Ejercicios 4 y 5 pi

February 1, 2018

```
In [ ]: #Ejercicio 4
```

1. Define una función de Sage suma(N) que devuelve el valor de la suma

```
In [9]: def suma(N):
            res = 0
            for i in xsrange (0,N+1):
                x = ((factorial(2*i))^3)*(42*i + 5)
                y = ((factorial(i))^6)*(16^(3*i + 1))
                res = res + x/y
            return n(res)
```

2. Comprueba que 1/suma(N) se aproxima a cuando N crece.

```
In [22]: %%time
         for i in xsrange (90,101):
             print i, n(1/suma(i)), n(pi) - n(1/suma(i))
```

```
90 3.14159265358979 0.000000000000000
91 3.14159265358979 0.000000000000000
92 3.14159265358979 0.000000000000000
93 3.14159265358979 0.000000000000000
94 3.14159265358979 0.000000000000000
95 3.14159265358979 0.000000000000000
96 3.14159265358979 0.000000000000000
97 3.14159265358979 0.000000000000000
98 3.14159265358979 0.000000000000000
99 3.14159265358979 0.000000000000000
100 3.14159265358979 0.000000000000000
CPU times: user 64 ms, sys: 16 ms, total: 80 ms
Wall time: 69.8 ms
```

3. Intenta estimar, produciendo un programa adecuado, cuantas cifras correctas de se obtienen cada vez que añadimos 10 sumandos más a suma(N), es decir, cada vez que incrementamos N en 10 unidades.

```
In [21]: L = [10*i for i in [1..100]]
         for i in xsrange(1,101):
             print n(pi) - n(1/suma(i))
```

```
-0.000716514035830063
-9.53915551571427e-6
-1.31908705292005e-7
-1.86773174704058e-9
-2.68740585340765e-11
-3.91242593877905e-13
-5.77315972805081e-15
-4.44089209850063e-16
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
```

0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000

```
0.000000000000000
0.000000000000000
0.000000000000000
0.000000000000000
```

```
In [ ]: #Ejercicio 5
```

```
In [29]: def afun(k):
             if (k < 0):
                 return 1
             return (afun(k-1) + bfun(k-1))/2
         def bfun(k):
             if (k < 0):
                 return (sqrt(2)/2)
             return sqrt(afun(k-1)*bfun(k-1))
         def cfun(k):
             return afun(k)^2-bfun(k)^2
         def sfun(k):
             if (k < 0):
                 return 1/2
             return sfun(k-1) - (2^k)*cfun(k)
         def pfun(k):
             return ((2*afun(k)^2)/sfun(k))
```

```
In [41]: %%time
         n(pfun(9))
```

```
CPU times: user 29 s, sys: 164 ms, total: 29.2 s
Wall time: 29 s
```

```
Out[41]: 3.00025111229396
```

Como podemos ver el algoritmo es mucho más lento que el anterior ya que solo para una iteración con un n muy pequeño se tarda más que para 10 con n más grandes