 200, -300, 10, -10]) → false
winRace([-25, 20, 3, 2]) → true

```
public boolean winRaceB(ArrayList<Integer> votesList) {
  int sumPos = 0, sumNeg = 0;
  for (int i : votesList) {
    if (i >= 0) {
      sumPos += i;
    } else {
      sumNeg += i;
    }
  }
  return sumPos >= Math.abs(sumNeg);
}
```

An array list contains a list of pets that are following each other to a big pet party. The pet at position 0 is following the pet at position 1. If the pet is a cat that is being followed by a dog, it gets scared and runs away, so you should remove it from the array list. Return the array list of pets.

```
scaredyCat(["cat", "dog", "guinea pig"]) → ["cat", "dog", "guinea pig"]
scaredyCat(["cat", "frog", "cat", "dog", "cat"]) → ["cat", "frog", "cat", "dog"]
scaredyCat(["cat", "dog", "cat", "cat", "snake"]) → ["cat", "dog", "snake"]
```

```
public ArrayList<String> scaredyCatB(ArrayList<String> petsList) {
  for (int i = 0; i < petsList.size()-1; i++) {
    if (petsList.get(i).equals("dog") && petsList.get(i+1).equals("cat")) {
      petsList.remove(i+1);
      i--;  // If two cats are in a row
    }
  }
  return petsList;
}
```

A field is considered level if all of its elevation readings are within 8 inches of each other. Given a list of elevation readings of the field in inches, return true or false to indicate whether the field is level. If there are less than 3 readings, there is not enough data and you should return false in that case.

```
levelField([11, 10, 14, 8]) → true
levelField([11, 10, 14, 8, 10, 17]) → false
levelField([4, 6]) → false
```

```
public boolean levelFieldB(ArrayList<Integer> numsList) {
  if (numsList.size() < 3)  return false;
  int min = Integer.MAX_VALUE;
  int max = Integer.MIN_VALUE;
  for (int n : numsList) {
    if (n < min)  min = n;
    if (n > max)  max = n;
  }
  return (max - min) <= 8;
}
```

An array list contains the ages of married couples in a village. The ages are listed in pairs, so if there are two couples, ages 50, 52 and 35, 30 they would be listed in the array list as {50, 52, 35, 30}. The village considers the wisest couple to be the one with the highest combined age. Return the combined age of the wisest couple. There will be an even number of elements in the array list.

wiseGuys([20, 18]) → 38
wiseGuys([30, 33, 40, 21]) → 63
wiseGuys([25, 25, 30, 29, 60, 62]) → 122

```
public int wiseGuysB(ArrayList<Integer> agesList) {
  int max = 0;  // not Integer.MIN_VALUE;
  for (int i = 0; i < agesList.size(); i+=2) {
    int temp = agesList.get(i) + agesList.get(i+1);
    if (temp > max)  max = temp;
  }
  return max;
}
```

Three brother are very competitive and have a small wager on who is the best golfer this month. They have recorded their scores for the month in separate array lists. Return the name of the brother who has the lowest average for the month. If a brother has less than two scores, he does not have enough scores to qualify as the best golfer. At least one of the brothers will have enough scores to qualify.

golfingBrothers([70, 75, 80], [80, 82, 85], [90, 91, 92]) → "adam"
golfingBrothers([100, 75], [80, 82, 85, 86], [90, 91, 92]) → "bob"

```
golfingBrothers([70], [80, 82], [90, 91, 92]) → "bob"

public String golfingBrothersB(ArrayList<Integer> adamList,
ArrayList<Integer> bobList, ArrayList<Integer> craigList) {
  String lowest = "";
  double lowestAve = Double.MAX_VALUE;
  double sum = 0;
  double tempAve = 0;  // need double!
  for (int a : adamList) {
    sum += a;
  }
  if (adamList.size() > 1) {
    tempAve = sum/adamList.size();
  }
  if (tempAve < lowestAve && tempAve > 0) {
    lowestAve = tempAve;
    lowest = "adam";
  }

  sum = 0;
  tempAve = 0;
  for (int b : bobList) {
    sum += b;
  }
  if (bobList.size() > 1) {
    tempAve = sum/bobList.size();
  }
  if (tempAve < lowestAve && tempAve > 0) {
    lowestAve = tempAve;
    lowest = "bob";
  }

  sum = 0;
  tempAve = 0;
  for (int c : craigList) {
    sum += c;
  }
  if (craigList.size() > 1) {
    tempAve = sum/craigList.size();
  }
  if (tempAve < lowestAve && tempAve > 0) {
    lowestAve = tempAve;
    lowest = "craig";
  }
  return lowest;
}
```

You are having dinner with your significant other and you would like to have soup if it is free. The restaurant offers free soup if the total purchase of both of your entrees is at least $20 and each of you orders at least $5 worth of desserts or appetizers. Each array list identifies the amount of money each of you spends on the appetizer, entree, and dessert (in that order). Return true if you get free soup. Else if no soup for you, then return false.

freeSoup([6, 11, 0], [0, 16, 5]) → true
freeSoup([6, 8, 0], [0, 16, 5]) → true
freeSoup([4, 12, 0], [5, 16, 5]) → false

```
public boolean freeSoupB(ArrayList<Integer> aList, ArrayList<Integer>
bList) {
  if ((aList.get(0) + aList.get(2) < 5) || (bList.get(0) + bList.get(2) < 5)) {
    return false;
  } else if (aList.get(1) + bList.get(1) < 20) {
    return false;
  }
  return true;
}
```

An array list contains a list of words. Take the first character of the first word and concatenate it as the first character of each word and take the last character of the last word and concatenate it as the last character of each word. Return the modified array list of words.

wackyWords(["buffalo", "dog"]) → ["bbuffalog", "bdogg"]
wackyWords(["football", "player", "silly"]) → ["ffootbally", "fplayery", "fsillyy"]
wackyWords(["great"]) → ["ggreatt"]

```
public ArrayList<String> wackyWordsB(ArrayList<String> wordsList) {
  if (wordsList.size() < 1)  return wordsList;
  String chr1 = wordsList.get(0).substring(0, 1);
  String chr2 =
wordsList.get(wordsList.size()-1).substring(wordsList.get(wordsList.size()-1)
.length()-1);
  // Note: Simply changing the words in wordsList (look below) doesn't work.
I needed to create a new ArrayList
  /**
  for (String word : wordsList) {
     word = chr1 + word + chr2;
  }
  */
  ArrayList<String> newWords = new ArrayList<String>();
  for (String word : wordsList) {
    newWords.add(chr1 + word + chr2);
  }
  return newWords;
}
```

A teacher gives her students extra credit if they can guess the class average for a given test and their guess is within 5% of the average. Given a list of grades and the student's guess, return true if the student receives extra credit and false if they don't receive extra credit.

guessAverage([90, 80, 85], 85) → true
guessAverage([88, 78, 92], 92) → false
guessAverage([80], 75) → false

```
public boolean guessAverageB(ArrayList<Integer> numsList, int guess) {
  double sum = 0;
  for (int num : numsList) {
    sum += num;
  }
  double ave = sum/numsList.size();
  return (guess <= (ave+ave*0.05) && guess >= (ave-ave*0.05));
  // Another way to test is: (Math.abs(avg-guess)/avg <= .05)
}
```

An array list contains the high school class ranks of my students. If at least two of the students are in the top 10 of their class, order the array list elements from highest rank to lowest rank. Otherwise, order the array list elements from lowest rank to highest rank. Return the array list of class ranks. Hint: Don't forget about the methods in the Collections class.

classRanks([40, 25, 12, 35, 125, 300, 63]) → [300, 125, 63, 40, 35, 25, 12]
classRanks([35, 72, 5, 22, 78, 3]) → [3, 5, 22, 35, 72, 78]
classRanks([2, 56, 24, 100]) → [100, 56, 24, 2]

```
public ArrayList<Integer> classRanksB(ArrayList<Integer> rankings) {
  Collections.sort(rankings);
  int top10 = 0;
  for (int rank : rankings) {
    if (rank <= 10) {
      top10++;
    }
  }
  if (top10 < 2) {
    Collections.reverse(rankings);
  }
  return rankings;
}
```

Given a string, return an array list that contains each character of the string in backwards order.

backwardChars("Career Center") → ["r", "e", "t", "n", "e", "C", " ", "r", "e", "e", "r", "a", "C"]
backwardChars("football") → ["l", "l", "a", "b", "t", "o", "o", "f"]
backwardChars("?") → ["?"]

```
public ArrayList<String> backwardCharsB(String str) {
  ArrayList<String> letters = new ArrayList<String>();
  for (int i = str.length(); i > 0; i--) {
    letters.add(str.substring(i-1, i));
  }
  return letters;
}
```

AP-1

Given an array of scores, return true if each score is equal or greater than the one before. The array will be length 2 or more.

scoresIncreasing([1, 3, 4]) → true
scoresIncreasing([1, 3, 2]) → false
scoresIncreasing([1, 1, 4]) → true

```
public boolean scoresIncreasing(int[] scores) {
  for (int i = 1; i < scores.length; i++) {
     if (scores[i] < scores[i-1]) {
        return false;
     }
  }
  return true;
}
```

Given an array of scores, return true if there are scores of 100 next to each other in the array. The array length will be at least 2.

scores100([1, 100, 100]) → true
scores100([1, 100, 99, 100]) → false
scores100([100, 1, 100, 100]) → true

```
public boolean scores100(int[] scores) {
  for (int i = 1; i < scores.length; i++) {
     if (scores[i] == scores [i-1] && scores[i] == 100) {
        return true;
     }
  }
  return false;
}
```

Given an array of scores sorted in increasing order, return true if the array contains 3 adjacent scores that differ from each other by at most 2, such as with {3, 4, 5} or {3, 5, 5}.

scoresClump([3, 4, 5]) → true
scoresClump([3, 4, 6]) → false
scoresClump([1, 3, 5, 5]) → true

```
public boolean scoresClump(int[] scores) {
  if (scores.length < 3) {
    return false;
  }
  for (int i = 2; i < scores.length; i++) {
    if (scores[i]-scores[i-1] < 3 && scores[i-1]-scores[i-2] < 3 &&
scores[i]-scores[i-2] < 3) {
       return true;
    }
  }
  return false;
}
```

Given an array of scores, compute the int average of the first half and the second half, and return whichever is larger. We'll say that the second half begins at index length/2. The array length will be at least 2. To practice decomposition, write a separate helper method <br>int average(int[] scores, int start, int end) { which computes the average of the elements between indexes start..end. Call your helper method twice to implement scoresAverage(). Write your helper method after your scoresAverage() method in the JavaBat text area. Normally you would compute averages with doubles, but here we use ints so the expected results are exact.

scoresAverage([2, 2, 4, 4]) → 4
scoresAverage([4, 4, 4, 2, 2, 2]) → 4
scoresAverage([3, 4, 5, 1, 2, 3]) → 4

```
public int scoresAverage(int[] scores) {
  int avgOne, avgTwo;
  int[] one = new int[scores.length/2];
  for (int i = 0; i < one.length; i++) {
    one[i] = scores[i];
  }
  int[] two = new int[scores.length - one.length];
  for (int k = scores.length-1; k > two.length - 1; k--) {
    two[k-two.length] = scores[k];
  }
  avgOne = average(one);
  avgTwo = average(two);
  if (avgOne >= avgTwo) {
    return avgOne;
  }
  return avgTwo;
}

public int average(int[] scores) {
  int sum = 0;
  for (int i : scores) {
    sum += i;
  }
  return sum/scores.length;
}
```

Given an array of strings, return the count of the number of strings with the given length.

wordsCount(["a", "bb", "b", "ccc"], 1) → 2
wordsCount(["a", "bb", "b", "ccc"], 3) → 1
wordsCount(["a", "bb", "b", "ccc"], 4) → 0

```
public int wordsCount(String[] words, int len) {
  int count = 0;
  for (String s : words) {
    if (s.length() == len) {
      count++;
    }
  }
  return count;
}
```

Given an array of strings, return a new array containing the first N strings. N will be in the range 1..length.

wordsFront(["a", "b", "c", "d"], 1) → ["a"]

wordsFront(["a", "b", "c", "d"], 2) → ["a", "b"]
wordsFront(["a", "b", "c", "d"], 3) → ["a", "b", "c"]

```
public String[] wordsFront(String[] words, int n) {
  String[] result = new String[n];
  for (int i = 0; i < result.length; i++) {
    result[i] = words[i];
  }
  return result;
}
```

Given an array of strings, return a new List (e.g. an ArrayList) where all the strings of the given length are omitted. See wordsWithout() below which is more difficult because it uses arrays.

wordsWithoutList(["a", "bb", "b", "ccc"], 1) → ["bb", "ccc"]
wordsWithoutList(["a", "bb", "b", "ccc"], 3) → ["a", "bb", "b"]
wordsWithoutList(["a", "bb", "b", "ccc"], 4) → ["a", "bb", "b", "ccc"]

```
public List wordsWithoutList(String[] words, int len) {
  ArrayList<String> result = new ArrayList<String>();
  for (int i = 0; i < words.length; i++) {
    result.add(words[i]);
  }
  for (int k = result.size()-1; k >= 0; k--) {
    if (result.get(k).length() == len) {
      result.remove(k);
    }
  }
  return result;
}
```

Given a positive int n, return true if it contains a 1 digit. Note: use % to get the rightmost digit, and / to discard the rightmost digit.

hasOne(10) → true
hasOne(22) → false
hasOne(220) → false

```
public boolean hasOne(int n) {
  if (n % 10 == 1) {
    return true;
  } else if (n == 0) {
    return false;
  } else {
    return hasOne(n/10);
  }
}
```

We'll say that a positive int divides itself if every digit in the number divides into the number evenly. So for example 128 divides itself since 1, 2, and 8 all divide into 128 evenly. We'll say that 0 does not divide into anything evenly, so no number with a 0 digit divides itself. Note: use % to get the rightmost digit, and / to discard the rightmost digit.

dividesSelf(128) → true
dividesSelf(12) → true
dividesSelf(120) → false

```
public boolean dividesSelf(int n) {
  int num = n;
  while (num > 0) {
   int digit = num%10;
   if (digit == 0 || n % digit != 0) {
     return false;
   }
   num /= 10;
  }
  return true;

  /**
   * First Attempt
  int len = 1;
  int num = n;
  while (num > 9) {
    num /= 10;
    len++;
  }
  int num3 = n; // changing
  int myMod = num3%10; // mod
  while (len > 0) {
    if (myMod == 0) {
      return false;
    }
    if (n % (myMod) != 0) {
      return false;
    }
    num3 /= 10;
    myMod = num3%10;
    len--;
  }
  return true;
  */
}
```

Given an array of positive ints, return a new array of length "count" containing the first even numbers from the original array. The original array will contain at least "count" even numbers.

copyEvens([3, 2, 4, 5, 8], 2) → [2, 4]
copyEvens([3, 2, 4, 5, 8], 3) → [2, 4, 8]
copyEvens([6, 1, 2, 4, 5, 8], 3) → [6, 2, 4]

```
public int[] copyEvens(int[] nums, int count) {
  int[] evens = new int[count];
  int evensIndex = 0;
  for (int i = 0; i < nums.length; i++) {
    if (evensIndex >= evens.length) {
      break;
    }
    if (nums[i] % 2 == 0) {
      evens[evensIndex] = nums[i];
      evensIndex++;
    }
  }
  return evens;
}
```

We'll say that a positive int n is "endy" if it is in the range 0..10 or 90..100 (inclusive). Given an array of positive ints, return a new array of length "count" containing the first endy numbers from the original array. Decompose out a separate isEndy(int n) method to test if a number is endy. The original array will contain at least "count" endy numbers.

copyEndy([9, 11, 90, 22, 6], 2) → [9, 90]
copyEndy([9, 11, 90, 22, 6], 3) → [9, 90, 6]
copyEndy([12, 1, 1, 13, 0, 20], 2) → [1, 1]

```
public int[] copyEndy(int[] nums, int count) {
  int[] ends = new int[count];
  int endsIndex = 0;
  for (int i = 0; i < nums.length; i++) {
    if (endsIndex >= ends.length) {
      break;
    }
    if (isEndy(nums[i])) {
      ends[endsIndex] = nums[i];
      endsIndex++;
    }
  }
  return ends;
}

public boolean isEndy(int n) {
  return ((0 <= n && n <= 10) || (90 <= n && n <= 100));
}
```

Given 2 arrays that are the same length containing strings, compare the 1st string in one array to the 1st string in the other array, the 2nd to the 2nd and so on. Count the number of times that the 2 strings are non-empty and start with the same char. The strings may be any length, including 0.

matchUp(["aa", "bb", "cc"], ["aaa", "xx", "bb"]) → 1
matchUp(["aa", "bb", "cc"], ["aaa", "b", "bb"]) → 2
matchUp(["aa", "bb", "cc"], ["", "", "ccc"]) → 1

```
public int matchUp(String[] a, String[] b) {
  int count = 0;
  for (int i = 0; i < a.length; i++) {
    if (a[i].length() != 0 && b[i].length() != 0 && (a[i].charAt(0) ==
b[i].charAt(0))) {
      count++;
    }
  }
  return count;
}
```

The "key" array is an array containing the correct answers to an exam, like {"a", "a", "b", "b"}. the "answers" array contains a student's answers, with "?" representing a question left blank. The two arrays are not empty and are the same length. Return the score for this array of answers, giving +4 for each correct answer, -1 for each incorrect answer, and +0 for each blank answer.

```
scoreUp(["a", "a", "b", "b"], ["a", "c", "b", "c"]) → 6
scoreUp(["a", "a", "b", "b"], ["a", "a", "b", "c"]) → 11
scoreUp(["a", "a", "b", "b"], ["a", "a", "b", "b"]) → 16
```

```
public int scoreUp(String[] key, String[] answers) {
  int score = 0;
  for (int i = 0; i < key.length; i++) {
    if (key[i].equals(answers[i])) {
      score += 4;
    } else if (!answers[i].equals("?")) {
      score--;
    }
  }
  return score;
}
```

Given an array of strings, return a new array without the strings that are equal to the target string. One approach is to count the occurrences of the target string, make a new array of the correct length, and then copy over the correct strings.

```
wordsWithout(["a", "b", "c", "a"], "a") → ["b", "c"]
wordsWithout(["a", "b", "c", "a"], "b") → ["a", "c", "a"]
wordsWithout(["a", "b", "c", "a"], "c") → ["a", "b", "a"]

public String[] wordsWithout(String[] words, String target) {
  int targetCount = 0;
  for (String s : words) {
    if (s.equals(target)) {
      targetCount++;
    }
  }
  String[] result = new String[words.length - targetCount];
  int resultIndex = 0;
  for (String e : words) {
    if (!e.equals(target)) {
      result[resultIndex] = e;
      resultIndex++;
    }
  }
  return result;
}
```

Given two arrays, A and B, of non-negative int scores. A "special" score is one which is a multiple of 10, such as 40 or 90. Return the sum of largest special score in A and the largest special score in B. To practice decomposition, write a separate helper method which finds the largest special score in an array. Write your helper method after your scoresSpecial() method in the JavaBat text area.

```
scoresSpecial([12, 10, 4], [2, 20, 30]) → 40
scoresSpecial([20, 10, 4], [2, 20, 10]) → 40
scoresSpecial([12, 11, 4], [2, 20, 31]) → 20

public int scoresSpecial(int[] a, int[] b) {
  int aSpesh = singleSpecial(a);
  int bSpesh = singleSpecial(b);
  return aSpesh + bSpesh;
}
```

```
public int singleSpecial(int[] c) {
  int spesh = 0;
  for (int i : c) {
    if (i % 10 == 0 && i > spesh) {
      spesh = i;
    }
  }
  return spesh; // Return 0 if not spesh
}
```

We have an array of heights, representing the altitude along a walking trail. Given start/end indexes into the array, return the sum of the changes for a walk beginning at the start index and ending at the end index. For example, with the heights {5, 3, 6, 7, 2} and start=2, end=4 yields a sum of 1 + 5 = 6. The start end end index will both be valid indexes into the array with start <= end.

sumHeights([5, 3, 6, 7, 2], 2, 4) → 6
sumHeights([5, 3, 6, 7, 2], 0, 1) → 2
sumHeights([5, 3, 6, 7, 2], 0, 4) → 11

```
public int sumHeights(int[] heights, int start, int end) {
  int sum = 0;
  for (int i = start; i < end; i++) {
    sum += Math.abs(heights[i]-heights[i+1]);
  }
  return sum;
}
```

(A variation on the sumHeights problem.) We have an array of heights, representing the altitude along a walking trail. Given start/end indexes into the array, return the sum of the changes for a walk beginning at the start index and ending at the end index, however increases in height count double. For example, with the heights {5, 3, 6, 7, 2} and start=2, end=4 yields a sum of 1*2 + 5 = 7. The start end end index will both be valid indexes into the array with start <= end.

sumHeights2([5, 3, 6, 7, 2], 2, 4) → 7
sumHeights2([5, 3, 6, 7, 2], 0, 1) → 2

sumHeights2([5, 3, 6, 7, 2], 0, 4) → 15

```
public int sumHeights2(int[] heights, int start, int end) {
  int sum = 0;
  for (int i = start; i < end; i++) {
    int tempSum = (heights[i+1] - heights[i] > 0) ? 2*(heights[i+1] -
heights[i]) : Math.abs(heights[i+1] - heights[i]);
    sum += tempSum;
  }
  return sum;
}
```

(A variation on the sumHeights problem.) We have an array of heights, representing the altitude along a walking trail. Given start/end indexes into the array, return the number of "big" steps for a walk starting at the start index and ending at the end index. We'll say that step is big if it is 5 or more up or down. The start end end index will both be valid indexes into the array with start <= end.

bigHeights([5, 3, 6, 7, 2], 2, 4) → 1
bigHeights([5, 3, 6, 7, 2], 0, 1) → 0
bigHeights([5, 3, 6, 7, 2], 0, 4) → 1

```
public int bigHeights(int[] heights, int start, int end) {
  int count = 0;
  for (int i = start; i < end; i++) {
    if (Math.abs(heights[i+1] - heights[i]) >= 5) {
      count++;
    }
  }
  return count;
}
```

We have data for two users, A and B, each with a String name and an int id. The goal is to order the users such as for sorting. Return -1 if A comes before B, 1 if A comes after B, and 0 if they are the same. Order first by the string names, and then by the id numbers if the names are the same. Note: with Strings str1.compareTo(str2) returns an int value which is negative/0/positive to indicate how str1 is ordered to str2 (the value is not limited to -1/0/1). (On the AP, there would be two User objects, but here the code simply takes the two strings and two ints directly. The code logic is the same.)

userCompare("bb", 1, "zz", 2) → -1
userCompare("bb", 1, "aa", 2) → 1
userCompare("bb", 1, "bb", 1) → 0

```
public int userCompare(String aName, int aId, String bName, int bId) {
  if (aName.compareTo(bName) < 0) {
    return -1;
  } else if (aName.compareTo(bName) > 0) {
    return 1;
  } else {
    if (aId - bId < 0) {
      return -1;
    } else if (aId - bId > 0) {
      return 1;
    } else {
      return 0;
    }
  }
}
```

Start with two arrays of strings, A and B, each with its elements in alphabetical order and without duplicates. Return a new array containing the first N elements from the two arrays. The result array should be in alphabetical order and without duplicates. A and B will both have a length which is N or more. The best "linear" solution makes a single pass over A and B, taking advantage of the fact that they are in alphabetical order, copying elements directly to the new array.

mergeTwo(["a", "c", "z"], ["b", "f", "z"], 3) → ["a", "b", "c"]
mergeTwo(["a", "c", "z"], ["c", "f", "z"], 3) → ["a", "c", "f"]
mergeTwo(["f", "g", "z"], ["c", "f", "g"], 3) → ["c", "f", "g"]

```
public String[] mergeTwo(String[] a, String[] b, int n) {
  String[] result = new String[n];
  int aIndex = 0;
  int bIndex = 0;
  for (int i = 0; i < result.length; i++) {
    if (a[aIndex].compareTo(b[bIndex]) < 0) {
      result[i] = a[aIndex];
      aIndex++;
    } else if (a[aIndex].compareTo(b[bIndex]) > 0) {
      result[i] = b[bIndex];
      bIndex++;
    } else {
      result[i] = a[aIndex];
      aIndex++;
      bIndex++;
    }
  }
  return result;
}
```

Start with two arrays of strings, a and b, each in alphabetical order, possibly with duplicates. Return the count of the number of strings which appear in both arrays. The best "linear" solution makes a single pass over both arrays, taking advantage of the fact that they are in alphabetical order.

commonTwo(["a", "c", "x"], ["b", "c", "d", "x"]) → 2
commonTwo(["a", "c", "x"], ["a", "b", "c", "x", "z"]) → 3
commonTwo(["a", "b", "c"], ["a", "b", "c"]) → 3

```
public int commonTwo(String[] a, String[] b) {
  ArrayList<String> common = new ArrayList<String>();
  for (String s : a) {
    for (String q : b) {
      if (s.equals(q) && !isInCommon(common, q)) {
        common.add(q);
      }
    }
  }
  return common.size();
}

public boolean isInCommon(ArrayList<String> com, String str) {
  for (String s : com) {
    if (s.equals(str)) {
      return true;
    }
  }
  return false;
}
```

Python

Warmup-1

The parameter weekday is True if it is a weekday, and the parameter vacation is True if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return True if we sleep in.

```
sleep_in(False, False) → True
sleep_in(True, False) → False
sleep_in(False, True) → True
```

```
def sleep_in(weekday, vacation):
  if (not weekday or vacation):
    return True
  return False
```

We have two monkeys, a and b, and the parameters a_smile and b_smile indicate if each is smiling. We are in trouble if they are both smiling or if neither of them is smiling. Return True if we are in trouble.

```
monkey_trouble(True, True) → True
monkey_trouble(False, False) → True
monkey_trouble(True, False) → False
```

```
def monkey_trouble(a_smile, b_smile):
  return a_smile == b_smile
```

Given two int values, return their sum. Unless the two values are the same, then return double their sum.

```
sum_double(1, 2) → 3
sum_double(3, 2) → 5
sum_double(2, 2) → 8
```

```
def sum_double(a, b):
  if a == b:
    return 4*a
  else:
    return a+b
```

Given an int n, return the absolute difference between n and 21, except return double the absolute difference if n is over 21.

diff21(19) → 2
diff21(10) → 11
diff21(21) → 0

```
def diff21(n):
  if n > 21:
    return 2*(n-21)
  else:
    return 21-n
```

We have a loud talking parrot. The "hour" parameter is the current hour time in the range 0..23. We are in trouble if the parrot is talking and the hour is before 7 or after 20. Return True if we are in trouble.

parrot_trouble(True, 6) → True
parrot_trouble(True, 7) → False
parrot_trouble(False, 6) → False

```
def parrot_trouble(talking, hour):
  if talking and (hour < 7 or hour > 20):
    return True
  return False
```

Given 2 ints, a and b, return True if one if them is 10 or if their sum is 10.

makes10(9, 10) → True
makes10(9, 9) → False
makes10(1, 9) → True

```
def makes10(a, b):
  if (a == 10 or b == 10 or a+b == 10):
    return True
  return False
```

Given an int n, return True if it is within 10 of 100 or 200. Note: abs(num) computes the absolute value of a number.

```
near_hundred(93) → True
near_hundred(90) → True
near_hundred(89) → False

def near_hundred(n):
  if (abs(100-n) <= 10 or abs(200-n) <= 10):
    return True
  return False
```

Given 2 int values, return True if one is negative and one is positive. Except if the parameter "negative" is True, then return True only if both are negative.

```
pos_neg(1, -1, False) → True
pos_neg(-1, 1, False) → True
pos_neg(-4, -5, True) → True

def pos_neg(a, b, negative):
  if (negative):
    return a < 0 and b < 0
  else:
    return (a > 0 and b < 0) or (a < 0 and b > 0)
```

Given a string, return a new string where "not " has been added to the front. However, if the string already begins with "not", return the string unchanged.

```
not_string('candy') → 'not candy'
not_string('x') → 'not x'
not_string('not bad') → 'not bad'

def not_string(str):
  if len(str) >= 3 and str[:3] == "not":
    return str
  return "not " + str
  """
  First Attempt
  if len(str) < 3:
    return "not " + str
  elif str[0:3] == 'not':
    return str
  else:
    return "not " + str
  """
```

Given a non-empty string and an int n, return a new string where the char at index n has been removed. The value of n will be a valid index of a char in the original string (i.e. n will be in the range 0..len(str)-1 inclusive).

```
missing_char('kitten', 1) → 'ktten'
missing_char('kitten', 0) → 'itten'
missing_char('kitten', 4) → 'kittn'

def missing_char(str, n):
  return str[:n] + str[n+1:]
```

Given a string, return a new string where the first and last chars have been exchanged.

```
front_back('code') → 'eodc'
front_back('a') → 'a'
front_back('ab') → 'ba'

def front_back(str):
  if len(str) < 2:
    return str
  ## return str[len(str)-1] + str[1:len(str)-1] + str[0] <-----First attempt,
which works
  ## After I had more experience with list slicing, I realized below was easier
-- 5/26/16
  return str[-1] + str[1:-1] + str[0]
```

Given a string, we'll say that the front is the first 3 chars of the string. If the string length is less than 3, the front is whatever is there. Return a new string which is 3 copies of the front.

front3('Java') → 'JavJavJav'
front3('Chocolate') → 'ChoChoCho'
front3('abc') → 'abcabcabc'

```
def front3(str):
  if len(str) < 3:
    return 3*str
  else:
    return 3*str[:3]
```

Warmup-2

Given a string and a non-negative int n, return a larger string that is n copies of the original string.

string_times('Hi', 2) → 'HiHi'
string_times('Hi', 3) → 'HiHiHi'
string_times('Hi', 1) → 'Hi'

```
def string_times(str, n):
  return n*str
```

Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;

front_times('Chocolate', 2) → 'ChoCho'
front_times('Chocolate', 3) → 'ChoChoCho'
front_times('Abc', 3) → 'AbcAbcAbc'

```
def front_times(str, n):
  if len(str) < 3:
    return n*str
  return n*str[:3]
```

Given a string, return a new string made of every other char starting with the first, so "Hello" yields "Hlo".

string_bits('Hello') → 'Hlo'
string_bits('Hi') → 'H'
string_bits('Heeololeo') → 'Hello'

```
def string_bits(str):
  result = "";
  for i in range(0, len(str), 2):
    result += str[i]
  return result
```

Given a non-empty string like "Code" return a string like "CCoCodCode".

string_splosion('Code') → 'CCoCodCode'
string_splosion('abc') → 'aababc'
string_splosion('ab') → 'aab'

```
def string_splosion(str):
  result = "";
  for i in range(len(str)+1): ## Need the +1!
    result += str[:i]
  return result
```

Given a string, return the count of the number of times that a substring length 2 appears in the string and also as the last 2 chars of the string, so "hixxxhi" yields 1 (we won't count the end substring).

last2('hixxhi') → 1
last2('xaxxaxaxx') → 1
last2('axxxaaxx') → 2

```
def last2(str):
  sub = str[-2:]  ## str[len(str)-2:] works too
  count = 0;
  for i in range(len(str)-1):
    if str[i:i+2] == sub:
      count += 1
  if count == 0:  return 0
  return count-1
```

Given an array of ints, return the number of 9's in the array.

array_count9([1, 2, 9]) → 1
array_count9([1, 9, 9]) → 2
array_count9([1, 9, 9, 3, 9]) → 3

```
def array_count9(nums):
  count = 0
  for num in nums:
    if num == 9:
```

```
    count += 1
  return count
```

Given an array of ints, return True if one of the first 4 elements in the array is a 9. The array length may be less than 4.

```
array_front9([1, 2, 9, 3, 4]) → True
array_front9([1, 2, 3, 4, 9]) → False
array_front9([1, 2, 3, 4, 5]) → False

def array_front9(nums):
  for i in range(len(nums)):
    if i < 4 and nums[i] == 9:
      return True
  return False
```

Given an array of ints, return True if .. 1, 2, 3, .. appears in the array somewhere.

```
array123([1, 1, 2, 3, 1]) → True
array123([1, 1, 2, 4, 1]) → False
array123([1, 1, 2, 1, 2, 3]) → True

def array123(nums):
  for i in range(len(nums)-2):
    if nums[i] == 1 and nums[i+1] == 2 and nums[i+2] == 3:
      return True
  return False
```

Given 2 strings, a and b, return the number of the positions where they contain the same length 2 substring. So "xxcaazz" and "xxbaaz" yields 3, since the "xx", "aa", and "az" substrings appear in the same place in both strings.

```
string_match('xxcaazz', 'xxbaaz') → 3
string_match('abc', 'abc') → 2
string_match('abc', 'axc') → 0

def string_match(a, b):
```

```
count = 0
for i in range(len(a)-1):
  if a[i:i+2] == b[i:i+2]:
    count += 1
return count
```

Logic-1

When squirrels get together for a party, they like to have cigars. A squirrel party is successful when the number of cigars is between 40 and 60, inclusive. Unless it is the weekend, in which case there is no upper bound on the number of cigars. Return True if the party with the given values is successful, or False otherwise.

```
cigar_party(30, False) → False
cigar_party(50, False) → True
cigar_party(70, True) → True
```

```
def cigar_party(cigars, is_weekend):
  if is_weekend:
    return cigars >= 40
  return cigars >= 40 and cigars <= 60
```

You and your date are trying to get a table at a restaurant. The parameter "you" is the stylishness of your clothes, in the range 0..10, and "date" is the stylishness of your date's clothes. The result getting the table is encoded as an int value with 0=no, 1=maybe, 2=yes. If either of you is very stylish, 8 or more, then the result is 2 (yes). With the exception that if either of you has style of 2 or less, then the result is 0 (no). Otherwise the result is 1 (maybe).

```
date_fashion(5, 10) → 2
date_fashion(5, 2) → 0
date_fashion(5, 5) → 1
```

```
def date_fashion(you, date):
  if (you <= 2 or date <= 2):
    return 0
  elif (you >= 8 or date >= 8):
    return 2
  return 1
```

The squirrels in Palo Alto spend most of the day playing. In particular, they play if the temperature is between 60 and 90 (inclusive). Unless it is summer, then the upper limit is 100 instead of 90. Given an int temperature and a boolean is_summer, return True if the squirrels play and False otherwise.

squirrel_play(70, False) → True
squirrel_play(95, False) → False
squirrel_play(95, True) → True

```
def squirrel_play(temp, is_summer):
  if temp < 60:
    return False
  if is_summer:
    return temp <= 100
  return temp <= 90

  """
  First solution (attempt)
  if is_summer:
    return temp >= 60 and temp <= 100
  return temp >= 60 and temp <= 90
  """
```

You are driving a little too fast, and a police officer stops you. Write code to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday -- on that day, your speed can be 5 higher in all cases.

caught_speeding(60, False) → 0
caught_speeding(65, False) → 1
caught_speeding(65, True) → 0

```
def caught_speeding(speed, is_birthday):
  if is_birthday:
    speed -= 5  ## -!
  if speed <= 60:
    return 0
  elif speed >= 81:
    return 2
  return 1
```

Given 2 ints, a and b, return their sum. However, sums in the range 10..19 inclusive, are forbidden, so in that case just return 20.

```
sorta_sum(3, 4) → 7
sorta_sum(9, 4) → 20
sorta_sum(10, 11) → 21

def sorta_sum(a, b):
  sum = a+b
  if sum >= 10 and sum <= 19:
    return 20
  return sum
```

Given a day of the week encoded as 0=Sun, 1=Mon, 2=Tue, ...6=Sat, and a boolean indicating if we are on vacation, return a string of the form "7:00" indicating when the alarm clock should ring. Weekdays, the alarm should be "7:00" and on the weekend it should be "10:00". Unless we are on vacation -- then on weekdays it should be "10:00" and weekends it should be "off".

```
alarm_clock(1, False) → '7:00'
alarm_clock(5, False) → '7:00'
alarm_clock(0, False) → '10:00'

def alarm_clock(day, vacation):
  if vacation:
    if day == 0 or day == 6:
      return "off"
    return "10:00"
  if day == 0 or day == 6:
    return "10:00"
  return "7:00"
```

The number 6 is a truly great number. Given two int values, a and b, return True if either one is 6. Or if their sum or difference is 6. Note: the function abs(num) computes the absolute value of a number.

```
love6(6, 4) → True
love6(4, 5) → False
love6(1, 5) → True

def love6(a, b):
  return a == 6 or b == 6 or a+b == 6 or abs(a-b) == 6
```

Given a number n, return True if n is in the range 1..10, inclusive. Unless "outsideMode" is True, in which case return True if the number is less or equal to 1, or greater or equal to 10.

```
in1to10(5, False) → True
in1to10(11, False) → False
in1to10(11, True) → True

def in1to10(n, outside_mode):
  if outside_mode:
    return n <= 1 or n >= 10
  return n >= 1 and n <= 10
```

Given a non-negative number "num", return True if num is within 2 of a multiple of 10. Note: (a % b) is the remainder of dividing a by b, so (7 % 5) is 2. See also: Introduction to Mod

```
near_ten(12) → True
near_ten(17) → False
near_ten(19) → True

def near_ten(num):
  return num%10 <= 2 or num%10 >= 8
```

Logic-2

We want to make a row of bricks that is **goal** inches long. We have a number of small bricks (1 inch each) and big bricks (5 inches each). Return True if it is possible to make the goal by choosing from the given bricks. This is a little harder than it looks and can be done without any loops. See also: Introduction to MakeBricks

make_bricks(3, 1, 8) → True
make_bricks(3, 1, 9) → False
make_bricks(3, 2, 10) → True

```
def make_bricks(small, big, goal):
  if (small + 5*big < goal):
    return False
  if (goal%5 > small):
    return False
  return True
```

Given 3 int values, a b c, return their sum. However, if one of the values is the same as another of the values, it does not count towards the sum.

lone_sum(1, 2, 3) → 6
lone_sum(3, 2, 3) → 2
lone_sum(3, 3, 3) → 0

```
def lone_sum(a, b, c):
  if a == b and b == c:
    return 0
  elif a == b:
    return c
  elif b == c:
    return a
  elif a == c:
    return b
  else:
    return a + b + c
```

Given 3 int values, a b c, return their sum. However, if one of the values is 13 then it does not count towards the sum and values to its right do not count. So for example, if b is 13, then both b and c do not count.

```
lucky_sum(1, 2, 3) → 6
lucky_sum(1, 2, 13) → 3
lucky_sum(1, 13, 3) → 1
```

```
def lucky_sum(a, b, c):
  if a == 13:
    return 0
  elif b == 13:
    return a
  elif c == 13:
    return a + b
  else:
    return a + b + c
```

Given 3 int values, a b c, return their sum. However, if any of the values is a teen -- in the range 13..19 inclusive -- then that value counts as 0, except 15 and 16 do not count as a teens. Write a separate helper "def fix_teen(n):"that takes in an int value and returns that value fixed for the teen rule. In this way, you avoid repeating the teen code 3 times (i.e. "decomposition"). Define the helper below and at the same indent level as the main no_teen_sum().

```
no_teen_sum(1, 2, 3) → 6
no_teen_sum(2, 13, 1) → 3
no_teen_sum(2, 1, 14) → 3
```

```
def no_teen_sum(a, b, c):
  return fix_teen(a) + fix_teen(b) + fix_teen(c)

def fix_teen(n):
  if n != 15 and n != 16 and n > 12 and n < 20:
    return 0
  else:
    return n
```

For this problem, we'll round an int value up to the next multiple of 10 if its rightmost digit is 5 or more, so 15 rounds up to 20. Alternately, round down to the previous multiple of 10 if its rightmost digit is less than 5, so 12 rounds down to 10. Given 3 ints, a b c, return the sum of their rounded values. To avoid code repetition, write a separate helper "def round10(num):" and call it 3 times. Write the helper entirely below and at the same indent level as round_sum().

round_sum(16, 17, 18) → 60
round_sum(12, 13, 14) → 30
round_sum(6, 4, 4) → 10

```
def round_sum(a, b, c):
  return round10(a) + round10(b) + round10(c)

def round10(num):
  if num%10 >= 5:
    return num + (10-num%10)
  else:
    return num - num%10
```

Given three ints, a b c, return True if one of b or c is "close" (differing from a by at most 1), while the other is "far", differing from both other values by 2 or more. Note: abs(num) computes the absolute value of a number.

close_far(1, 2, 10) → True
close_far(1, 2, 3) → False
close_far(4, 1, 3) → True

```
def close_far(a, b, c):
  if (abs(b-a) < 2 or abs(c-b) < 2 or abs(c-a) < 2):
    """
    This doesn't work because it has to be both other values
    return (abs(b-a) >= 2 or abs(c-b) >= 2 or abs(c-a) >= 2)
    """
    return ((abs(b-a) >= 2 and abs(c-a) >= 2) or (abs(b-a) >= 2 and
abs(b-c) >= 2) or (abs(c-a) >= 2 and abs(c-b) >= 2))
  return False
```

We want make a package of **goal** kilos of chocolate. We have small bars (1 kilo each) and big bars (5 kilos each). Return the number of small bars to use, assuming we always use big bars before small bars. Return -1 if it can't be done.

make_chocolate(4, 1, 9) → 4
make_chocolate(4, 1, 10) → -1
make_chocolate(4, 1, 7) → 2

```
def make_chocolate(small, big, goal):
  if (small + 5*big < goal):
    return -1
  if (goal%5 > small):
    return -1
  if (5*big < goal):
    return goal-(5*big)
  return goal%5
```

String-1

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

```
hello_name('Bob') → 'Hello Bob!'
hello_name('Alice') → 'Hello Alice!'
hello_name('X') → 'Hello X!'
```

```
def hello_name(name):
  return "Hello " + name + "!"
```

Given two strings, a and b, return the result of putting them together in the order abba, e.g. "Hi" and "Bye" returns "HiByeByeHi".

```
make_abba('Hi', 'Bye') → 'HiByeByeHi'
make_abba('Yo', 'Alice') → 'YoAliceAliceYo'
make_abba('What', 'Up') → 'WhatUpUpWhat'
```

```
def make_abba(a, b):
  return a + 2*b + a
```

The web is built with HTML strings like "<i>Yay</i>" which draws Yay as italic text. In this example, the "i" tag makes <i> and </i> which surround the word "Yay". Given tag and word strings, create the HTML string with tags around the word, e.g. "<i>Yay</i>".

```
make_tags('i', 'Yay') → '<i>Yay</i>'
make_tags('i', 'Hello') → '<i>Hello</i>'
make_tags('cite', 'Yay') → '<cite>Yay</cite>'
```

```
def make_tags(tag, word):
  return "<" + tag + ">" + word + "</" + tag + ">"
```

Given an "out" string length 4, such as "<<>>", and a word, return a new string where the word is in the middle of the out string, e.g. "<<word>>".

```
make_out_word('<<>>', 'Yay') → '<<Yay>>'
make_out_word('<<>>', 'WooHoo') → '<<WooHoo>>'
```

make_out_word('[[]]', 'word') → '[[word]]'

```
def make_out_word(out, word):
  return out[:2] + word + out[2:]
```

Given a string, return a new string made of 3 copies of the last 2 chars of the original string. The string length will be at least 2.

extra_end('Hello') → 'lololo'
extra_end('ab') → 'ababab'
extra_end('Hi') → 'HiHiHi'

```
def extra_end(str):
  return 3*str[len(str)-2:]  ## str[-2] would also work
```

Given a string, return the string made of its first two chars, so the String "Hello" yields "He". If the string is shorter than length 2, return whatever there is, so "X" yields "X", and the empty string "" yields the empty string "".

first_two('Hello') → 'He'
first_two('abcdefg') → 'ab'
first_two('ab') → 'ab'

```
def first_two(str):
  if len(str) < 2:
    return str
  return str[:2]
```

Given a string of even length, return the first half. So the string "WooHoo" yields "Woo".

first_half('WooHoo') → 'Woo'
first_half('HelloThere') → 'Hello'
first_half('abcdef') → 'abc'

```
def first_half(str):
  return str[:len(str)/2]
```

Given a string, return a version without the first and last char, so "Hello" yields "ell". The string length will be at least 2.

without_end('Hello') → 'ell'
without_end('java') → 'av'
without_end('coding') → 'odin'

```
def without_end(str):
  """
  This if statement isn't needed, apparently
  if len(str) == 2:
    return ""
  """
  return str[1:len(str)-1]
```

Given 2 strings, a and b, return a string of the form short+long+short, with the shorter string on the outside and the longer string on the inside. The strings will not be the same length, but they may be empty (length 0).

combo_string('Hello', 'hi') → 'hiHellohi'
combo_string('hi', 'Hello') → 'hiHellohi'
combo_string('aaa', 'b') → 'baaab'

```
def combo_string(a, b):
  if (len(a) <= len(b)):  ## Need len()!
    shrt = a
    lng= b
  else:
    shrt = b
    lng = a
  return shrt + lng + shrt
```

Given 2 strings, return their concatenation, except omit the first char of each. The strings will be at least length 1.

non_start('Hello', 'There') → 'ellohere'
non_start('java', 'code') → 'avaode'
non_start('shotl', 'java') → 'hotlava'

```
def non_start(a, b):
  return a[1:] + b[1:]
```

Given a string, return a "rotated left 2" version where the first 2 chars are moved to the end. The string length will be at least 2.

left2('Hello') → 'lloHe'
left2('java') → 'vaja'
left2('Hi') → 'Hi'

```
def left2(str):
  return str[2:] + str[:2]
```

String-2

Given a string, return a string where for every char in the original, there are two chars.

double_char('The') → 'TThhee'
double_char('AAbb') → 'AAAAbbbb'
double_char('Hi-There') → 'HHii--TThheerree'

```
def double_char(str):
  result = ""
  for i in str:
    result += 2*i
  return result
```

Return the number of times that the string "hi" appears anywhere in the given string.

count_hi('abc hi ho') → 1
count_hi('ABChi hi') → 2
count_hi('hihi') → 2

```
def count_hi(str):
  count = 0
  for i in range(len(str)-1):
    if str[i:i+2] == 'hi':
      count += 1
  return count
```

Return True if the string "cat" and "dog" appear the same number of times in the given string.

cat_dog('catdog') → True
cat_dog('catcat') → False
cat_dog('1cat1cadodog') → True

```
def cat_dog(str):
  cat = 0
  dog = 0
  for i in range(len(str)-2):
    if str[i:i+3] == 'cat':
      cat += 1
    elif str[i:i+3] == 'dog':
      dog += 1
  return cat == dog
```

Return the number of times that the string "code" appears anywhere in the given string, except we'll accept any letter for the 'd', so "cope" and "cooe" count.

```
count_code('aaacodebbb') → 1
count_code('codexxcode') → 2
count_code('cozexxcope') → 2
```

```
def count_code(str):
  count = 0
  for i in range(len(str)-3):
    if str[i:i+2] == 'co' and str[i+3] == 'e':
      count += 1
  return count
```

Given two strings, return True if either of the strings appears at the very end of the other string, ignoring upper/lower case differences (in other words, the computation should not be "case sensitive"). Note: s.lower() returns the lowercase version of a string.

```
end_other('Hiabc', 'abc') → True
end_other('AbC', 'HiaBc') → True
end_other('abc', 'abXabc') → True
```

```
def end_other(a, b):
  a = a.lower()
  b = b.lower()
  if len(a) >= len(b):
    return a[len(a)-len(b):] == b
  else:
    return b[len(b)-len(a):] == a
```

Return True if the given string contains an appearance of "xyz" where the xyz is not directly preceeded by a period (.). So "xxyz" counts but "x.xyz" does not.

```
xyz_there('abcxyz') → True
xyz_there('abc.xyz') → False
xyz_there('xyz.abc') → True
```

```
def xyz_there(str):
  if len(str) > 2 and str[:3] == 'xyz':
    return True
  for i in range(len(str)-3):
    if str[i+1:i+4] == 'xyz' and str[i] != '.':
      return True
  return False
```

List-1

Given an array of ints, return True if 6 appears as either the first or last element in the array. The array will be length 1 or more.

first_last6([1, 2, 6]) → True
first_last6([6, 1, 2, 3]) → True
first_last6([13, 6, 1, 2, 3]) → False

```
def first_last6(nums):
  return nums[0] == 6 or nums[len(nums)-1] == 6
```

Given an array of ints, return True if the array is length 1 or more, and the first element and the last element are equal.

same_first_last([1, 2, 3]) → False
same_first_last([1, 2, 3, 1]) → True
same_first_last([1, 2, 1]) → True

```
def same_first_last(nums):
  return len(nums) >= 1 and (nums[0] == nums[len(nums)-1])  ##
nums[-1] also works
```

Return an int array length 3 containing the first 3 digits of pi, {3, 1, 4}.

make_pi() → [3, 1, 4]

```
def make_pi():
  return [3, 1, 4]
```

Given 2 arrays of ints, a and b, return True if they have the same first element or they have the same last element. Both arrays will be length 1 or more.

common_end([1, 2, 3], [7, 3]) → True
common_end([1, 2, 3], [7, 3, 2]) → False
common_end([1, 2, 3], [1, 3]) → True

```
def common_end(a, b):
  return a[0] == b[0] or a[len(a)-1] == b[len(b)-1]
```

Given an array of ints length 3, return the sum of all the elements.

```
sum3([1, 2, 3]) → 6
sum3([5, 11, 2]) → 18
sum3([7, 0, 0]) → 7
```

```
def sum3(nums):
  sum = 0
  for num in nums:
    sum += num
  return sum
```

Given an array of ints length 3, return an array with the elements "rotated left" so {1, 2, 3} yields {2, 3, 1}.

```
rotate_left3([1, 2, 3]) → [2, 3, 1]
rotate_left3([5, 11, 9]) → [11, 9, 5]
rotate_left3([7, 0, 0]) → [0, 0, 7]
```

```
def rotate_left3(nums):
  return nums[1:] + nums[0:1]
```

Given an array of ints length 3, return a new array with the elements in reverse order, so {1, 2, 3} becomes {3, 2, 1}.

```
reverse3([1, 2, 3]) → [3, 2, 1]
reverse3([5, 11, 9]) → [9, 11, 5]
reverse3([7, 0, 0]) → [0, 0, 7]
```

```
def reverse3(nums):
  return nums[2:] + nums[1:2] + nums[0:1]

  """
  First Attempt
  rev = []
  index = 0
  for i in range(len(nums)-1, -1, -1):
    rev[index] = nums[i]
    index += 1
  return rev
  """
```

Given an array of ints length 3, figure out which is larger between the first and last elements in the array, and set all the other elements to be that value. Return the changed array.

```
max_end3([1, 2, 3]) → [3, 3, 3]
max_end3([11, 5, 9]) → [11, 11, 11]
max_end3([2, 11, 3]) → [3, 3, 3]

def max_end3(nums):
  if nums[0] >= nums[len(nums)-1]:
    max = nums[0]
  else:
    max = nums[len(nums)-1]
  return [max, max, max]
```

Given an array of ints, return the sum of the first 2 elements in the array. If the array length is less than 2, just sum up the elements that exist, returning 0 if the array is length 0.

```
sum2([1, 2, 3]) → 3
sum2([1, 1]) → 2
sum2([1, 1, 1, 1]) → 2

def sum2(nums):
  sum = 0
  for i in range(len(nums)):
    if i < 2:
      sum += nums[i]
  return sum
```

Given 2 int arrays, a and b, each length 3, return a new array length 2 containing their middle elements.

```
middle_way([1, 2, 3], [4, 5, 6]) → [2, 5]
middle_way([7, 7, 7], [3, 8, 0]) → [7, 8]
middle_way([5, 2, 9], [1, 4, 5]) → [2, 4]

def middle_way(a, b):
  return [a[1], b[1]]
```

Given an array of ints, return a new array length 2 containing the first and last elements from the original array. The original array will be length 1 or more.

```
make_ends([1, 2, 3]) → [1, 3]
make_ends([1, 2, 3, 4]) → [1, 4]
make_ends([7, 4, 6, 2]) → [7, 2]

def make_ends(nums):
  return [nums[0], nums[len(nums)-1]]
```

Given an int array length 2, return True if it contains a 2 or a 3.

```
has23([2, 5]) → True
has23([4, 3]) → True
has23([4, 5]) → False

def has23(nums):
  for num in nums:
    if num == 2 or num == 3:
      return True
  return False
```

List-2

Return the number of even ints in the given array. Note: the % "mod" operator computes the remainder, e.g. 5 % 2 is 1.

count_evens([2, 1, 2, 3, 4]) → 3
count_evens([2, 2, 0]) → 3
count_evens([1, 3, 5]) → 0

```
def count_evens(nums):
  count = 0
  for n in nums:
    if n%2 == 0:
      count += 1
  return count
```

Given an array length 1 or more of ints, return the difference between the largest and smallest values in the array. Note: the built-in min(v1, v2) and max(v1, v2) functions return the smaller or larger of two values.

big_diff([10, 3, 5, 6]) → 7
big_diff([7, 2, 10, 9]) → 8
big_diff([2, 10, 7, 2]) → 8

```
def big_diff(nums):
  ## I can also use the list max() and min() functions. Super helpful
  return max(nums)-min(nums)
```

Return the "centered" average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division to produce the final average. You may assume that the array is length 3 or more.

centered_average([1, 2, 3, 4, 100]) → 3
centered_average([1, 1, 5, 5, 10, 8, 7]) → 5
centered_average([-10, -4, -2, -4, -2, 0]) → -3

```
def centered_average(nums):
  real_sum = sum(nums)-max(nums)-min(nums)
  return real_sum/(len(nums)-2)
```

Return the sum of the numbers in the array, returning 0 for an empty array. Except the number 13 is very unlucky, so it does not count and numbers that come immediately after a 13 also do not count.

sum13([1, 2, 2, 1]) → 6
sum13([1, 1]) → 2
sum13([1, 2, 2, 1, 13]) → 6

```
def sum13(nums):
  my_sum = 0
  for i in range(len(nums)-1):
    if nums[i] != 13:
      my_sum += nums[i]
    else:
      ## i += 1 <----------This doesn't work! I don't know why
      nums[i+1] = 0
  if len(nums) > 1 and nums[-2] != 13 and nums[-1] != 13:
    return my_sum + nums[-1]
  return my_sum
```

Return the sum of the numbers in the array, except ignore sections of numbers starting with a 6 and extending to the next 7 (every 6 will be followed by at least one 7). Return 0 for no numbers.

sum67([1, 2, 2]) → 5
sum67([1, 2, 2, 6, 99, 99, 7]) → 5
sum67([1, 1, 6, 7, 2]) → 4

```
def sum67(nums):
  my_sum = 0
  for i in range(len(nums)):
    if nums[i] != 6:
      my_sum += nums[i]
    else:
      temp = nums[:]
      for q in range(nums.index(6)):
        temp[q] = 0
      for k in range(i, temp.index(7)+1):  ## Ohhhhhhh! nums.index(7)+1
doesn't work because there might be a 7 before a 6!
        ## nums[i:] also doesn't work because the indeces are off
        nums[k] = 0
  return my_sum

  """
  First attempt
  int sum = 0;
  for (int i = 0; i < nums.length; i++) {
    if (nums[i] != 6) {
      sum += nums[i];
    } else {
      for (int k = i+1; k < nums.length; k++) {
        if (nums[k] == 7) {
          i = k; // Really k+1-1, but y'know...substring rules
          break;
        }
      }
    }
  }
  return sum;
}
"""
```

Given an array of ints, return True if the array contains a 2 next to a 2 somewhere.

has22([1, 2, 2]) → True
has22([1, 2, 1, 2]) → False
has22([2, 1, 2]) → False

```
def has22(nums):
  for i in range(len(nums)-1):
    if nums[i] == 2 and nums[i+1] == 2:
      return True
  return False
```