Lab 2

Exercise 1: In my high school, students were given letter grades based on their numerical scores. Here is the table of which numerical scores correspond to which letter grades:

Score	Grade
99-100	A+
96-98	A
93-95	A-
90-92	B+
87-89	B
84-86	B-
81-83	C+
78-80	C
75-77	C-
70-74	D
0-69	F

Write a function `convertScoreToGrade`(n) which takes an `int` numerical score n and returns a string corresponding to the letter grade in my high school.

Exercise 2: Write a function `listOfWords`(s) which takes as input a sentence s and outputs a list containing all the words. We'll assume that the sentence s is just a list of words each separated by a single space. So, for example, `listOfWords`('How are you today') should return the list ['How', 'are', 'you', 'today'].

Exercise 3: Recall, an integer is prime if it is larger than 1 and it has no divisors other than 1 and itself. Write a function `isPrime`(n) which returns `True` if n is prime, and returns `False` otherwise. Then, write a function `listOfPrimes`(n) which returns a list of all primes, in order, between 2 and n — you might want to call the function `isPrime` from within `listOfPrimes`. Notice that if you try to run `listOfPrimes` on a large number, it will take a long time. Is there a way to write `isPrime` so that `listOfPrimes`(n) takes only a few seconds to return its result when n is 1000000?

Exercise 4: Let's call an integer *well-spaced* if when you write down its prime factorization, you don't need to use two adjacent primes. For example, 10 is well-spaced since $10 = 2 \cdot 5$, and $2, 5$ are not adjacent primes since the prime 3 is in between them. However, 6 is not well-spaced, since $6 = 2 \cdot 3$, and $2, 3$ are adjacent primes, and neither is $154 = 2 \cdot 7 \cdot 11$, since $7, 11$ are adjacent primes. Any prime number itself is of course also well-spaced. Write a function `wellSpaced`(n) which returns `True` if n is well-spaced, and returns `False` otherwise. Is 147525307 well-spaced? How about 147914243?

Exercise 5: An integer is said to be a *palindrome* if its digits are the same forward and backwards (not including leading zeroes). For example, 12321 is a palindrome, as is 5. 1231 on the other hand is not a palindrome, and neither is 50 (remember we are not including leading zeroes). Write a function `isPalindrome`(n) which returns `True` if n is a palindrome and `False` otherwise.

Exercise 6: Write a function `nextPalindrome`(n) which returns the smallest integer m larger than n such that m is a palindrome. For example, `nextPalindrome`(9) should return 11, and `nextPalindrome`(12) should return 22.