Programming Exercise 0: Python

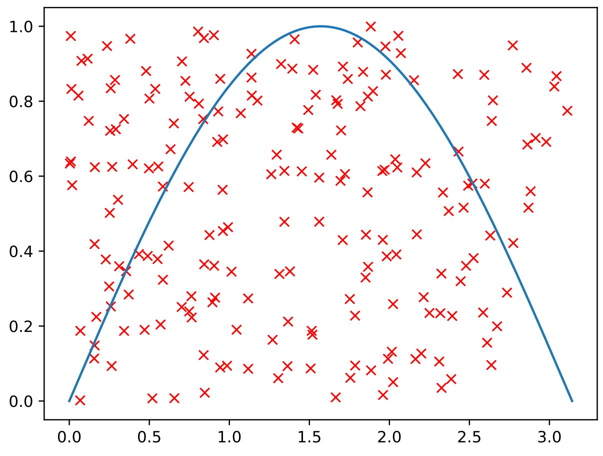
Problem Statement

You must implement a numerical integration algorithm based on the Monte Carlo method. Given a real, integrable function of a single variable $f(x)$, and its integral $F(x)$, the definite integral of $f(x)$ between a and b is given by the expression

$$I = \int_a^b f(x)dx = F(b) - F(a)$$

Since the symbolic calculation of the integral $F(x)$ can be very difficult, numerical methods are used to approximate its value by exploiting the geometric interpretation of the definite integral, which corresponds to the area under the curve $f(x)$ between a and b.

Given a function f(x) that is positive on the interval $x \in [a, b]$ and whose maximum value within that interval is M, we can define a rectangle of area $(b - a) \times M$, as shown in the figure for the interval $[0, 3]$.



The Monte Carlo method for computing the integral consists of randomly generating points (shown in red in the figure) inside this rectangle and approximating the value of the integral by the percentage of points that fall below the function:

$$I \approx \frac{N_{below}}{N_{total}}(b-a)M$$

where N_{below} is the number of randomly generated points (x, y) whose y-coordinate is less than the value of the function $f(x)$ for that value of x, and N_{total} is the total number of points randomly generated within the rectangle.

Implement in Python a function with the following signature:

```python
def integrate_mc(fun, a, b, num_points=10000)
```

that computes the integral of fun between a and b using the Monte Carlo method described above, generating num_points random points to do so. You can check the correctness of the result by comparing it with the result obtained using Python's $scipy.integrate.quad$ function.

It is not necessary for your implementation to solve the problem in a fully general way; it is sufficient for it to compute the result for a function defined by you that is ≥ 0 on the interval $[a, b]$ and that can be applied both to a single number and to a NumPy array. For example:

```python
def square(x):
    return x * x
```

You must implement two versions of the algorithm: an iterative one that performs num_points iterations to compute the result, and another that uses vector operations instead of loops, comparing the execution times obtained with both versions.