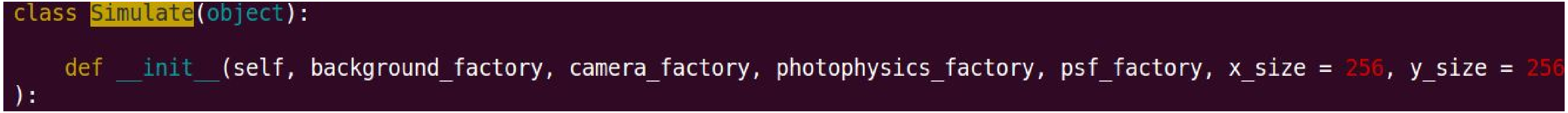
170330 smBIO meeting

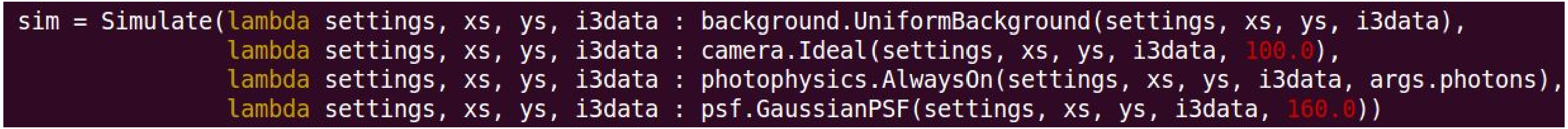
서정훈

simulator/simulate.py

parameters

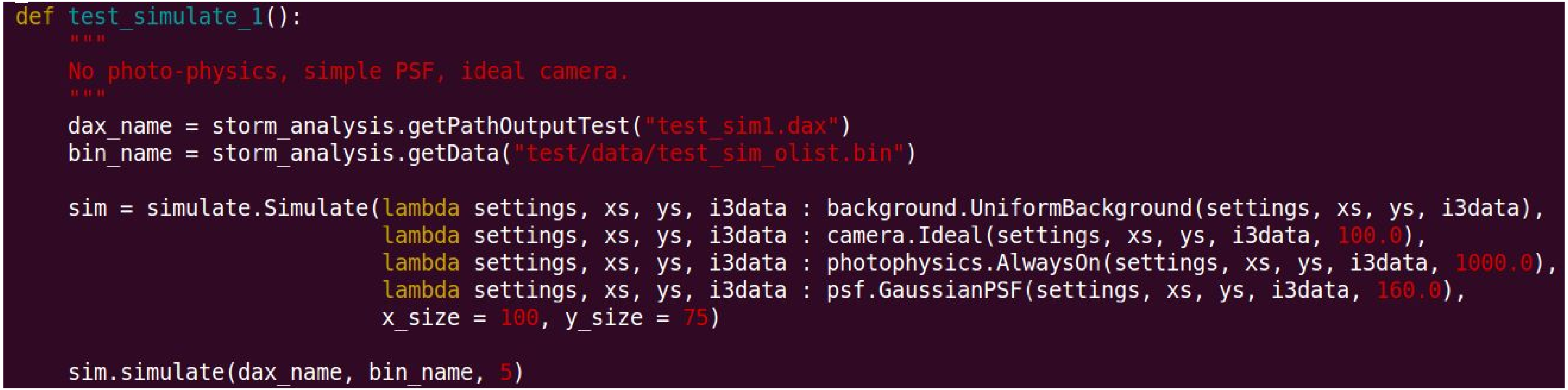


```
class Simulate(object):

    def __init__(self, background_factory, camera_factory, photophysics_factory, psf_factory, x_size = 256, y_size = 256
):
```



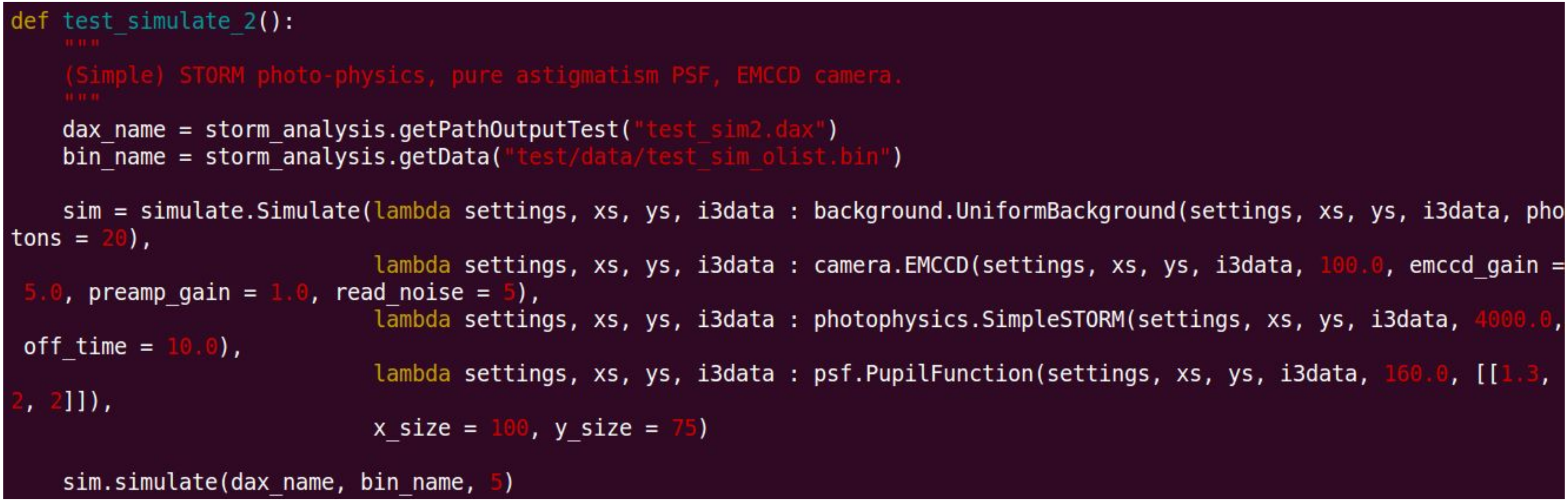
```
sim = Simulate(lambda settings, xs, ys, i3data : background.UniformBackground(settings, xs, ys, i3data),
               lambda settings, xs, ys, i3data : camera.Ideal(settings, xs, ys, i3data, 100.0),
               lambda settings, xs, ys, i3data : photophysics.AlwaysOn(settings, xs, ys, i3data, args.photons),
               lambda settings, xs, ys, i3data : psf.GaussianPSF(settings, xs, ys, i3data, 160.0))
```

test/test_simulator.py



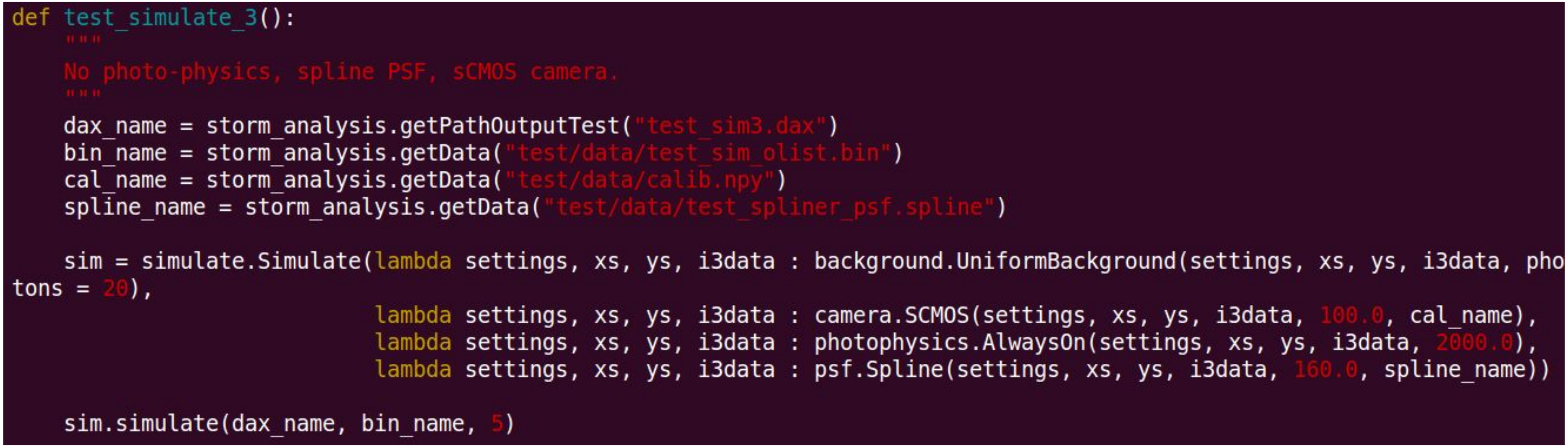
```
def test_simulate_1():
    """
    No photo-physics, simple PSF, ideal camera.
    """
    dax_name = storm_analysis.getPathOutputTest("test_sim1.dax")
    bin_name = storm_analysis.getData("test/data/test_sim_olist.bin")

    sim = simulate.Simulate(lambda settings, xs, ys, i3data : background.UniformBackground(settings, xs, ys, i3data),
                            lambda settings, xs, ys, i3data : camera.Ideal(settings, xs, ys, i3data, 100.0),
                            lambda settings, xs, ys, i3data : photophysics.AlwaysOn(settings, xs, ys, i3data, 1000.0),
                            lambda settings, xs, ys, i3data : psf.GaussianPSF(settings, xs, ys, i3data, 160.0),
                            x_size = 100, y_size = 75)

    sim.simulate(dax_name, bin_name, 5)
```

test/test_simulator.py



```
def test_simulate_2():
    """
    (Simple) STORM photo-physics, pure astigmatism PSF, EMCCD camera.
    """
    dax_name = storm_analysis.getPathOutputTest("test_sim2.dax")
    bin_name = storm_analysis.getData("test/data/test_sim_olist.bin")

    sim = simulate.Simulate(lambda settings, xs, ys, i3data : background.UniformBackground(settings, xs, ys, i3data, pho
tons = 20),
                            lambda settings, xs, ys, i3data : camera.EMCCD(settings, xs, ys, i3data, 100.0, emccd_gain =
 5.0, preamp_gain = 1.0, read_noise = 5),
                            lambda settings, xs, ys, i3data : photophysics.SimpleSTORM(settings, xs, ys, i3data, 4000.0,
 off_time = 10.0),
                            lambda settings, xs, ys, i3data : psf.PupilFunction(settings, xs, ys, i3data, 160.0, [[1.3,
2, 2]]),
                            x_size = 100, y_size = 75)

    sim.simulate(dax_name, bin_name, 5)
```

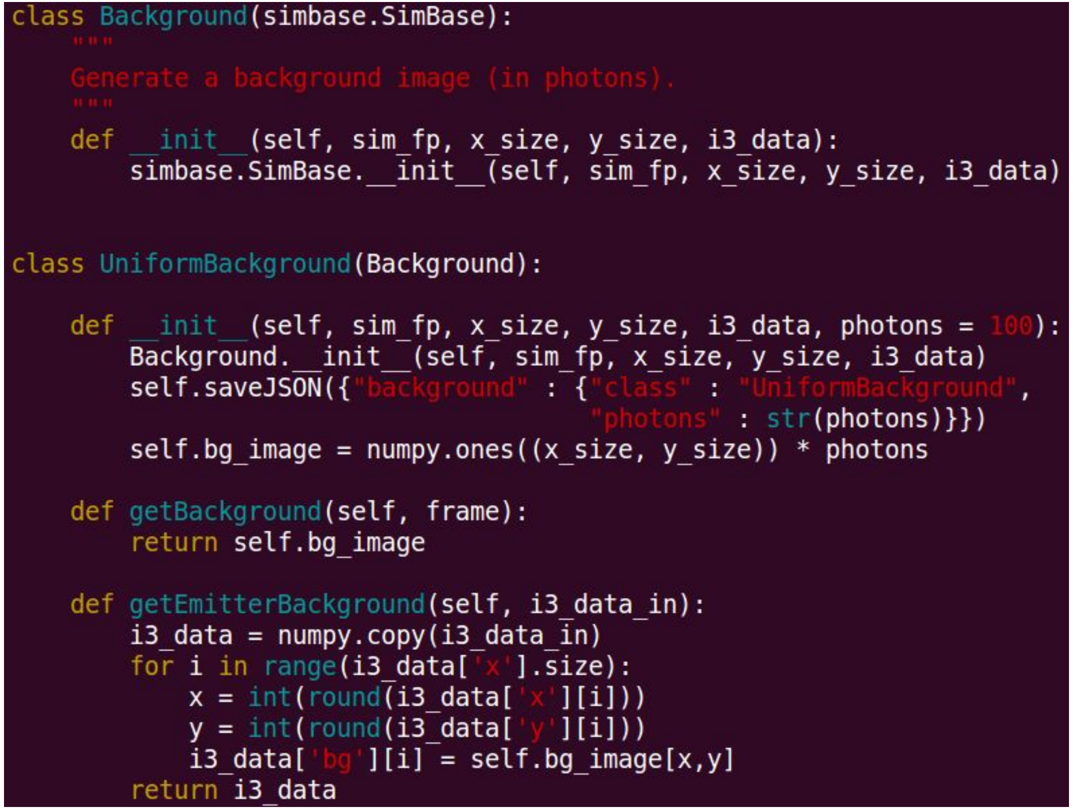
test/test_simulator.py



```
def test_simulate_3():
    """
    No photo-physics, spline PSF, sCMOS camera.
    """
    dax_name = storm_analysis.getPathOutputTest("test_sim3.dax")
    bin_name = storm_analysis.getData("test/data/test_sim_olist.bin")
    cal_name = storm_analysis.getData("test/data/calib.npy")
    spline_name = storm_analysis.getData("test/data/test_spliner_psf.spline")

    sim = simulate.Simulate(lambda settings, xs, ys, i3data : background.UniformBackground(settings, xs, ys, i3data, pho
tons = 20),
                            lambda settings, xs, ys, i3data : camera.SCMOS(settings, xs, ys, i3data, 100.0, cal_name),
                            lambda settings, xs, ys, i3data : photophysics.AlwaysOn(settings, xs, ys, i3data, 2000.0),
                            lambda settings, xs, ys, i3data : psf.Spline(settings, xs, ys, i3data, 160.0, spline_name))

    sim.simulate(dax_name, bin_name, 5)
```

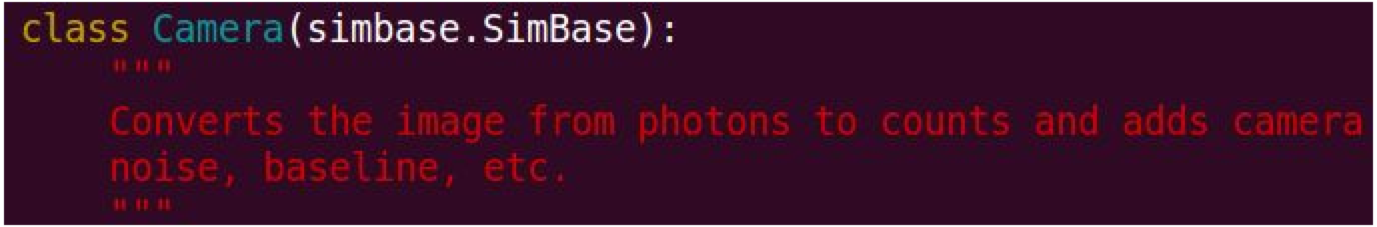
simulator/background.py

1



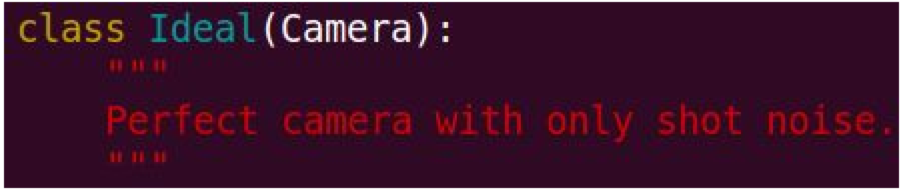
```
class Background(simbase.SimBase):
    """
    Generate a background image (in photons).
    """
    def __init__(self, sim_fp, x_size, y_size, i3_data):
        simbase.SimBase.__init__(self, sim_fp, x_size, y_size, i3_data)


class UniformBackground(Background):

    def __init__(self, sim_fp, x_size, y_size, i3_data, photons = 100):
        Background.__init__(self, sim_fp, x_size, y_size, i3_data)
        self.saveJSON({"background" : {"class" : "UniformBackground",
                                       "photons" : str(photons)}})
        self.bg_image = numpy.ones((x_size, y_size)) * photons

    def getBackground(self, frame):
        return self.bg_image

    def getEmitterBackground(self, i3_data_in):
        i3_data = numpy.copy(i3_data_in)
        for i in range(i3_data['x'].size):
            x = int(round(i3_data['x'][i]))
            y = int(round(i3_data['y'][i]))
            i3_data['bg'][i] = self.bg_image[x,y]
        return i3_data
```

simulator/camera.py



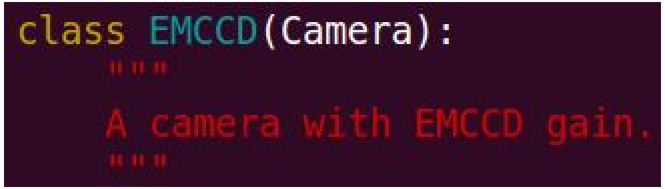
```
class Camera(simbase.SimBase):
    """
    Converts the image from photons to counts and adds camera
    noise, baseline, etc.
    """
```

1



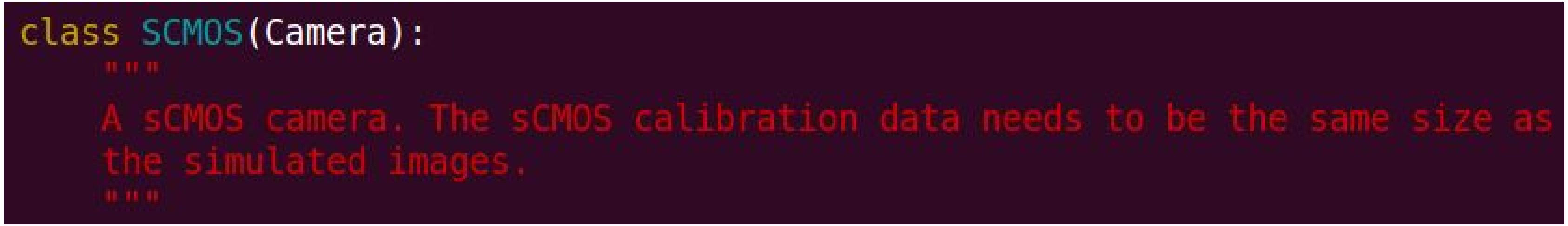
```
class Ideal(Camera):
    """
    Perfect camera with only shot noise.
    """
```

2



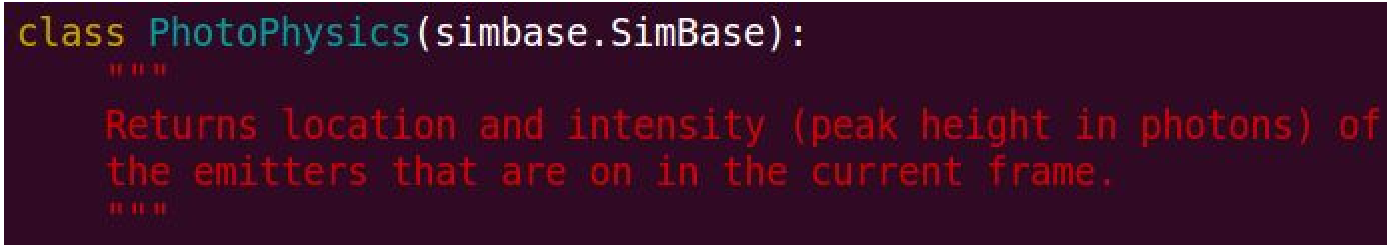
```
class EMCCD(Camera):
    """
    A camera with EMCCD gain.
    """
```

3



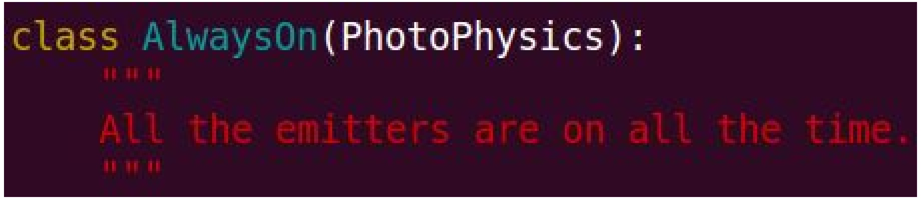
```
class SCMOS(Camera):
    """
    A sCMOS camera. The sCMOS calibration data needs to be the same size as
    the simulated images.
    """
```

simulator/photophysics.py



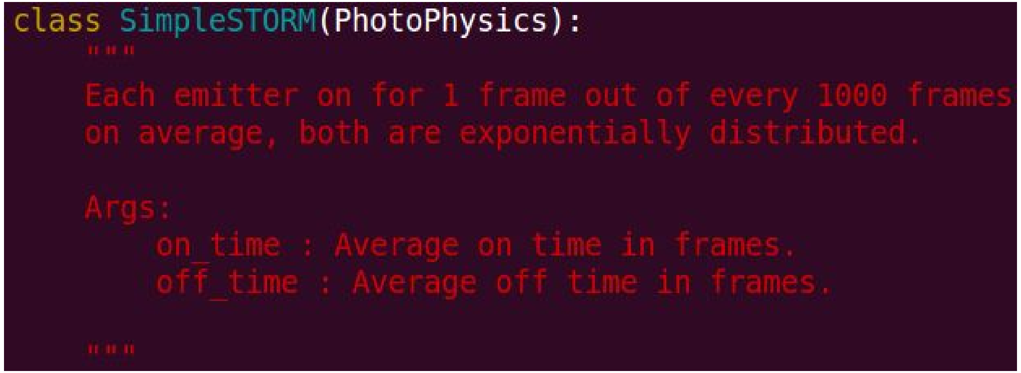
```python
class PhotoPhysics(simbase.SimBase):
    """
    Returns location and intensity (peak height in photons) of
    the emitters that are on in the current frame.
    """
```

1



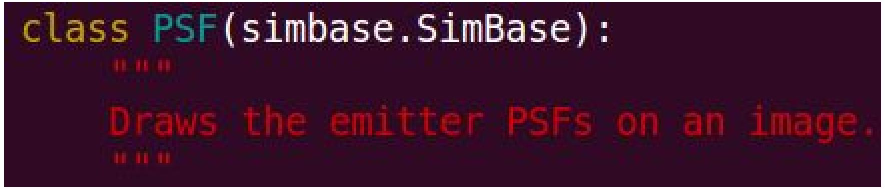
```python
class AlwaysOn(PhotoPhysics):
    """
    All the emitters are on all the time.
    """
```

2



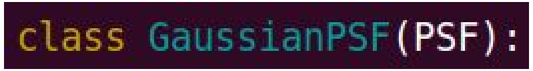
```python
class SimpleSTORM(PhotoPhysics):
    """
    Each emitter on for 1 frame out of every 1000 frames
    on average, both are exponentially distributed.

    Args:
        on_time : Average on time in frames.
        off_time : Average off time in frames.

    """
```

simulator/psf.py



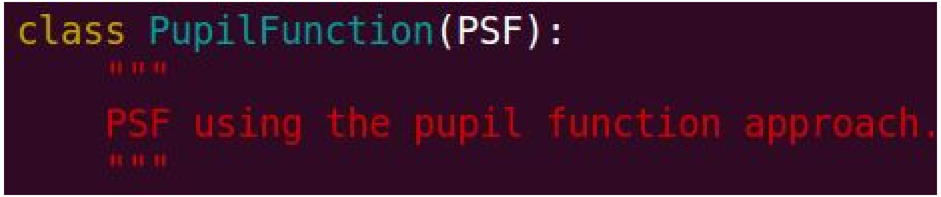
```python
class PSF(simbase.SimBase):
    """
    Draws the emitter PSFs on an image.
    """
```

1



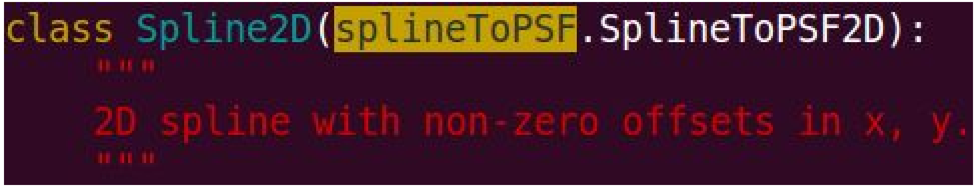
```python
class GaussianPSF(PSF):
```

2



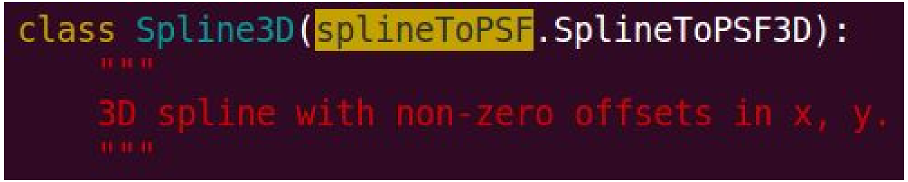
```python
class PupilFunction(PSF):
    """
    PSF using the pupil function approach.
    """
```

3



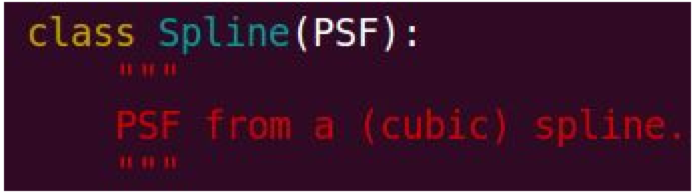
```python
class Spline2D(splineToPSF.SplineToPSF2D):
    """
    2D spline with non-zero offsets in x, y.
    """
```

4



```python
class Spline3D(splineToPSF.SplineToPSF3D):
    """
    3D spline with non-zero offsets in x, y.
    """
```

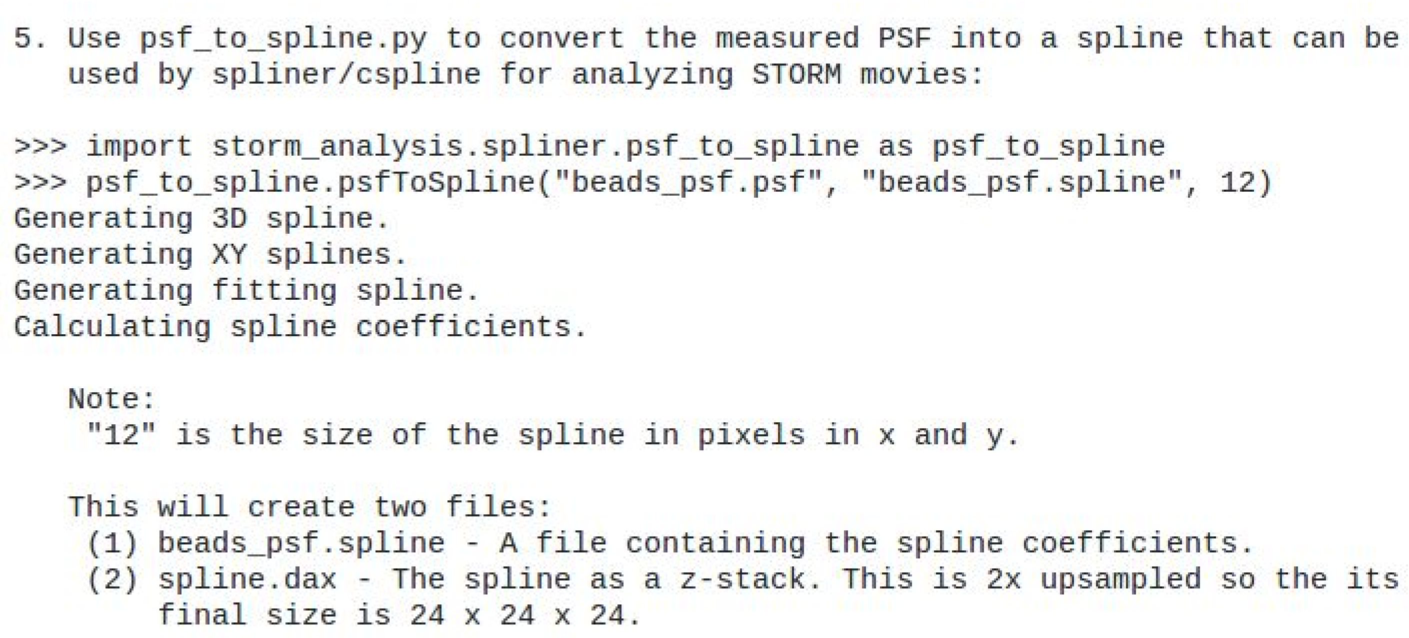
5



```python
class Spline(PSF):
    """
    PSF from a (cubic) spline.
    """
```

spline_file parameter 필요

spliner/psf_to_spline.py



```
5. Use psf_to_spline.py to convert the measured PSF into a spline that can be
   used by spliner/cspline for analyzing STORM movies:

>>> import storm_analysis.spliner.psf_to_spline as psf_to_spline
>>> psf_to_spline.psfToSpline("beads_psf.psf", "beads_psf.spline", 12)
Generating 3D spline.
Generating XY splines.
Generating fitting spline.
Calculating spline coefficients.

   Note:
    "12" is the size of the spline in pixels in x and y.

   This will create two files:
    (1) beads_psf.spline - A file containing the spline coefficients.
    (2) spline.dax - The spline as a z-stack. This is 2x upsampled so the its
        final size is 24 x 24 x 24.
```