0.1
You want to implement combat within a role playing game on a computer. Specifically, the game rules for damage inflicted by a hit are:

- In order to figure out damage from one hit, you throw a N-sided die.
- The result of one throw will be between 1 and N (including both, e.g., a 6-sided die has six sides, labeled 1 to 6).
- If the result is 1 to N-1, that is the resulting damage from the hit.
- If the result in N, however, you hit critically, and you throw again, adding the results.
- If you throw again, the same rules apply, potentially resulting in doubly or more critical hits.

For example, if you use a 4-sided die and throw a 3, the damage is 3. If you throw a 4 instead, you throw again. If that results in a 3, the total damage is 7. If you happen to throw two 4s after each other and then a 2, the total damage is 10.

Fill in the missing code. The number of lines corresponds to the answer key. Your code may vary.

```java
final static Random rand = new Random();
public static int damage(int n) {
    int result = rand.nextInt(n) + 1;
    --------------
    if (result < n) return result;
    --------------
    return result + damage(n);
    ---------------------
}
```

0.2
You want to implement flipping a coin on a computer. However, you want to be fancy and include the possibility of the coin standing on edge, with a probability of 2%. In that case you flip it again, until the coin lands on either side. “Heads” give 1 point, “Tails” give 0 points, and an edge gives an additional 2 points (potentially multiple times). Most of the time the return value will be 0 or 1. However, if the coin landed on edge once, the return value will be either 2 or 3, depending on the final toss. If it landed on edge twice, it could be either 4 or 5, in a similar way. Fill in the missing code. The number of lines corresponds to the answer key. Your code may vary.

```java
final static Random rand = new Random();
public static int flip() {
    int randomPercentage = rand.nextInt(100);
    if (randomPercentage < 2) return 2 + flip();
    ----------------------
    if (randomPercentage < 51) return 0;
    ----------------------
    return 1;
    ----------------------
}
```

0.3
This task involves a game with plush owls. You start the game with a given number of owls. You can then give back some of the owls, according to the rules below. If multiple rules apply, it is your choice which to use. If none applies, you lose. The goal is to end up with exactly 42 owls. n is the number of owls you currently have.

- If n is even, you may give back exactly n/2 owls.
- If \( n \) is divisible by 3 or 4, you may multiply the last two digits of \( n \) and give back this many owls. (the last digit of \( n \) is \( n\%10 \), the next-to-last digit is \( (n\%100)/10 \)).

- If \( n \) is divisible by 5, then you may give back exactly 42 owls.

For example, suppose you start with 250 owls, you could do the following moves:

- Since 250 is divisible by 5, you may return 42 owls, leaving you with 208.
- Since 208 is even, you may return half of them, leaving you with 104.
- Since 104 is even again, you may do the same again, leaving you with 52.
- Since 52 is divisible by 4, you may multiply the last two digits (2*5=10), and return 10 owls, leaving you with 42 owls, and resulting in a win.

Write a recursive function to meet this specification:

```java
public class Owls {
    public static boolean owls(int n) {
        if (n<42) return false;
        if (n==42) return true;
        if (n%2==0 && owls(n/2)) return true;
        if ((n%3==0||n%4==0) && owls(n-((n%10)*((n%100)/10)))) return true;
        if (n%5==0 && owls(n-42)) return true;
        return false;
    }
}
```

This task involves a game with plush bunnies. You start the game with a given number of bunnies. You can then give back some of the bunnies, according to the rules below. If multiple rules apply, it is your choice which to use. If none applies, you loose. The goal is to end up with exactly 42 bunnies. \( n \) is the number of bunnies you currently have.

- If \( n \) is even, you may give back exactly \( n/2 \) bunnies.
- If \( n \) is divisible by 3 or 4, you may multiply the last two digits of \( n \) and give back this many bunnies. (the last digit of \( n \) is \( n\%10 \), the next-to-last digit is \( (n\%100)/10 \)).
- If \( n \) is divisible by 5, then you may give back exactly 42 bunnies.

For example, suppose you start with 250 bunnies, you could do the following moves:

- Since 250 is divisible by 5, you may return 42 bunnies, leaving you with 208.
- Since 208 is even, you may return half of them, leaving you with 104.
- Since 104 is even again, you may do the same again, leaving you with 52.
- Since 52 is divisible by 4, you may multiply the last two digits (2*5=10), and return 10 bunnies, leaving you with 42 bunnies, and resulting in a win.

Write a recursive function to meet this specification:
The return value should indicate whether it is possible to win this game if you start with n bunnies. For example:

- bunnies(250) is true
- bunnies(42) is true
- bunnies(84) is true
- bunnies(53) is false
- bunnies(41) is false

```java
class Bunnies {
    public static boolean bunnies(int n) {
        // smaller than 42
        if (n<42) return false;
        // equal 42
        if (n==42) return true;
        // divisible by 2
        if (n%2==0 && bunnies(n/2)) return true;
        // divisible by 3 or 4
        if ((n%3==0||n%4==0) && bunnies(n-((n%10)*((n%100)/10)))) return true;
        // divisible by 5
        if (n%5==0 && bunnies(n-42)) return true;
        return false;
    }
}
```

0.5

What is wrong with the following program, what happens when you run it? If this method would have been implemented using a loop, and would have a similar error, what would happen then if run?

```java
class JavaIsToJavascriptWhatCarIsToCarpet {
    public static int factorial(int n) {
        return(n * factorial(n-1));
    }
    public static void main(String args[]) {
        factorial(4);
    }
}
```

There is an exit-condition missing in the recursive function. When run, you get a stack overflow. This is similar to infinite loops, so the same in an iterative program would result in a hang during execution.

0.6

What is wrong with the following program, what happens when you run it? If this method would have been implemented recursively, and would have a similar error, what would happen then if run?

```java
class JavaIsToJavascriptWhatCarIsToCarpet {
    public static int factorial(int n) {
        int result = 1;
        while (n > 1) {
            result *= n;
        }
        return result;
    }
    public static void main(String args[]) {
        assert(factorial(4) == 24);
    }
}
```

There is an exit-condition missing in the while loop. When run, the program hangs. This is similar to infinite recursions, so the same in a recursive program would result in a stack overflow.

0.7

Implement the factorial() method in the JavaIsToJavascriptWhatCarIsToCarpet class above recursively, correctly.
public static int factorial(int n) {
    if (n < 1) return 1;
    return(n * factorial(n-1));
}

0.8

What is wrong with the following program, what happens when you run it? If this method would have been implemented recursively, and would have a similar error, what would happen then if run?

public class JavaIsToJavascriptWhatCarIsToCarpet {
    public static int factorial(int n) {
        int result = 1;
        while (n > 1) {
            result *= n;
        }
        return result;
    }
    public static void main(String args[]) {
        assert(factorial(4) == 24);
    }
}

There is an exit-condition missing in the while loop. When run, the program hangs. This is similar to infinite recursions, so the same in a recursive program would result in a stack overflow.

0.9

Given an List of Integers, is it possible to choose a group of Integers, such that the sum of all Integers in that group matches a given target with this additional constraint: If a value in the List is chosen to be in the group, the value immediately following it in the List must not be chosen. Do not use loops.

Example inputs and returns for groups():

[] ,0     false
[1],0     false
[1],1     true
[1,2], 1   true
[1,2], 3   false
[1,0], 1   true
[1,0], 0   true
[1,2,3], 4 true

You may (but don’t have to) use the subList method, defined in java.util.List, which is declared as follows, and returns a part (slice) of the input array (fromIndex inclusive, toIndex exclusive):

List<E> subList(int fromIndex, int toIndex)

Fill in the missing code. The number of lines corresponds to the answer key. Your code may vary.

public static boolean groups(List<Integer> list, int target) {
    if (list.size() == 0) return false;
    -------------------
    if (list.get(0) == target) return true;
    -------------------
    if (list.size() >= 2 &&
    -------------------
        groups(list.subList(2, list.size()), target-list.get(0)) == true) return true;
    -------------------
    return groups(list.subList(1, list.size()), target);
    -------------------
}
0.10
Write a recursive program to compute a double factorial, \( n!! = n \times (n-2)!! \), \( 0!! = 1 \), \( 1!! = 1 \).

```java
public class DFac {
    static int dfac(int n) {
        if(n < 2) return 1;
        return n*dfac(n-2);
    }
    public static void main(String[] args) {
        assert dfac(7) == 7*5*3*1; } }
```

0.11
Does the following code compile? If it does not, how can it be fixed? If it does, what is the output? Does it throw an exception? If so, how can it be fixed?

```java
public class Series {
    public static int func(int j) {
        System.out.println(j);
        if (j==1) return 1;
        return 2*func(j-1) + 5*func(j-2); }
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        if (N<1) {
            System.out.println("invalid argument");
            return;
        }
        System.out.println(func(N)); } }
```

The code compiles, but when it runs it throws a `StackOverflowError`

replace the code: if (j==1) return 1;

with the code: if (j<1) return 1;

0.12
Which of these will happen if the recursive method does not have a base case?

1. An infinite loop occurs, hanging forever.
2. The program stops after some time with an error.
3. After 1000000 calls it will be automatically stopped.
4. None of the mentioned

The program stops after some time with an error. (stack overflow)

0.13
What is the output of this program?

```java
class recursion {
    int func (int n) {
        int result;
        result = func (n - 1);
        return result;
    }
}
class Output {
```
public static void main(String args[]) {
    recursion obj = new recursion();
    System.out.print(obj.func(12));
}

1. 0
2. 1
3. 12
4. Compilation Error
5. Runtime Error

Runtime Error: Exception in thread main java.lang.StackOverflowError

0.14

What is recursion?

1. Recursion is a generic class.
2. Recursion is a process of setting a value based on its previous value.
3. Recursion is a process of defining a method that calls itself.
4. Recursion is a process of repeatedly calling other methods.

Recursion is a process of defining a method that calls itself.