Harmonic Oscillators: Raising and lowering operators

- This exercise introduces symbolic manipulation -- the historical Mathematical stronghold, but here we use the sympy extensions to Python.
- Harmonic oscillator Hamiltonian in one dimension. Ground state ψ_0. Check normalization. Plot ground state, evaluated at ...

In [ ]:
```
from sympy import *
```

Fancy printing of Greek characters and symbols

In [ ]:
```
# init_printing(use_unicode=True) for Anaconda
init_printing(use_latex=True) # for Wakari
```

Tell Python which variables are treated as sympy symbols, which are real and which positive, and how to draw them.

In [ ]:
```
x = Symbol('x', real=True)
m, hbar, omega = symbols('m hbar omega', real=True, positive=True)
```

Define ψ_0 symbolically. Note: we write 1/4 as Rational(1,4) to avoid Python turning it into 0.25 (or worse, 0 as a ratio of integers).

In [ ]:
```
psi0 = (m*omega/(pi*hbar))...
psi0
```

Is ψ_0 normalized?

In [ ]:
```
integrate(..., (x,-oo,oo))
```

To plot sympy expressions, we need to give values for all symbolic variables (except x, which it varies in the plot). We take our constant's from McEuen's bouncing buckyballs (Park et al. "Nanomechanical oscillations in a single-C_60 transistor", Nature 407, 57 (2000)).

We do so by defining a dictionary 'value', where value[hbar] gives the numerical evaluation. Dictionaries in Python are stored as key:value pairs in {}:

In [ ]:
```
amu = 1.660538782e-24;
value = {m:..., hbar:1.054571628251774e-27, omega:..., pi:float(pi)}
```

```
value[pi]
```

To substitute values into a sympy expression using our dictionary 'value', we use expr.subs(value)

In [ ]:
```
a0 = sqrt(...).subs(value)
a0, psi0.subs(value)
```

Now we plot psi0. How do the zero-point fluctuations compare to the size of an atom?

In [ ]:
```
plotting.plot(psi0.subs(value),(x,-4*a0,...));
```

How do the zero-point fluctuations for McEuen's bouncing buckyball compare to the size of an atom?

Define the Hamiltonian: an operator that returns a symbolic function given a symbolic function. sympy.diff(psi,x,2) differentiates psi twice with respect to x.

In [ ]:
```
def H(psi):
    return ...*diff(psi,x,2) + ... * psi
```

Is ψ_0 the ground state? To simplify a sympy expression, we can use sympy.simplify(expr) or expr.simplify()

In [ ]:
```
simplify(H(psi0)/psi0)
```

Momentum operator p. Creation operator $a^\dagger$, Exciting the ground state using creation operators.

In [ ]:
```
def p(psi):
    return ...
p(psi0)
```

In [ ]:
```
def adag(psi):
    return ...
```

In [ ]:
```
adag(psi0)
```

In [ ]:
```
(H(adag(psi0))/...).simplify()
```

In [ ]:
```
adag(adag(psi0))
```

In [ ]:
```
integrate(...,(x,-oo,oo))
```

In [ ]:
```
integrate(...)
```

This isn't the eigenstate ψ_2, since it has the wrong norm. Try taking the norm of higher powers of $(a^\dagger)^n\psi_0$ until you figure out what to divide by to normalize it.

In [ ]:
```
integrate(...)
```

Defining normalized eigenstates. Use recursion, defining ψ_n in terms of ψ_{n-1}, unless $n = 1$. It should be $a^\dagger\psi_{n-1}$ times the constant you figured out that depends on n.

In [ ]:
```
def psi(n):
    if n==0:
        return ...
    return (adag(...)/...).simplify()
```

In [ ]:
```
psi(0)
```

In [ ]:
```
psi(1)
```

In [ ]:
```
psi(2)
```

In [ ]:
```
psi(3)
```

In [ ]:
```
psi(4)
```

In [ ]:
```
integrate(psi(4)**2, (x, -oo, oo))
```

In [ ]:
```
plotting.plot(psi(0).subs(value), psi(1).subs(value), ...,(x,-5*a0,5*a0));
```