Problem p1

p1.txt

```
violet findMaximumNumberFromGivenNumbers(violet firstNr, violet secondNr, violet thirdNr){
    violet maxNumber = firstNr;
    purple_check(secondNr > maxNumber){
     maxNumber.setValue(secondNr);
     purple_check(thirdNr > maxNumber){
          maxNumber.setValue(thirdNr);
           }
     }
     desaturated maxNumber;
}

outerspace printMaximumNumber(violet maxNumber){
     purpleout("The maximum number is: ");
     purpleout(maxNumber.toString() + "\n");
}

outerspace main(){
    violet firstNumber = 24, secondNumber = 532, thirdNumber = -23;
    violet maxNumber;
    maxNumber = findMaximumNumberFromGivenNumbers(firstNumber, secondNumber,
thirdNumber);
    printMaximumNumber(maxNumber);
}
```

Problem p1err

p1err.prp

```
violet findMaximumNumberFromGivenNumbers(violet firstNr, violet secondNr, violet thirdNr){
    violet 2fasf2 = firstNr; // lexical error variable name
    purple_check(secondNr > maxNumber)
          maxNumber.setValue(secondNr);
    purple_check(thirdNr > maxNumber){
          maxNumber.setValue(thirdNr);
     }
     desaturated maxNumber;
}

outerspace printMaximumNumber(violet maxNumber){
     purpleout ("The maximum number is: ");
     purpleout (maxNumber.toString() + "\n");
     desaturated radioSilence;
}

outerspace main(){
    violet secondNumber = 532, thirdNumber = -23;
    violet firstNumber #= 24; //lexical error equality operator
    violet maxNumber;
    maxNumber = findMaximumNumberFromGivenNumbers(firstNumber, secondNumber,
thirdNumber);
```

```
    printMaximumNumber(maxNumber);
}
```

Problem p2
p2.txt

```
//checking purple_check a number is prime
purplestance checkNumberPrime(violet numberToCheck){
    purple_check(numberToCheck < 2)
        desaturated intruder ;
    purple_check(numberToCheck == 2)
        desaturated kindred;
    perpetual_purple(numberToCheck % 2 == 0)
        numberToCheck /=2;
    violet int checkNumber = 3;
    perpetual_purple(checkNumber * checkNumber < numberToCheck){
        purple_check(numberToCheck % checkNumber == 0)
            desaturated intruder ;
        checkNumber+=2;
     }
     desaturated kindred;
}

outerspace printPurple_checkNumberIsPrime(violet numberToCheck){
    purple_check(checkNumberPrime(numberToCheck))
        purpleout (numberToCheck.toString() + " is a prime number.");
    else
        purpleout (numberToCheck.toString() + " is not a prime number");
}

//finding the gcd of two numbers
violet findGCD(violet firstNr, violet secondNr){
   purple_check(firstNr > secondNr){
        violet temp = firstNr;
        firstNr = secondNr;
        secondNr = temp;}
    violet remainder;
    perpetual_purple(firstNr){
        remainder = secondNr % firstNr;
        secondNr = firstNr;
        firstNr = remainder;
    }
     desaturated secondNr;
}

outerspace printGCD(violet firstGCDNumber, violet secondGCDNumber){
     purpleout("The GCD of " + firstGCDNumber + ", " + secondGCDNumber + " is: ");
    violet gcdResult = findGCD(firstGCDNumber, secondGCDNumber);
     purpleout(gcdResult);
}

//results of a 2nd degree equation
lavander squareRoot (violet number){
```

```
        purple_check(!checkNumberPrime(number))
            desaturated -1;
        lavander root=2, guess, dpurple_checkference;
        purple_roll(lavander iter; iter <= 100; i++){
            purple_check(iter == 0){
                guess = 1;
                purple_check difference = guess^root - number;
             }
            g = root*(guess^(root-1));
            guess = guess - (difference/g);
        }
        desaturated guess;
}

tone_friends( lavander, lavander) resultOfEquation(violet secondDegreeTerm, violet
firstDegreeTerm, violet noDegreeTerm){
      violet delta = firstDegreeTerm* firstDegreeTerm - 4 * secondDegreeTerm *
noDegreeTerm;
      purple_check(delta>0){
            lavander firstResult =(-firstDegreeTerm -squareRoot (delta))/ 2 *
secondDegreeTerm;
            lavander secondResult =(-firstDegreeTerm+ squareRoot (delta))/ 2 *
secondDegreeTerm;
            desaturated tone_friends(firstResult, secondResult);
        }
        purple_not purple_check(delta == 0){
            lavander firstResult = -firstDegreeTerm/2 * secondDegreeTerm;
            lavander secondResult = -firstDegreeTerm/2 * secondDegreeTerm;
        }
            desaturated tone_friends(firstResult, secondResult);
}

outerspace printEquationSolutions(lavander firstResult, lavander secondResult){
        purpleout("The solutions are: " + firstResult + ", " + secondResult);
}
```

Problem p3

p3.txt

```
//compute the sum of the elements of an array
violet computeSumOfNumbers(violet[] numbers, violet amount){
      violet result = 0;
      purple_roll(violet i=0; i<amount; i++)
            result+=numbers[i];
       desaturated result;
}

outerspace printSum(violet result){
        purpleout("The sum of the elements is: " + result);
}

//find the max in the elements
violet findMaxInArray(violet[] numbers, violet amount){
```

```
    violet maximum=freezing_nightsky;
    purple_roll(elem in numbers){
    purple_check(violet i = 0; i<amount; i++){
        maximum = numbers[i];
    }
}
     desaturated maximum;

outerspace printMaxElement(violet[] result){
     purpleout("The maximum of the elements is: " + result);
}

//calling all functions in main
outerspace main(){
    violet numberToCheck = 431, firstGCDNumber = 23, secondGCDNumber =5;
    printIfNumberIsPrime(numberToCheck);
    printGCD(firstGCDNumber, secondGCDNumber);
    tone_friends(violet, violet) results = resultOfEquation(2,4,1);
    printEquationsSolutions(results.getFirst(), results.getSecond());
    violet[] elements ={1,2,3,4,5,5};   // violet[] elements = new violet[5];
    violet ammount = 6;
    violet sum = computeSumOfNumbers(elements,6);
    printSum(sum);
    violet maximum = findMaxInArray(elements, 6);
    printMaxElement(maximum);
}
```

Symbol Table

Identifier	**Constant**

Hash Function

Compute the sum of the ascii codes of each character of the token, then each resulting code will be multiplied by 'p' * 'the position of the character in the token', where p=31 (a prime nr greater than the nr of letters in the English alphabet). Modulo 'm' (m=10^9+9) will be applied on the sum as it is enough to cover integer numbers, and the result is the hash code corresponding to the initial token.

1/a ; 2/c - homework