Problems

- Teenager

Given a person's age, return `true` if the person is a teenager (age 13 - 19).
`isTeenager(5) = false`
`isTeenager(15) = true`

`Solution:`

```
public Boolean isTeenager(Integer age) {
    //code here
    // if(age>=13 && age<=19){
    //     return true;
    // }else{
    //     return false;
    // }
  return  age>=13 && age <= 19 ? true : false;
}
```

- Number Difference

- Implement a function `diff` that calculates the absolute difference between two integers.
- `diff(5, 2) = 3`
- `diff(2, 5) = 3`

```
public Integer diff(Integer a, Integer b) {
    //code here
  //  return Math.abs(a-b);
    if(a<b) return b - a;
    else return a - b;
}
```

- Sum Equals

Given Integers a, b, and c, return `true` if a and b add up to c.
`sumEquals(5, 5, 10) = true`
`sumEquals(2, 8, 9) = false`

```
public Boolean sumEquals(Integer a, Integer b, Integer c) {
    //code here
    return a+b==c ? true : false;
}
```

- Ascending Order
- Given three Integers a, b, and c, return `true` if they are in ascending order. For our purposes, two equal numbers will be considered to be in an ascending order.

- ascendingOrder(10, 10, 15) = true
- ascendingOrder(15, 14, 13) = false

```
public Boolean ascendingOrder(Integer a, Integer b, Integer c) {
    //code here
    return b>=a && c>=b ? true : false;
}
```

- A or An

Given a work, prepend it with the correct indefinite article ("a" or "an") followed by a space based on the following rule: words starting with a vowel (a, e, i, o, or u) are prepended with "an", while words starting with any other letter are prepended with "a".
aOrAn('apple') = 'an apple'
aOrAn('banana') = 'a banana'

```
public String aOrAn(String word) {
    //code here
    if(word.startsWith('a')||
    word.startsWith('e')||
    word.startsWith('i')||
    word.startsWith('o')||
    word.startsWith('u')) {
        return 'an ' + word;
    }else{
     return   'a ' + word;
    }
}
```

- Largest of Three

Given three Integers, return the largest

```
public static Integer findLargest(Integer num1, Integer num2, Integer num3) {
    //code here
    if(num1>num2 && num1>num3){
        return num1;
    }else if(num2>num1 && num2>num3){
        return num2;
    }else{
        return num3;
    }

}
```

- Passing Students

A student passes a course if *any* ***two*** *of the following three* conditions are true: they have passed the exam, they have passed assignments, and they have passed the course project.
Implement the function `isPassed` that takes in three parameters `passedExam`, `passedAssignments`, and `passedProject`, and returns `true` of at least two of the passed variables are true.
`isPassed(true, false, true) = true`. Student did not pass assignments, but passes overall because they passed the exam and the project.
`isPassed(false, false, true) = false`. Student only passed the project, and therefore does not pass overall.

```
public Boolean isPassed(Boolean passedExam, Boolean passedAssignments, Boolean
 passedProject) {
    //code here
    return (passedExam && passedAssignments)
        || (passedExam && passedProject)
        || (passedAssignments && passedProject);
}
```

- Ends With 0

Given an integer, return `true` if the integer ends with 0, otherwise return `false`.
`isEndWithZero(12) == false`
`isEndWithZero(1230) == true`

```
public Boolean isEndWithZero(Integer num){
     //code here
    //  if(Math.mod(num,5)==0){
    //      return true;
    //  }else{
    //      return false;
    //  }

    return Math.mod(num,5)==0 ? true : false;
}
```

- Which Two

Given Integers a, b, and c, determine if any two of them add up to the third and
return `'a'`, `'b'`, `'c'` depending on which the sum is. If no two numbers add up to a third number, return an empty string. Assume that multiple solutions do not exist.
`whichTwo(5, 10, 5) = 'b'`
`whichTwo(2, 0, 3) = ''`

```
public String whichTwo(Integer a, Integer b, Integer c) {
    //code here
    if(a+b==c){
        return 'c';
    }else if(a+c==b){
        return 'b';
```

```
    }else if(b+c==a){
        return 'a';
    }else{
        return '';
    }
}
```

- Even or Odd

Given an Integer, return `'even'` if the Integer is even, or `'odd'` if the Integer is odd. Remember to use the `Math.mod` function.
`evenOrOdd(15) = 'odd'`
`evenOrOdd(-64) = 'even'`

```
public String evenOrOdd(Integer num) {
    //code here
    return Math.mod(num,2)!=0 ? 'odd' : 'even';
}
```

- Rock Paper Scissors

Rock beats scissors, scissors beats paper, paper beats rock. Implement the method `rockPaperScissors` that takes as parameters two strings `player1` and `player2` representing the moves played by player 1 and player 2, valid moves being `'rock'`, `'paper'`, and `'scissors'`. Return 1 if player 1 wins, 2 if player 2 wins, and 0 if no one wins.
`rockPaperScissors('rock', 'paper') = 2`
`rockPaperScissors('scissors', 'paper') = 1`
`rockPaperScissors('paper', 'paper') = 0`

```
public Integer rockPaperScissors(String player1, String player2) {
    //code here
    // if(player1==player2) return 0;
    //  if (player1 == 'rock' && player2 == 'paper') return 2;
    // if (player1 == 'rock' && player2 == 'scissors') return 1;
    // if (player1 == 'paper' && player2 == 'rock') return 1;
    // if (player1 == 'paper' && player2 == 'scissors') return 2;
    // if (player1 == 'scissors' && player2 == 'rock') return 2;
    // if (player1 == 'scissors' && player2 == 'paper') return 1;
    // return 0;

    if(player1==player2) return 0;
    Map<String,String> winMap = new Map<String,String> {
        'rock' => 'scissors',
        'paper' => 'rock',
        'scissors' => 'paper'
    };
    if(winMap.get(player1)==player2) return 1;
```

```
    else return 2;

}
```

- Age Group

Given a person's age, return their age group as a string: 'Infant' for ages 0-1, 'Child' for ages 2 - 14, 'Youth' for ages 15 - 21, and 'Adult' for ages 22+.
`ageGroup(5) = 'Child'`
`ageGroup(15) = 'Youth'`

```
public String ageGroup(Integer n) {
    //code here
    if(n<=1) return 'Infant';
    if(n<= 14) return 'Child';
    if(n<=21) return 'Youth';
    return 'Adult';
}
```

- Companion Plants

Some plants are considered companion plants. They grow better when planted next to each other. For the purpose of this problem, we consider the following plants to be companions: lettuce and cucumbers, lettuce and onions, onions and carrots, and onions and tomatoes.
Write a function `isCompanion` that takes as input two strings `plant1` and `plant2`. If the two plants are companion plants based on the criteria described above, return true. Otherwise, return false.
`companionPlants('onions', 'lettuce') = true`
`companionPlants('lettuce', 'tomatoes') = false`

```
public Boolean companionPlants(String plant1, String plant2) {
    //code here
    // if(plant1=='lettuce' && plant2 == 'cucumbers')
    // return true;
    // if(plant1 == 'cucumbers' && plant2 == 'lettuce')
    // return true;
    // if(plant1 == 'lettuce' && plant2 == 'onions')
    // return true;
    // if(plant1 == 'onions' && plant2 == 'lettuce')
    // return true;
    // if(plant1 == 'onions' && plant2 == 'carrots')
    // return true;
    // if(plant1 == 'carrots' && plant2 == 'onions')
    // return true;
    // if(plant1 == 'onions' && plant2 == 'tomatoes')
    // return true;
    // if(plant1 == 'tomatoes' && plant2 == 'onions')
    // return true;
    // return false;
```

```
      Map<String, Set<String>> companionPlants  = new Map<String, Set<String>>
();
        companionPlants.put('lettuce', new Set<String>{'cucumbers', 'onions'})
;
        companionPlants.put('cucumbers', new Set<String>{'lettuce'});
        companionPlants.put('onions', new Set<String>{'lettuce', 'carrots', 't
omatoes'});
        companionPlants.put('carrots', new Set<String>{'onions'});
        companionPlants.put('tomatoes', new Set<String>{'onions'});

        return companionPlants.get(plant1).contains(plant2);









}
```

- Leap Year

A year is considered a leap year if it is evenly divisible by 4, with the exception of years that are also evenly divisible by 100. Years evenly divisible by 100 must also be evenly evenly divisible by 400 to by considered leap years. Implement a method `isLeapYear` that takes as input an Integer `year` and returns `true` if the year is a leap year, and `false` otherwise.
`isLeapYear(1900) = false`. Year 1900 is evenly divisible by 4, but it is also evenly divisible by 100 which means it additionally needs to be evenly divisible by 400 to qualify as a leap year. 1900 is not evenly divisible by 400.
`isLeapYear(2000) = true`. Year 2000 is evenly divisible by 4. It is also evenly divisibly by 100, which means it additionally needs to be evenly divisible by 400, which it is. Therefore, it is a leap year.
`isLeapYear(2004) = true`. Year 2004 is evenly divisible by 4. It is not divisible by 100, and therefore a leap year.
`isLeapYear(1933) = false`. Year 1933 is not evenly divisible by 4, and therefore not a leap year.

```
public Boolean isLeapYear(Integer year) {
```

```
    //code here
//    Boolean divBy4 = Math.mod(year, 4) == 0;
//    Boolean divBy100 = Math.mod(year,100) == 0;
//    Boolean divBy400 = Math.mod(year,400) == 0;

//    if(!divBy4) return false;
//    else if(!divBy100) return true;
//    else if(!divBy400) return false;
//    else return true;
      return (Date.isLeapYear(year));
}
```

- Prime Number

A prime number is a number greater than 1 that is not evenly divisible by any number greater than one and smaller than itself. For example, 13 is a prime number because it is not evenly divisible by any number between 1 and 13.
Implement the function `isPrime` that takes as input an integer greater than 1, returns `true` if the integer is a prime number, and returns `false` otherwise. Assume that the input will always be greater than 1.
`isPrime(10) = false`. 10 is not a prime number because it is evenly divisible by 2 and 5.
`isPrime(23) = true`. 23 is a prime number because it is not evenly divisible by any number from 2 to 22.

```
public Boolean isPrime(Integer num) {
    //code here
    for(Integer i = 2; i<num; i++){
        if(Math.mod(num,i) == 0) return false;

    }
    return true;
}
```

- Sum 1 to N

Implement the method `sumToN` that calculates and returns the sum of all numbers (inclusive) from 1 to n. Assume that n will be non-zero positive integer.
`sumToN(5) = 15`
`sumToN(2) = 3`

```
public Integer sumToN(Integer n) {
    //code here
    Integer sum = 0;

   for(Integer i = 1; i<=n;i++){
       sum = sum + i;
```

```
    }
     return sum;
}
```

- Full Name

Given two non-empty strings `firstName` and `lastName`, return the name as a single string with a space in between (`firstName lastName`).
`formatName('Jane', 'Doe') = 'Jane Doe'`

```
public String formatName(String firstName, String lastName) {
    //code here
    return firstName+ ' ' + lastName;
}
```

- Format Name

Given two strings `firstName` and `lastName`, return the name in the format LastName, FirstName. In case one of the names is null or empty, return only the non-empty part of the name. If both are null or empty, return an empty string.
`formatName('Jane', 'Doe') = 'Doe, Jane'`
`formatName('Jane', '') = 'Jane'`

```
public String formatName(String firstName, String lastName) {
    //code here
    if(String.isBlank(firstName) && String.isBlank(lastName)) return '';
    else if(String.isBlank(firstName)) return lastName;
    else if(String.isBlank(lastName)) return firstName;
    return lastName + ', ' + firstName;
}
```

- Name from Email

Implement a function `nameFromEmail` that takes as input a valid email address in the format `firstname.lastname@example.com`. The function should extract the first name and last name from this email address and return a capitalized full name (i.e. FirstName LastName). Assume that the input will always be a valid email address with both the first name and last name separated by a period (.).
`nameFromEmail('john.doe@apexsandbox.io') = 'John Doe'`
`nameFromEmail('JANE.DOE@GMAIL.COM') = 'Jane Doe'`

```
public String nameFromEmail(String email) {
    //code here
    String firstName = email.substringBefore('.');
    String lastName = email.substringBetween('.','@');
    return firstName.toLowerCase().capitalize() + ' ' + lastName.toLowerCase()
.capitalize();
```

```
}
```

- Change Time Format

`13:50` and `01:50 PM` are 24-hour and 12-hour representations of the same time. Implement the method `changeTimeFormat` that takes as input a string `strTime` formatted as a 24-hour string, and returns the equivalent 12-hour string.
Examples:
`changeTimeFormat('08:05')` returns `'08:05 AM'`
`changeTimeFormat('00:05')` returns `'12:00 AM'`
`changeTimeFormat('23:15')` returns `'11:15 PM'`

```
public String changeTimeFormat(String strTime) {
    //code here
    String[] parts = strTime.split(':');
    Integer hours  = Integer.valueOf(parts[0]);
    Integer minutes = Integer.valueOf(parts[1]);
    String suffix = ' AM';

    if(hours == 12) suffix = ' PM';
    if(hours>12){
        suffix= ' PM';
        hours -= 12;
    }
    if(hours == 0){
        hours = 12;
    }
    String strHours = String.valueOf(hours);
    if(strHours.length() == 1) strHours = '0' + strHours;
    return strHours + ':' + parts[1] + suffix;
}
```

- Fibonacci

The first two numbers in the fibonacci sequence are 1, and all other numbers in the sequence are defined as the sum of the last two fibonacci numbers. The first 10 numbers in the fibonacci sequence are 1, 1, 2, 3, 5, 8, 13, 21, 34, and 55.
Implement the function `fibonacci` that takes as input an Integer n and returns the *nth* fibonacci number. Assume that n will always be greater than 0.
`fibonacci(1) = 1`
`fibonacci(2) = 1`
`fibonacci(5) = 5`

```
public Integer fibonacci(Integer n) {
    //code here
    Integer count = 1;
    Integer a = 0;
    Integer b = 1;
```

```
    while(count<n+1){
        Integer sum = a + b;
        a = b;
        b = sum;
        count++;
    }
    return a;
}
```

- Next Prime

A prime number is a number greater than 1 that is not evenly divisible by any number greater than one and smaller than itself. For example, 13 is a prime number because it is not evenly divisible by any number from 2 to 12.
Implement the function `nextPrime` that takes as input an integer `num` and returns the smallest prime number greater than `num`.
`nextPrime(10) = 11`. 11 is the smallest prime number greater than 10
`nextPrime(8) = 11`. 11 is the smallest prime number greater than 8

```
public Integer nextPrime(Integer num) {
    //code here
    if(num<2) return 2;
    while(!isPrime(++num));
    return num;
}
public Boolean isPrime(Integer num){
    for(Integer i = 2;i<num;i++){
        if(Math.mod(num,i) == 0) return false;

    }
    return true;
}
```