```python
# -*- coding: utf-8 -*-
'''
# Info
Title       : Integration With Monte Carlo Method
Version     : 1.0
Developer   : Furkan TOKAC e2016145
Interpreter : Python 3.5.2
License     : GNU General Public License v3.0
Indent      : 4 spaces

# Problem
- Last 4 digits of my id    : 6145.
- Take the remaining        : 6145 % 17 = 8
- So my problem is          : integrate 0 to 2 (x^2 + 5) dx
- Its exact value is        : 12.667

# Explanation
- We create x values according to given range.
- By passing x values to the integral function, we create y values.
- We find correcponding area by using xmin, xmax, ymin, ymax values.
- Its time to use Monte Carlo method. We generate a random point by a given x
range and y range.
- We check if the random point is over the integration function line or under
the integration function line.
- We repeat that process for some time. After all, we'll have a proportion. We
multiply that proportion by the area.
'''

import numpy as np


# This class stores basic Integral data to be used in Integration process
class Integral:
    def __init__(self, xmin=0.0, xmax=2.0, num_of_steps=100):
        # n number x will be produced between xmin and xmax
        self.n = num_of_steps

        self.xmin = xmin
        self.xmax = xmax

        # Get the n number between xmin and xmax as list
        self.xspace = self.get_xspace()

        # According to xspace, produce yspace
        self.yspace = self.get_yspace()

        self.ymin = 0
        self.ymax = max(self.yspace)

    # Function of the integral. It can be changed.
    def f(self, x):
        return x ** 2 + 5.0

    # Return n numbers between xmin and xmax
    def get_xspace(self):
        return np.linspace(self.xmin, self.xmax, self.n)

    # According to xpsace, produce yspace by using f function.
    def get_yspace(self):
        return self.f(self.xspace)
```

```python
# This class handles the ingtegration process for any given Integral
class Integrate:
    def __init__(self, integral=Integral()):
        self.i = integral

    def apply_monte_carlo(self, num_of_points=100, plot=True):
        area = (self.i.xmax - self.i.xmin) * (self.i.ymax - self.i.ymin)
        unders = dict()

        points = {
            "under": {
                "x": list(),
                "y": list()
            },
            "over": {
                "x": list(),
                "y": list()
            }}

        mean = list()

        for _ in range(num_of_points):
            # Generate uniform random x point between xmin and xmax
            x = np.random.uniform(self.i.xmin, self.i.xmax)
            # Generate uniform random y point between ymin and ymax
            y = np.random.uniform(self.i.ymin, self.i.ymax)

            # At this point, I have x and y inside the area that I'm interested

            # Check if the point is over the x axis of the function f
            if y > self.i.f(x):
                points["over"]["x"].append(x)
                points["over"]["y"].append(y)
            else:
                points["under"]["x"].append(x)
                points["under"]["y"].append(y)

        if plot:
            self.plot_monte_carlo(points["over"], points["under"])

        mean = len(points["over"]["x"]) * [0] + len(points["under"]["x"]) * [1]

        # At last, multiply area with the proportion
        return area * np.mean(mean)

    def plot_monte_carlo(self, overs, unders):
        import matplotlib.pyplot as plotter
        plotter.plot(self.i.xspace, self.i.f(self.i.xspace))
        plotter.scatter(overs["x"], overs["y"], color='red')
        plotter.scatter(unders["x"], unders["y"], color='blue')
        plotter.show()

    def replace_integral(self, new_integral):
        self.i = new_integral


my_integral = Integral(xmin=0.0, xmax=2.0, num_of_steps=100)
integrate = Integrate(integral=my_integral)
result = integrate.apply_monte_carlo(num_of_points=100, plot=True)
print("Result of the integration :", result)
```