# Animation

## Introduction

I am animating the movement of a light showing the corresponding electric and magnetic fields.

In [3]:

```
from vpython import box,sphere, vector,color,rate,canvas
import numpy as np
scene = canvas()
vix = 1
light = sphere (color = color.white, radius = 0.4, make_trail=True, retain=200)
elec = sphere (color = color.green, radius = 0.4, make_trail=True, retain=200)
mag = sphere (color = color.orange, radius = 0.4, make_trail=True, retain=200)


timelist = list(np.arange(0,100,0.1))

def r_light(t):
    x = vix*t
    y = 0
    return x,y
def r_elec(t):
    x = vix*t
    y = 5*np.sin(t)
    return x,y
def r_mag(t):
    x = vix*t
    z = 5*np.sin(t)
    return x,z
scene.camera.follow(light)
scene.range = 30
for t in timelist:
    rate(100)
    x,y = r_light(t)
    light.pos = vector(x,y,0)
    x,y = r_elec(t)
    elec.pos=vector(x,y,0)
    x,z = r_mag(t)
    mag.pos=vector(x,0,z)
```

```
<IPython.core.display.Javascript object>
```

## Conclusion

I enjoyed this a lot more than I expected. It is a good visualization of the fields.

# 4.1 Factorial

## Introduction

Make a factorial function both with ints and floats

```
In [12]:
def int_factorial(num):
    if int(num) == 0:
        return 1
    return int(num) * int_factorial(int(num-1))
int_factorial(4)
def float_factorial(num):
    if num == 0:
        return 1.0
    return float(num)*float_factorial(float(num-1))
print(int_factorial(200),float_factorial(200))
```

```
788657867364790503552363213932185062295135977687173263294742533244359449963403342920
304284011984623904177212138919638830257642790242637105061926624952829931113462857270
763317237396988943922445621451664240254033291864131227428294853277524242407573903240
321257405579568660226031904170324062351700858796178922222789623703897374720000000
0000000000000000000000000000000000000000000 inf
```

### Conclusion

If I use floats, I cannot calculate as high value numbers.

## 4.3 Calculating Derivatives

### Introduction

This will have me use the forward difference definiton of the derivative. Function: $f(x) = x(x-1)$

In [6]:

```
func = lambda x: x*(x-1)
x = 1
print('Analytically, the derivative at x = 1 is 1')
δ = 1e-2
der = (func(x+δ)-func(x))/δ
print(f'When δ = {δ:.1e}, the derivative is {der :.5f}')
δ = 1e-4
der = (func(x+δ)-func(x))/δ
print(f'When δ = {δ:.1e}, the derivative is {der :.5f}')
δ = 1e-6
der = (func(x+δ)-func(x))/δ
print(f'When δ = {δ:.1e}, the derivative is {der :.5f}')
δ = 1e-8
der = (func(x+δ)-func(x))/δ
print(f'When δ = {δ:.1e}, the derivative is {der :.5f}')
δ = 1e-10
der = (func(x+δ)-func(x))/δ
print(f'When δ = {δ:.1e}, the derivative is {der :.5f}')
δ = 1e-12
der = (func(x+δ)-func(x))/δ
print(f'When δ = {δ:.1e}, the derivative is {der :.5f}')
δ = 1e-14
der = (func(x+δ)-func(x))/δ
print(f'When δ = {δ:.1e}, the derivative is {der :.5f}')
```

```
Analytically, the derivative at x = 1 is 1
When δ = 1.0e-02, the derivative is 1.01000
When δ = 1.0e-04, the derivative is 1.00010
When δ = 1.0e-06, the derivative is 1.00000
When δ = 1.0e-08, the derivative is 1.00000
When δ = 1.0e-10, the derivative is 1.00000
When δ = 1.0e-12, the derivative is 1.00009
When δ = 1.0e-14, the derivative is 0.99920
```

### Conclusion

The derivative gets closer to the correct value and then gets farther away.

# 4.4 Calculating Integrals

### Introduction

Take the integral of:

$$\int_{-1}^{1} \sqrt{1-x^2}dx$$

In [10]:

```
%%time
import numpy as np
f = lambda x: np.sqrt(1-x*x)
N = 10000000
lower = -1
upper = 1
x,dx = np.linspace(lower,upper,N,retstep = True)
F = f(x)
value = dx*sum(F)
print(f'When cut into {N} slices, the integral evaluates to be {value:.10f}')
print(np.pi/2)
```

```
When cut into 10000000 slices, the integral evaluates to be 1.5707963267
1.5707963267948966
Wall time: 857 ms
```

### Conclusion

With 10000000 slices, the integral is acurate to 10 decimal places.

# In Class Notes

In [16]:

```
from vpython import *
from numpy import arange
scene = canvas()
maxrate = 10

xi = 0
yi = 0
vix = 3
viy = 20
ax = 0
ay = -9.8

timelist = list(arange(0,5,0.1))
ball = sphere(pos=vector(xi,0,yi))

def r(t):
    x = xi+vix*t+0.5*ax*t*t
    y = yi+viy*t+0.5*ay*t*t
    return x,y
scene.camera.pos = vector(0,0,0)
scene.range = 30
for t in timelist:
    rate(maxrate)
    x,y = r(t)
    ball.pos = vector(x,0,y)
```

```
<IPython.core.display.Javascript object>
```

In [ ]: