Install required package

```
pip install pycodestyle pep257 pytest
```

```
Collecting pycodestyle
  Downloading https://files.pythonhosted.org/packages/10/5b/88879fb861ab79aef4
     |████████████████████████████████| 51kB 4.8MB/s
Collecting pep257
  Downloading https://files.pythonhosted.org/packages/ec/31/e432e1aa35f692e3f6
Requirement already satisfied: pytest in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: atomicwrites>=1.0 in /usr/local/lib/python3.6/c
Requirement already satisfied: more-itertools>=4.0.0 in /usr/local/lib/python3
Requirement already satisfied: pluggy<0.8,>=0.5 in /usr/local/lib/python3.6/di
Requirement already satisfied: py>=1.5.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.6/dist-
Installing collected packages: pycodestyle, pep257
Successfully installed pep257-0.7.0 pycodestyle-2.6.0
```

Checking code style by pycodestyle and pep257

```
!pycodestyle chisq.py
!pep257 chisq.py
```

```
%%bash
pycodestyle chisq.py
pep257 chisq.py
```

Unit testing

```
!pytest chisq.py
```

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.17, pytest-3.6.4, py-1.8.0, pluggy-0.7.1
rootdir: /content, inifile:
collected 9 items

chisq.py .........                                                       [100

=========================== 9 passed in 0.52 seconds ===========================
```

Attached code

```
import chisq

%timeit -n 1000 chisq.chisq_1(chisq.sample_x, chisq.sample_y)
%timeit -n 1000 chisq.chisq_2(chisq.sample_x, chisq.sample_y)
%timeit -n 1000 chisq.chisq_3(chisq.sample_x, chisq.sample_y)
```

```
1000 loops, best of 3: 234 ms per loop
1000 loops, best of 3: 118 ms per loop
1000 loops, best of 3: 56 ms per loop
```

From the results of timing of single function, one can see the chisq_1 is the slowest, chisq_2 is the second, and chisq_3 is the fastest. The chisq_1 is expected to be the slowest since it used list and for each step it use list comprehesion if need. However, the chisq_1 is easy to read since for each element required, it has a separate line. The chisq_2, instead, using array as a data structure and by using some default numpy function it saves some time for instance, by using 'return_counts=True' one does not need to compute counts separately. Like chisq_1, it is also easy to read except some results are inside for saving some time. The chisq_3 does not use any functions from other packages it uses all for loop for computing needed results. Among all three functions, it is the hardest to read, since using all for loops people need to check each loop to understand.

Overall the first method takes the most time with twice as much as second method and four time as that of the third. However the first one is the easiest to read while the third one take a while to read.

```
"""
```

```
Chi-Square for two samples.

Three functions with different methods.
chisq_1 uses list comprehension, chisq_2 uses array, and chisq_3 uses loop.
"""


import numpy as np
import timeit
import random
from random import randint
import time


def chisq_1(x, y):
    """
    Calculate a Chi-Square for two samples.

    Arguments:
    x:  list
        sample 1
    y:  list
        sample 2

    Returns
    chi_s:  float
            value of Chi-Square result
    """
    n, m = len(x), len(y)
    z = x + y
    u = np.unique(z)
    p = [float(z.count(i)) / float(m+n) for i in u]
    E_k = [n * j for j in p]
    O_k = [x.count(k) for k in u]
    chi_s = sum([float((O_k[i] - E_k[i])**2)/float(E_k[i])
                 for i in range(len(u))])
    return chi_s


def chisq_2(x, y):
    """
    Calculate a Chi-Square for two samples.

    Keyword arguments:
    x:  list sample 1
```

```
    y:  list sample 2

    Returns
    chi_s:  float
            value of Chi-Square result
    """
    n, m = len(x), len(y)
    z = np.append(x, y)
    u, p = np.unique(z, return_counts=True)
    p = np.true_divide(p, (n+m))
    E_k = p*n
    O_k = [x.count(k) for k in u]
    chi_s = np.nansum(np.true_divide(np.square(O_k - E_k), E_k))
    return chi_s


def chisq_3(x, y):
    """
    Calculate a Chi-Square for two samples.

    Keyword arguments:
    x:  list sample 1
    y:  list sample 2

    Returns
    chi_s:  float
            value of Chi-Square result
    """
    n, m = len(x), len(y)
    z = x + y
    u = [z[0]]
    for i in z:
        if i not in u:
            u += [i]
    p = []
    for j in u:
        count = 0
        for k in z:
            if j == k:
                count += 1
        p += [float(count)/float(n+m)]
    E_k = []
    for freq in p:
        E_k += [freq*n]
    O_k = []
```

```
    for r in u:
        count = 0
        for q in x:
            if r == q:
                count += 1
        O_k += [count]
    chi_s = 0
    for s in range(len(u)):
        chi_s += float((O_k[s]-E_k[s])**2) / float(E_k[s])
    return chi_s


# Establishing the simulation
fixtest_x = [1, 1, 2, 2, 2, 3, 4, 4, 4, 5]
fixtest_y = [2, 2, 3, 4, 4, 5, 5, 5]

random.seed(10)
sample_x = [randint(1, 6) for i in range(100000)]
sample_y = [randint(1, 6) for i in range(10000)]

coin_x = [randint(0, 1) for i in range(1000)]
coin_y = [randint(0, 1) for i in range(100)]

simple_x = [1, 1, 1]
simple_y = [1, 1, 0]

%timeit -n 20 chisq_1(sample_x, sample_y)
%timeit -n 20 chisq_2(sample_x, sample_y)
%timeit -n 20 chisq_3(sample_x, sample_y)

"""
From the results of timing of single function, one can see the chisq_1 is the
slowest, chisq_2 is the second, and chisq_3 is the fastest. The chisq_1 is
expected to be the slowest since it used list and for each step it use list
comprehesion if need. However, the chisq_1 is easy to read since for each
element required, it has a separate line. The chisq_2, instead, using array as
a data structure and by using some default numpy function it saves some time
for instance, by using 'return_counts=True' one does not need to compute counts
separately. Like chisq_1, it is also easy to read except some results are
inside for saving some time. The chisq_3 does not use any functions from other
packages it uses all for loop for computing needed results. Among all three
functions, it is the hardest to read, since using all for loops people need to
check each loop to understand.
Overall the first method takes the most time with twice as much as second
method and four time as that of the third. However the first one is the easiest
```

```
to read while the third one take a while to read.
"""


def test_simple_1():
    """
    Test chisq_1 with a simple test.

    Testing chisq_1 by using x = [1, 1, 1], y = [1, 0, 0]
    """
    assert chisq_1(simple_x, simple_y) == 0.6


def test_simple_2():
    """
    Test chisq_2 with a simple test.

    Testing chisq_2 by using x = [1, 1, 1], y = [1, 0, 0]
    """
    assert chisq_2(simple_x, simple_y) == 0.6


def test_simple_3():
    """
    Test chisq_3 with a simple test.

    Testing chisq_3 by using x = [1, 1, 1], y = [1, 0, 0]
    """
    assert chisq_3(simple_x, simple_y) == 0.6


def test_method_1():
    """
    Test chisq_1.

    Testing chisq_1 by using random samples but fixed seed
    """
    assert round(chisq_1(sample_x, sample_y), 5) == 0.36701


def test_method_2():
    """
    Test chisq_2.

    Testing chisq_2 by using random samples but fixed seed
```

```
    """
    assert round(chisq_2(sample_x, sample_y), 5) == 0.36701


def test_method_3():
    """
    Test chisq_3.

    Testing chisq_3 by using random samples but fixed seed
    """
    assert round(chisq_3(sample_x, sample_y), 5) == 0.36701


def test_fixed_1():
    """
    Test chisq_1.

    Testing chisq_1 by using fixed samples
    """
    assert round(chisq_1(fixtest_x, fixtest_y), 5) == 1.43


def test_fixed_2():
    """
    Test chisq_2.

    Testing chisq_2 by using fixed samples
    """
    assert round(chisq_2(fixtest_x, fixtest_y), 5) == 1.43


def test_fixed_3():
    """
    Test chisq_3.

    Testing chisq_3 by using fixed samples
    """
    assert round(chisq_3(fixtest_x, fixtest_y), 5) == 1.43

```

```
20 loops, best of 3: 234 ms per loop
20 loops, best of 3: 118 ms per loop
20 loops, best of 3: 55.9 ms per loop
```