Kernel: SageMath 10.7

Solutions to Practice Exercises

Caution

It's best not to overwrite built-in functions, including **n** , with your own variables. But if you do overwrite something and need to restore it to its orginal functionality, here is how to do that.

In [3]:
```
n = 5 # store a value in the variable n
print(n)
```

Out[3]:
```
5
```

In [4]:
```
n(pi) # this no longer works since n now has the value 5
```

Out[4]:
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 n(pi) # this no longer works since n now has the value 5
TypeError: 'sage.rings.integer.Integer' object is not callable
```

In [5]:
```
reset('n') # this will restore the symbol n
```

In [6]:
```
n(pi) # now the symbol n again refers to the numerical approx function
```

Out[6]:
```
3.14159265358979
```

Exercise 1

Complete the following function **`numRoots`** that determines the *number of real roots* of a quadratic polynomial $ax^2 + bx + c = 0$. Your function should accept as arguments the three coefficients a, b, and c. Your function should return an integer, either 0, 1, or 2.

Hint: Don't forget the quadratic formula!

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

You can determine the number of real roots without actually finding the roots.

In [1]:
```
def numRoots(a, b, c):
    d = b**2 - 4*a*c
    if d > 0:
        return 2
    if d == 0:
        return 1
```

```
    if d < 0:
        return 0
```

Here is some code for testing your **`numRoots`** function:

In [2]:
```
print(numRoots(1,2,1))      # should print 1
print(numRoots(1,3,2))      # should print 2
print(numRoots(1,0,1))      # should print 0
```

Out[2]:
```
1
2
0
```

Exercise 2

Complete the following function that computes an approximation of the number e using a partial sum of the infinite series

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

(This is the Taylor series for e^x evaluated at $x = 1$.) Your function should accept as a parameter the number of terms of the series to add up. You may use the built-in **`factorial`** function.

In [12]:
```
def approxE(numTerms):
    total = 0
    for k in range(numTerms):
        total = total + 1/factorial(k)
    return total.n()
```

Here is some code for testing your **`approxE`** function:

In [13]:
```
print(approxE(2))  # should print 2
print(approxE(5))  # should print 2.708333..
print(approxE(10))  # should print 2.71828...
```

Out[13]:
```
2.00000000000000
2.70833333333333
2.71828152557319
```

Exercise 3

Complete the following function `invertible` that determines whether or not a 2-by-2 matrix is invertible. To do this, represent a matrix as a list of lists. For example, the matrix

$$M = \begin{bmatrix} 3 & 5 \\ -1 & 4 \end{bmatrix}$$

should be stored as `M = [[3, 5],[-1, 4]]` Your function should take the matrix as a single parameter. Your function should return `True` if the matrix is invertible, and `False` otherwise. *Hint: Remember the determinant of a matrix from linear algebra.*

In [5]:
```
def invertible(m):
    det = m[0][0]*m[1][1] - m[0][1]*m[1][0]
    return det != 0
```

Here is some code for testing your `invertible` function:

In [6]:
```
mat1 = [[3, 5],[-1, 4]]
print(invertible(mat1))    # should print True

mat2 = [[0, 0],[1, 1]]
print(invertible(mat2))    # should print False
```

Out[6]:
```
True
False
```

Exercise 4

Write a function that accepts as input a list of numbers and returns the *harmonic mean* of the numbers. If the input sequence is $x_1, x_2, \ldots, x_n$, then the return value should be:

$$\frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}}$$

In [14]:
```
def harmonicMean(seq):
    total = 0
    for x in seq:
        total += 1/x
    return len(seq)/total
```

Try it out:

In [16]:
```
print(harmonicMean([2,2,2]))
harmonicMean([2,3,4,5]).n()
```

Out[16]:
```
2
3.11688311688312
```

Exercise 5

Write a function that accepts as input the coordinates of three points in the plane and returns the area of the triangle whose vertices are those three points. You might compute the area using Heron's Formula:

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

where a, b, and c, are the side lengths, and $s = \frac{a+b+c}{2}$ is the semiperimeter of the triangle.

In [17]:
```
# this function accepts three points p, q, r, specified as pairs of
(x,y)-coordinates
def areaOfTriangle(p, q, r):
    # compute side lengths
    a = sqrt( (p[0]-q[0])**2 + (p[1]-q[1])**2 )
    b = sqrt( (p[0]-r[0])**2 + (p[1]-r[1])**2 )
    c = sqrt( (r[0]-q[0])**2 + (r[1]-q[1])**2 )

    # compute semiperimeter
    s = (a + b + c)/2

    # compute area
    area = sqrt(s*(s - a)*(s - b)*(s - c))

    return area
```

Try it out:

In [19]:
```
# this triangle has area 0.5
areaOfTriangle([1,1],[2,1],[1,2]).n()
```

Out[19]: 0.500000000000000

In [20]:
```
# this triangle has area 14
areaOfTriangle([-2,0],[3,-1],[1,5]).n()
```

Out[20]: 14.0000000000000

Exercise 6

Write a function that computes the Catalan numbers C_n using the following recurrence relation:

$C_0 = 1$ and $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$

In [21]:
```
# recursive solution: not efficient, but it works
def catalan(n):
    if n == 0:
        return 1

    total = 0
    for i in range(n):
        total += catalan(i)*catalan(n-1-i)

    return total
```

Check that we get the Catalan numbers:

In [22]:
```
[catalan(n) for n in range(10)]
```

Out[22]: [1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862]

In [0]:
```

```