The energy of a photon with frequency ν is $E = h\nu$. Because Hawking radiation (HR) is thermal, the energy can be written as $E = h\nu = k_B T$, where k_B is the Boltzmann constant. The inverse wavelength is $c/\lambda = \nu$ with c being the speed of light, so the Hawking radiation temperature is

$$\frac{hc}{\lambda} = k_B T \rightarrow T \sim \frac{hc}{k_B r_s}$$

where we guessed that a typical wavelength for Hawking radiation from a black hole (BH) is roughly the Schwarzchild radius $\lambda \sim r_s = 2GM/c^2$. So a rough estimate of the Hawking radiation temperature is

$$T \sim \frac{hc^3}{2k_B GM}$$

meaning HR from more massive BHs have lower T. The actual expression, from Hawking 1974, is

$$T = \frac{\kappa}{2\pi} = \frac{\hbar c^3}{8\pi k_B G}\frac{1}{M}$$

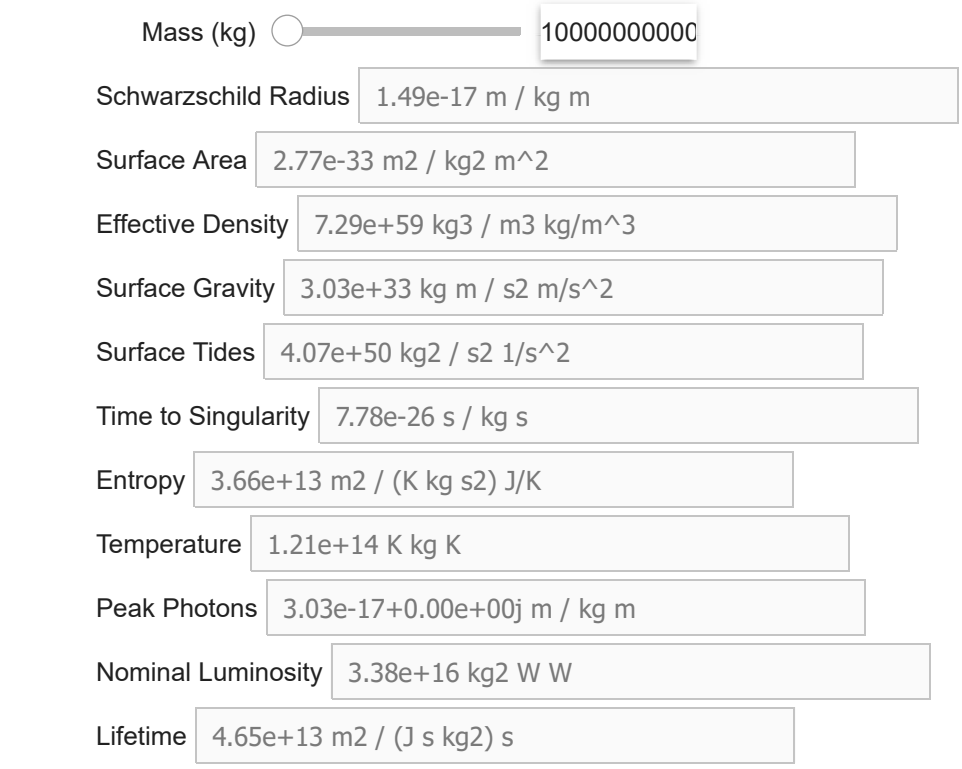
```
import astropy.constants as cnst
import astropy.units as u
import ipywidgets as widgets
import numpy as np
import tkinter as tk

from scipy.special import lambertw
from tkinter import ttk
```

```
class HawkingRadiationCalculator:
    def __init__(
        self, mass,
        c = cnst.c,
        G = cnst.G,
        hbar = cnst.hbar,
        k_b = cnst.k_B,
        sigma_sb = cnst.sigma_sb
    ):
        self.mass = mass*u.kg
        self.G = G
        self.c = c
        self.hbar = hbar
        self.k_b = k_b
        self.sigma_sb = sigma_sb

        self.calculated_quantities = [
            "schwarzschild_radius", "surface_area", "effective_density"
```

```
            "schwarzschild_radius", "surface_area", "effective_density",
            "surface_gravity", "surface_tides", "time_to_singularity",
            "entropy", "temperature", "peak_photons",
            "nominal_luminosity", "lifetime"
        ]

        self.update()

    def update(self):
        self.schwarzschild_radius = self.calculate_schwarzschild_radius()
        self.surface_area = self.calculate_surface_area()
        self.effective_density = self.calculate_effective_density()
        self.surface_gravity = self.calculate_surface_gravity()
        self.surface_tides = self.calculate_surface_tides()
        self.time_to_singularity = self.calculate_time_to_singularity()
        self.entropy = self.calculate_entropy()
        self.temperature = self.calculate_temperature()
        self.peak_photons = self.calculate_peak_photons()
        self.nominal_luminosity = self.calculate_nominal_luminosity()
        self.lifetime = self.calculate_lifetime()

    def calculate_schwarzschild_radius(self):
        return (2*self.G*self.mass) / (self.c**2)

    def calculate_surface_area(self):
        # schwarzschild radius
        Rs = self.calculate_schwarzschild_radius()

        return 4*np.pi*Rs**2

    def calculate_effective_density(self):
        scaling_constant = (3*self.c**6) / (32*np.pi*self.G**3)

        return scaling_constant / self.mass**2

    def calculate_surface_gravity(self):
        scaling_constant = (self.c**4) / (4*self.G)
        return scaling_constant / self.mass

    def calculate_surface_tides(self):
        scaling_constant = (self.c**6) / (4*self.G**2)
        return scaling_constant / self.mass**2

    def calculate_time_to_singularity(self):
        # free-fall time
        return np.pi*self.G*self.mass/self.c**3

    def calculate_entropy(self):
        A = self.calculate_surface_area()
        # dimensionless Bekenstein-Hawking entropy
        S_dimless = A*self.c**3 / (4*self.G*self.hbar)
```

```
        return S_dimless*self.k_b

    def calculate_temperature(self):
        kappa = (self.hbar*self.c**3)/(4*self.k_b*self.G*self.mass)

        return kappa / 2*np.pi

    def calculate_peak_photons(self):
        h = 2*np.pi*self.hbar
        T = self.calculate_temperature()
        E_therm = self.k_b*T

        # argument of Lambert W-function
        arg = -4*np.exp(-4)
        W = lambertw(arg)

        return h*self.c / (E_therm*(W + 4))

    def calculate_nominal_luminosity(self):
        A = self.calculate_surface_area()
        T = self.calculate_temperature()

        return A*self.sigma_sb*T**4

    def calculate_lifetime(self):
        numerator = 5120*np.pi*self.G**2
        denominator = 1.8083*self.hbar*self.c**4

        lifetime = (numerator/denominator)*self.mass**3

        return lifetime

def on_mass_change(change):
    calculator.mass = mass_slider.value
    calculator.update()
    update_outputs()

def update_outputs():
    outputs['Schwarzschild Radius'].value = f"{calculator.schwarzschild_radius:.2e} m"
    outputs['Surface Area'].value = f"{calculator.surface_area:.2e} m^2"
    outputs['Effective Density'].value = f"{calculator.effective_density:.2e} kg/m^3"
    outputs['Surface Gravity'].value = f"{calculator.surface_gravity:.2e} m/s^2"
    outputs['Surface Tides'].value = f"{calculator.surface_tides:.2e} 1/s^2"
    outputs['Time to Singularity'].value = f"{calculator.time_to_singularity:.2e} s"
    outputs['Entropy'].value = f"{calculator.entropy:.2e} J/K"
    outputs['Temperature'].value = f"{calculator.temperature:.2e} K"
    outputs['Peak Photons'].value = f"{calculator.peak_photons:.2e} m"
    outputs['Nominal Luminosity'].value = f"{calculator.nominal_luminosity:.2e} W"
    outputs['Lifetime'].value = f"{calculator.lifetime:.2e} s"
```

```
mass_slider = widgets.FloatSlider(value=1e20, min=1e10, max=1e30, step=1e10, description='M
mass_slider.observe(on_mass_change, names='value')

calculator = HawkingRadiationCalculator(mass_slider.value)

outputs = {
    'Schwarzschild Radius': widgets.Text(value='', disabled=True),
    'Surface Area': widgets.Text(value='', disabled=True),
    'Effective Density': widgets.Text(value='', disabled=True),
    'Surface Gravity': widgets.Text(value='', disabled=True),
    'Surface Tides': widgets.Text(value='', disabled=True),
    'Time to Singularity': widgets.Text(value='', disabled=True),
    'Entropy': widgets.Text(value='', disabled=True),
    'Temperature': widgets.Text(value='', disabled=True),
    'Peak Photons': widgets.Text(value='', disabled=True),
    'Nominal Luminosity': widgets.Text(value='', disabled=True),
    'Lifetime': widgets.Text(value='', disabled=True)
}

output_widgets = [widgets.HBox([widgets.Label(value=key), value]) for key, value in outputs

display(widgets.VBox([mass_slider, *output_widgets]))

update_outputs()
```




```
class HawkingRadiationCalculator:
    def __init__(
        self, mass,
```