```
class DistanceUnit:
    # Base class for all units. Empty class used to verify the unit is of the correct typ
e.
    pass


class Meter(DistanceUnit):
    factor = 1
    abbrv = 'm'


class Foot(DistanceUnit):
    factor = 0.3048
    abbrv = 'ft'


class Distance:
    def __init__(self, value, unit):
        self.value = value
        self.unit = unit
        if not issubclass(self.unit, DistanceUnit):
            raise TypeError("Unit is not a distance unit")

    def __add__(self, other):
        return Distance(self.value + other.value * other.unit.factor / self.unit.factor,
self.unit)

    def __sub__(self, other):
        return Distance(self.value - other.value * other.unit.factor / self.unit.factor,
self.unit)

    def __rmul__(self, factor):
        return Distance(self.value * factor, self.unit)

    __mul__ = __rmul__

    def __truediv__(self, other):
        if isinstance(other, Distance):
            return (self.value * self.unit.factor) / (other.value * other.unit.factor)
        else:
            return Distance(self.value / other, self.unit)

    def __str__(self):
        return str(self.value) + self.unit.abbrv



# Test it out:
meter = Distance(1, Meter)
foot = Distance(1, Foot)

x = 3 * meter
y = 2.3 * foot

print("x =", x)
print("y*2 =", y * 2)
print("x+y =", x + y)
print("(2.5m - 4ft)/5 =", (2.5 * meter - 4 * foot)/5)
print("x/y =", x / y)

# Output:
"""
x = 3m
y*2 = 4.6ft
x+y = 3.70104m
```

```
(2.5m - 4ft)/5 = 0.25616m
x/y = 4.279356384799726
"""
```

```
# Part A
def camel_caser(name):
    # Converts underscore to camel case, e.g. "my_class_name" to "MyClassName"
    return ''.join(x.capitalize() for x in name.split('_'))

x = camel_caser('my_class_name')
print(x)


# Part B: Replace the badly named classes:
inputfile = 'student_code.py'
outputfile = 'readable_code.py'

with open(outputfile, 'w') as fout:
    with open(inputfile, 'r') as fin:
        for line in fin:
            if line.startswith('class'):
                w = line.split(' ')
                w[1] = camel_caser(w[1])
                new_line = ' '.join(w)
                fout.write(new_line)
            else:
                fout.write(line)
```

```
import numpy
import time
import scipy.integrate
import math as m


def f(x):
    return m.cos(x) - m.sin(x**2)

# Part D
n = 10000000
x_span = (0, 5)
x = numpy.linspace(x_span[0], x_span[1], n)


time1 = time.time()

# Part A
x = numpy.linspace(x_span[0], x_span[1], n)
f_x = numpy.zeros(n)
for i in range(len(x)):
    f_x[i] = f(x[i])
val1 = scipy.integrate.trapz(f_x, x)


time2 = time.time()


# Part B
x = numpy.linspace(x_span[0], x_span[1], n)
f_x = numpy.cos(x) - numpy.sin(x**2)
val2 = scipy.integrate.trapz(f_x, x)



time3 = time.time()


# Part C
val3, err = scipy.integrate.quad(f, x_span[0], x_span[1])

time4 = time.time()


print("loop  = {:e}, {}".format(time2 - time1, val1))
print("numpy = {:e}, {}".format(time3 - time2, val2))
print("quad  = {:e}, {}, {}".format(time4 - time3, val3, err))

#########################

# Output:
"""
loop  = 7.192037e-01, -1.4868415558471133
numpy = 6.695676e-02, -1.4868415558471133
quad  = 1.788139e-04, -1.486841555828461, 1.9324843574322625e-09

loop  = 6.175759e+00, -1.4868415558286496
numpy = 7.587125e-01, -1.4868415558286496
quad  = 1.540184e-04, -1.486841555828461, 1.9324843574322625e-09
"""
```

```
class DeInterleave:
    def __init__(self, data):
        self.data = data
        if len(data) % 2 != 0:
            raise Error

    def __iter__(self):
        class DeInterleavIterator:
            def __init__(self, it):
                self.it = it

            def __next__(self):
                return next(self.it), next(self.it)

        return DeInterleavIterator(iter(self.data))


data = [0, 14, 54, 20, 35, 22, 31, 76]

for x, y in DeInterleave(data):
    print('x = ', x, 'and y = ', y)
```

```
class DistanceUnit:
    # Base class for all units. Empty class used to verify the unit is of the correct typ
e.
    pass


class Meter(DistanceUnit):
    factor = 1
    abbrv = 'm'


class Foot(DistanceUnit):
    factor = 0.3048
    abbrv = 'ft'


class Distance:
    def __init__(self, value, unit):
        self.value = value
        self.unit = unit
        if not issubclass(self.unit, DistanceUnit):
            raise TypeError("Unit is not a distance unit")

    def __add__(self, other):
        return Distance(self.value + other.value * other.unit.factor / self.unit.factor,
self.unit)

    def __sub__(self, other):
        return Distance(self.value - other.value * other.unit.factor / self.unit.factor,
self.unit)

    def __rmul__(self, factor):
        return Distance(self.value * factor, self.unit)

    __mul__ = __rmul__

    def __truediv__(self, other):
        if isinstance(other, Distance):
            return (self.value * self.unit.factor) / (other.value * other.unit.factor)
        else:
            return Distance(self.value / other, self.unit)

    def __str__(self):
        return str(self.value) + self.unit.abbrv



# Test it out:
meter = Distance(1, Meter)
foot = Distance(1, Foot)

x = 3 * meter
y = 2.3 * foot

print("x =", x)
print("y*2 =", y * 2)
print("x+y =", x + y)
print("(2.5m - 4ft)/5 =", (2.5 * meter - 4 * foot)/5)
print("x/y =", x / y)

# Output:
"""
x = 3m
y*2 = 4.6ft
x+y = 3.70104m
```

```
(2.5m - 4ft)/5 = 0.25616m
x/y = 4.279356384799726
"""
```