2.1 In a triangle, the sum of the lengths of two sides is always greater than the length of the third side. Write a function `triangle(a,b,c)` that checks this condition for the sides a, b, c; the result should be `True` or `False`.

2.2 The area A of a triangle whose sides measure a, b, and c can be calculated using the formula (attributed to the Greek mathematician Heron)

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where $s = (a+b+c)/2$ is the semi-perimeter of the triangle. Implement a function `area_triangle(a,b,c)` that calculates the area of a triangle using this formula.

2.3 Write a function `triangle(a,b,c)` that classifies a triangle as *equilateral*, *isosceles*, or *scalene* given the lengths a, b, c of the three sides; the result should be a string. Some examples:

```
>>> triangle(3,3,3)
'equilateral'
>>> triangle(3,2,3)
'isosceles'
>>> triangle(3,4,5)
'scalene'
```

2.4 Write a function `classify(p)` that, given the score p obtained in an exam (from 0 to 100), returns a string with the classification; see the examples and table below.

```
>>> classify(55)
'sufficient'
>>> classify(80)
'very good'
>>> classify(110)
'invalid'
```

< 0 or > 100	invalid
$\geq 0, < 50$	insufficient
$\geq 50, < 70$	sufficient
$\geq 70, < 80$	good
$\geq 80, < 90$	very good
$\geq 90, \leq 100$	excellent

2.5 A year is a *leap year* if it is divisible by 4, except if it is a multiple of 100 and not divisible by 400. Write the function `leap_year(n)` that returns `True` if `n` is a leap year and `False` otherwise.

2.6 Test the function from the previous exercise (2.5) by writing a program that outputs a table of leap years between 2000 and 2030. Check the results with the computer calendar.

2.7 We can count the decimal digits in the representation of a number by performing successive integer divisions by ten. For example: 9733 has 4 digits because we can perform four successive integer divisions by 10 until we get a quotient of zero.
Write a function `digits(n)` that returns the number of decimal digits of n. Hint: use a `while` loop.

2.8 A *proper divisor* of an integer n is a divisor d such that $1 \leq d < n$. The *largest proper divisor* of n is the greatest integer $d < n$ such that d divides n (i.e., the remainder of the division of n by d is zero).

(a) Write the definition of a function `maxdiv(n)` that calculates the largest proper divisor of n.

(b) An integer n is *prime* if $n > 1$ and its largest proper divisor is 1. Write a definition of the function `prime(n)` that tests if n is prime using the previous criterion; the result should be `True` or `False`.

(c) Observe that no proper divisor of n can be greater than $n/2$. Use this fact to restrict the search for the largest proper divisor in your function from item (a) and make the computation more efficient.

2.9 The Leibniz formula for approximating π is:

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \cdots\right) = 4 \times \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Implement the function `leibniz(k)` that calculates the sum of the first `k` terms of this series. Document your function with a *docstring*.