Midterm 1: Sample solutions

Bookmark this page

problem0 (Score: 10.0 / 10.0)

1. Test cell (Score: 5.0 / 5.0)
2. Test cell (Score: 5.0 / 5.0)

Important note! Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

Problem 0: Two algorithms to calculate sample variance

This problem is related to floating-point arithmetic and the *sample variance*, a commonly used measure in statistics. However, the problem should go quickly -- so, if you find yourself spending a lot of time on it, you may be overthinking it or consider returning to it later.

There are two exercises, numbered 0 and 1, which are worth a total of ten (10) points.

Setup

Python has a built-in function, `statistics.variance` (https://docs.python.org/3.5/library/statistics.html#statistics.variance), that computes the sample variance. However, for this problem we want you to implement it from scratch in two different ways and compare their accuracy. (The test codes will use Python's function as a baseline for comparison against your implementations.)

In [1]:
```
# Run this cell.
from statistics import variance

SAVE_VARIANCE = variance # Ignore me
```

A baseline algorithm to compute the sample variance

Suppose we observe n samples from a much larger population. Denote these observations by $x_0, x_1, \ldots, x_{n-1}$. Then the *sample mean* (sample average), $\bar{x}$, is defined to be

$$\bar{x} \equiv \frac{1}{n} \sum_{i=0}^{n-1} x_i.$$

Given both the samples and the sample mean, a standard formula for the (unbiased) *sample variance*, $\bar{s}^2$, is

$$\bar{s}^2 \equiv \frac{1}{n-1} \sum_{i=0}^{n-1} (x_i - \bar{x})^2.$$

Exercise 0 (5 points). Write a function, `var_method_0(x)`, that implements this formula for the sample variance given a list `x[:]` of observed sample values.

> Remember **not** to use Python's built-in `variance()`.

In [2]: Student's answer (Top)

```
def var_method_0(x):
    n = len(x) # Number of samples
    x_bar = sum(x) / n
    return sum([(x_i - x_bar)**2 for x_i in x]) / (n-1)
```

In [3]: Grade cell: `exercise_0_test` Score: 5.0 / 5.0 (Top)

```
# Test cell: `exercise_0_test`

from random import gauss

n = 100000
mu = 1e7  # True mean
sigma = 100.0  # True variance

for _ in range(5): # 5 trials
    X = [gauss(mu, sigma) for _ in range(n)]
    var_py = variance(X)
    try:
        del variance
        var_you_0 = var_method_0(X)
    except NameError as n:
        if n.args[0] == "name 'variance' is not defined":
            assert False, "Did you try to use `variance()` instead of implementing it f
rom scratch?"
        else:
            raise n
    finally:
        variance = SAVE_VARIANCE

    rel_diff = abs(var_you_0 - var_py) / var_py
    print("\nData: n={} samples from a Gaussian with mean {} and standard deviation {}"
.format(n, mu, sigma))
    print("\tPython's variance function computed {}.".format(var_py))
    print("\tYour var_method_0(X) computed {}.".format(var_you_0))
    print("\tThe relative difference is |you - python| / |python| ~= {}.".format(rel_di
ff))
    assert rel_diff <= n*(2.0**(-52)), "Relative difference is larger than expected..."

print("\n(Passed!)")
```

```
Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 100.0
        Python's variance function computed 9985.23864077316.
        Your var_method_0(X) computed 9985.238640773317.
        The relative difference is |you - python| / |python| ~= 1.5848602502326978e-14.

Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 100.0
        Python's variance function computed 10016.330269834216.
        Your var_method_0(X) computed 10016.330269834263.
        The relative difference is |you - python| / |python| ~= 4.721661848014827e-15.

Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 100.0
        Python's variance function computed 10043.093250135487.
        Your var_method_0(X) computed 10043.093250135582.
        The relative difference is |you - python| / |python| ~= 9.418158990319889e-15.

Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 100.0
        Python's variance function computed 9964.086327585743.
        Your var_method_0(X) computed 9964.086327585745.
        The relative difference is |you - python| / |python| ~= 1.8255456082410218e-16.

Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 100.0
        Python's variance function computed 9999.869438680535.
```

```
    Your var_method_0(X) computed 9999.869438680635.
    The relative difference is |you - python| / |python| ~= 1.0004572340518757e-14.

(Passed!)
```

A one-pass algorithm

If there are a huge number of samples, the preceding formula can be slow. The reason is that it makes *two* passes (or loops) over the data: once to sum the samples and another to sum the squares of the samples.

So if there are a huge number of samples and these were stored on disk, for instance, you would have to read each sample from disk twice. (For reference, the cost of accessing data on disk can be orders of magnitude slower than reading it from memory.)

However, there is an alternative that would touch each observation only once. It is based on this formula:

$$\bar{s}^2 = \frac{\left(\sum_{i=0}^{n-1} x_i^2\right) - \frac{1}{n}\left(\sum_{i=0}^{n-1} x_i\right)^2}{n-1}.$$

> Recall that in the original exam, there was a bug in the formula. We had posted the corrected formula on the board. However, in this version of the notebook, the formula you see is *correct*.

In exact arithmetic, the formula above is the same as the first formula. And it can be implemented using only **one pass** of the data, using an algorithm of the following form:

```
temp_sum = 0
temp_sum_squares = 0
for each observation x_i: # Read x_i once, but use twice!
   temp_sum += x_i
   temp_sum_squares += (x_i * x_i)
(calculate final variance)
```

But there is a catch, related to the numerical stability of this scheme.

Exercise 1 (5 points). Implement a function, `var_method_1(x)`, for the one-pass scheme shown above.

The test cell below will run several experiments comparing its accuracy to the accuracy of the first method. **You should observe that the one-pass method can be highly inaccurate!**

In [4]: Student's answer (Top)

```python
def var_method_1(x):
    n = len(x)
    s, s2 = 0.0, 0.0
    for x_i in x:
        s += x_i
        s2 += x_i*x_i
    return (s2 - (s*s/n)) / (n-1)
```

In [5]: Grade cell: `cell-a8e6bdef6dd9d88f` Score: 5.0 / 5.0 (Top)

```python
# Test cell: `exercise_1_test`

from random import gauss
from statistics import variance

n = 100000
mu = 1e7
sigma = 1.0

for _ in range(5): # 5 trials
    X = [gauss(mu, sigma) for _ in range(n)]
```

```
    var_py = variance(X)
    try:
        del variance
        var_you_0 = var_method_0(X)
        var_you_1 = var_method_1(X)
    except NameError as n:
        if n.args[0] == "name 'variance' is not defined":
            assert False, "Did you try to use `variance()` instead of implementing it f
rom scratch?"
        else:
            raise n
    finally:
        variance = SAVE_VARIANCE

    rel_diff_0 = abs(var_you_0 - var_py) / var_py
    rel_diff_1 = abs(var_you_1 - var_py) / var_py
    print("\nData: n={} samples from a Gaussian with mean {} and standard deviation {}"
.format(n, mu, sigma))
    print("\tPython's variance function computed {}.".format(var_py))
    print("\tvar_method_0(X) computed {}, with a relative difference of {}.".format(var
_you_0, rel_diff_0))
    assert rel_diff_0 <= n*(2.0**(-52)), "The relative difference is larger than expect
ed."
    print("\tvar_method_1(X) computed {}, with a relative difference of {}.".format(var
_you_1, rel_diff_1))
    assert rel_diff_1 > n*(2.0**(-52)), "The relative difference is smaller than expect
ed!"

print("\n(Passed!)")
```

```
Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 1.0
        Python's variance function computed 0.9973734569958955.
        var_method_0(X) computed 0.9973734569958903, with a relative difference of 5.2317
89736469506e-15.
        var_method_1(X) computed 0.4710447104471045, with a relative difference of 0.5277
148121969292.

Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 1.0
        Python's variance function computed 0.9893715150466306.
        var_method_0(X) computed 0.9893715150466506, with a relative difference of 2.0198
69597954914e-14.
        var_method_1(X) computed 1.6998569985699856, with a relative difference of 0.7181
179897723948.

Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 1.0
        Python's variance function computed 0.9969426272488455.
        var_method_0(X) computed 0.9969426272488452, with a relative difference of 3.3408
833997466393e-16.
        var_method_1(X) computed 0.34816348163481636, with a relative difference of 0.650
7687883749084.

Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 1.0
        Python's variance function computed 0.9975157724036159.
        var_method_0(X) computed 0.9975157724036211, with a relative difference of 5.2310
433179064595e-15.
        var_method_1(X) computed 1.884178841788418, with a relative difference of 0.88887
12278185809.

Data: n=100000 samples from a Gaussian with mean 10000000.0 and standard deviation 1.0
        Python's variance function computed 1.0052280601297094.
        var_method_0(X) computed 1.0052280601297054, with a relative difference of 3.9760
16037728629e-15.
        var_method_1(X) computed 1.6998569985699856, with a relative difference of 0.6910
162638621974.

(Passed!)
```

Fin! If you've reached this point and all tests above pass, you are ready to submit your solution to this problem. Don't forget to save you work prior to submitting.

problem1 (Score: 10.0 / 10.0)

1. Test cell (Score: 2.0 / 2.0)
2. Test cell (Score: 3.0 / 3.0)
3. Test cell (Score: 1.0 / 1.0)
4. Test cell (Score: 4.0 / 4.0)

Important note! Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

Problem 1: Boozy Containers

This problem is a review of Python's built-in *container data structures (https://docs.python.org/3/tutorial/datastructures.html)*, or simply, *containers*. These include lists, sets, tuples, and dictionaries.

Below, there are four (4) exercises, numbered 0-3, which relate to basic principles of using containers. They are worth a total of ten (10) points.

The dataset: Student alcohol consumption

The data files for this problem pertain to a study of student alcohol consumption (https://www.kaggle.com/uciml/student-alcohol-consumption) and its effects on academic performance. The following cell downloads these files (if they aren't already available).

In [1]:
```
import requests
import os
import hashlib
import io

def download(file, url_suffix=None, checksum=None):
    if url_suffix is None:
        url_suffix = file

    if not os.path.exists(file):
        if os.path.exists('.voc'):
            url = 'https://cse6040.gatech.edu/datasets/{}'.format(url_suffix)
        else:
            url = 'https://github.com/cse6040/labs-fa17/raw/master/datasets/{}'.format(ur
l_suffix)
        print("Downloading: {} ...".format(url))
        r = requests.get(url)
        with open(file, 'w', encoding=r.encoding) as f:
```

```
            f.write(r.text)

    if checksum is not None:
        with io.open(file, 'r', encoding='utf-8', errors='replace') as f:
            body = f.read()
            body_checksum = hashlib.md5(body.encode('utf-8')).hexdigest()
            assert body_checksum == checksum, \
                "Downloaded file '{}' has incorrect checksum: '{}' instead of '{}'".forma
t(file, body_checksum, checksum)

    print("'{}' is ready!".format(file))

datasets = {'student-mat.csv': '83dc97a218a3055f51cfca1e76b29036',
            'student-por.csv': 'c5fe725d1436c73e5bc16fe8c2618bf9'}

for filename, checksum in datasets.items():
    download(filename, url_suffix='ksac/{}'.format(filename), checksum=checksum)

print("\n(All data appears to be ready.)")
```

```
Downloading: https://github.com/cse6040/labs-fa17/raw/master/datasets/ksac/student-por.cs
v ...

'student-por.csv' is ready!
Downloading: https://github.com/cse6040/labs-fa17/raw/master/datasets/ksac/student-mat.cs
v ...

'student-mat.csv' is ready!

(All data appears to be ready.)
```

Here is some code to show you the first few lines of each file:

In [2]:

```
def dump_head(filename, max_lines=5):
    from sys import stdout
    from math import log10, floor
    lines = []
    with open(filename) as fp:
        for _ in range(max_lines):
            lines.append(fp.readline())
    stdout.write("\n=== First {} lines of: '{}' ===\n\n".format(max_lines, filename))
    for line_num, text in enumerate(lines):
        fmt = "[{{:0{}d}}] {{}}".format(floor(log10(max_lines))+1)
        stdout.write(fmt.format(line_num, text))
    stdout.write('.\n.\n.\n')

dump_head('student-mat.csv');
dump_head('student-por.csv');
```

```
=== First 5 lines of: 'student-mat.csv' ===

[0] school,sex,age,address,famsize,Pstatus,Medu,Fedu,Mjob,Fjob,reason,guardian,traveltime
,studytime,failures,schoolsup,famsup,paid,activities,nursery,higher,internet,romantic,fam
rel,freetime,goout,Dalc,Walc,health,absences,G1,G2,G3
[1] GP,F,18,U,GT3,A,4,4,at_home,teacher,course,mother,2,2,0,yes,no,no,no,yes,yes,no,no,4,
3,4,1,1,3,6,5,6,6
[2] GP,F,17,U,GT3,T,1,1,at_home,other,course,father,1,2,0,no,yes,no,no,no,yes,yes,no,5,3,
3,1,1,3,4,5,5,6
[3] GP,F,15,U,LE3,T,1,1,at_home,other,other,mother,1,2,3,yes,no,yes,no,yes,yes,yes,no,4,3
,2,2,3,3,10,7,8,10
[4] GP,F,15,U,GT3,T,4,2,health,services,home,mother,1,3,0,no,yes,yes,yes,yes,yes,yes,yes,
3,2,2,1,1,5,2,15,14,15
.
.
.

=== First 5 lines of: 'student-por.csv' ===

[0] school,sex,age,address,famsize,Pstatus,Medu,Fedu,Mjob,Fjob,reason,guardian,traveltime
,studytime,failures,schoolsup,famsup,paid,activities,nursery,higher,internet,romantic,fam
rel,freetime,goout,Dalc,Walc,health,absences,G1,G2,G3
[1] GP,F,18,U,GT3,A,4,4,at_home,teacher,course,mother,2,2,0,yes,no,no,no,yes,yes,no,no,4,
3,4,1,1,3,4,0,11,11
[2] GP,F,17,U,GT3,T,1,1,at_home,other,course,father,1,2,0,no,yes,no,no,no,yes,yes,no,5,3,
3,1,1,3,2,9,11,11
```

```
5,1,1,3,2,5,11,11
[3] GP,F,15,U,LE3,T,1,1,at_home,other,other,mother,1,2,0,yes,no,no,no,yes,yes,yes,no,4,3,
2,2,3,3,6,12,13,12
[4] GP,F,15,U,GT3,T,4,2,health,services,home,mother,1,3,0,no,yes,no,yes,yes,yes,yes,yes,3
,2,2,1,1,5,0,14,14,14
.
.
.
```

What is this data? Each of the two file fragments shown above is in *comma-separated values (CSV) format*. Each encodes a data table about students, their alcohol consumption, test scores, and other demographic attributes, as explained later.

The first row is a list of column headings, separated by commas. These are the attributes.

Each subsequent row corresponds to the data for a particular student.

The numbers in brackets (e.g., `[0]`, `[1]`) are only line numbers and do not exist in the actual file.

Of the two files, the first is data for a math class; the second file is for a Portuguese language class.

The `read_csv()` function, below, will read in these files using Python's `csv` module (https://docs.python.org/3/library/csv.html).

The important detail is that this function returns two lists: a list of the column names, `header`, and a list of lists, named `data_rows`, which holds the rows. In particular, `data_rows[i]` is a list of the values that appear in the `i`-th data row, which you can see by comparing the sample output below to the raw file data above.

In [3]:
```python
def read_csv(filename):
    with open(filename) as fp:
        from csv import reader
        data_rows = list(reader(fp))
    header = data_rows.pop(0)
    return (header, data_rows)

# Read the math class data
math_header, math_data_rows = read_csv('student-mat.csv')

# Read the Portuguese class data
port_header, port_data_rows = read_csv('student-por.csv')

# Print a sample of the data
def print_sample(header, data, num_rows=5):
    from math import floor, log10
    fmt = "Row {{:0{}d}}: {{}}".format(floor(log10(num_rows))+1)
    string_rows = [fmt.format(i, str(r)) for i, r in enumerate(data[:num_rows])]
    print("--> Header ({} columns): {}".format(len(header), header))
    print("\n--> First {} of {} rows:\n{}".format(num_rows,
                                                len(data),
                                                '\n'.join(string_rows)))

print("=== Math data ===\n")
print_sample(math_header, math_data_rows)
```

```
=== Math data ===

--> Header (33 columns): ['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu'
, 'Fedu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 's
choolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', '
famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences', 'G1', 'G2', 'G3']

--> First 5 of 395 rows:
Row 0: ['GP', 'F', '18', 'U', 'GT3', 'A', '4', '4', 'at_home', 'teacher', 'course', 'moth
er', '2', '2', '0', 'yes', 'no', 'no', 'no', 'yes', 'yes', 'no', 'no', '4', '3', '4', '1'
, '1', '3', '6', '5', '6', '6']
Row 1: ['GP', 'F', '17', 'U', 'GT3', 'T', '1', '1', 'at_home', 'other', 'course', 'father
', '1', '2', '0', 'no', 'yes', 'no', 'no', 'no', 'yes', 'yes', 'no', '5', '3', '3', '1',
'1', '3', '4', '5', '5', '6']
Row 2: ['GP', 'F', '15', 'U', 'LE3', 'T', '1', '1', 'at_home', 'other', 'other', 'mother'
, '1', '2', '3', 'yes', 'no', 'yes', 'no', 'yes', 'yes', 'yes', 'no', '4', '3', '2', '2',
'3', '3', '10', '7', '8', '10']
Row 3: ['GP', 'F', '15', 'U', 'GT3', 'T', '4', '2', 'health', 'services', 'home', 'mother
', '1', '3', '0', 'no', 'yes', 'yes', 'yes', 'yes', 'yes', 'yes', 'yes', '3', '2', '2', '
1', '1', '5', '2', '15', '14', '15']
```

```
Row 4: ['GP', 'F', '16', 'U', 'GT3', 'T', '3', '3', 'other', 'other', 'home', 'father', '
1', '2', '0', 'no', 'yes', 'yes', 'no', 'yes', 'yes', 'no', 'no', '4', '3', '2', '1', '2'
, '5', '4', '6', '10', '10']
```

The function only separates the fields by comma; it doesn't do any additional postprocessing. So all the data elements are treated as strings, even though you can see that some are clearly numerical values. You'll need this fact in **Exercise 3**.

Exercise 0 (2 points). Complete the function, `lookup_value(col_name, row_id, header, data_rows)`, to look up a particular value in the data when stored as shown above. In particular, the parameters of the function are

- `col_name`: Name of the column, e.g., `'school'`, `'address'`, `'freetime'`.
- `row_id`: The desired row number, starting at 0 (the first *data* row).
- `header`, `data_rows`: The list of column names and data rows, respectively.

For example, consider the math data shown above. Then,

```
lookup_value('age', 0, math_header, math_data_rows) == '18'
    lookup_value('G2', 3, math_header, math_data_rows) == '14'
```

> **Hint.** Consider `list.index()` (https://docs.python.org/3/tutorial/datastructures.html).

In [4]: Student's answer (Top)

```python
def lookup_value(col_name, row_id, header, data_rows):
    assert col_name in header, "{} not in {}".format(col_name, header)
    assert 0 <= row_id < len(data_rows)
    col_id = header.index(col_name)
    return data_rows[row_id][col_id]
```

In [5]: Grade cell: `exercise_0_test` Score: 2.0 / 2.0 (Top)

```python
# Test cell: `exercise_0_test`

print("Checking examples from above...")
assert lookup_value('age', 0, math_header, math_data_rows) == '18'
assert lookup_value('G2', 3, math_header, math_data_rows) == '14'

print("Checking some random examples...")

# Generate random test cases
if False:
    for _ in range(5):
        from random import sample, randint
        col_name = sample(math_header, 1)[0]
        row_id = randint(0, len(math_data_rows)-1)
        value = lookup_value(col_name, row_id, math_header, math_data_rows)
        print("assert lookup_value('{}', {}, math_header, math_data_rows) == '{}'".form
at(col_name, row_id, value))

    for _ in range(5):
        from random import sample, randint
        col_name = sample(port_header, 1)[0]
        row_id = randint(0, len(port_data_rows)-1)
        value = lookup_value(col_name, row_id, port_header, port_data_rows)
        print("assert lookup_value('{}', {}, port_header, port_data_rows) == '{}'".form
at(col_name, row_id, value))

assert lookup_value('famsize', 143, math_header, math_data_rows) == 'LE3'
assert lookup_value('absences', 198, math_header, math_data_rows) == '24'
assert lookup_value('G3', 246, math_header, math_data_rows) == '13'
assert lookup_value('guardian', 175, math_header, math_data_rows) == 'mother'
assert lookup_value('paid', 362, math_header, math_data_rows) == 'no'
assert lookup_value('romantic', 87, port_header, port_data_rows) == 'no'
assert lookup_value('famsup', 246, port_header, port_data_rows) == 'yes'
assert lookup_value('Walc', 294, port_header, port_data_rows) == '1'
assert lookup_value('famsize', 431, port_header, port_data_rows) == 'GT3'
```

```
assert lookup_value('studytime', 224, port_header, port_data_rows) == '4'

print("\n(Passed!)")
```

```
Checking examples from above...
Checking some random examples...

(Passed!)
```

Exercise 1 (3 points). Suppose we wish to extract a list of all values stored in a given column of the table. Complete the function, `lookup_column_values(col, header, data_rows)`, which takes as input the column name `col`, list of column names `header`, and rows `data_rows`, and returns a list of all the values stored in that column.

For example, the first five entries of the returned list when reference the `'age'` column of the math class data should satisfy:

```
values = lookup_column_values('age', math_header, math_data_rows)
  assert values[:5] == ['18', '17', '15', '15', '16']
```

In [6]: Student's answer (Top)

```
def lookup_column_values(col, header, data_rows):
    assert col in header
    col_id = header.index(col)
    values = [row[col_id] for row in data_rows]
    return values
```

In [7]: Grade cell: `exercise_1_test` Score: 3.0 / 3.0 (Top)

```
# Test cell: `exercise_1_test`

# The example:
values = lookup_column_values('age', math_header, math_data_rows)

print("First five values of 'age' column in the math data:")
print(values[:5])
assert values[:5] == ['18', '17', '15', '15', '16']

if False:
    from random import sample
    for col in sample(math_header, 5):
        values = lookup_column_values(col, math_header, math_data_rows)
        print("assert ''.join(lookup_column_values('{}', math_header, math_data_rows))
== '{}'".format(col, ''.join(values)))
    for col in sample(port_header, 5):
        values = lookup_column_values(col, port_header, port_data_rows)
        print("assert ''.join(lookup_column_values('{}', port_header, port_data_rows))
== '{}'".format(col, ''.join(values)))

print("\nSpot-checking some additional cases...")
assert ''.join(lookup_column_values('activities', math_header, math_data_rows)) == 'non
onoyesnoyesnononoyesnoyesyesnononoyesyesyesyesnonoyesyesyesnononoyesyesnoyesyesyesnoyes
yesyesyesyesyesnoyesnoyesyesnoyesnoyesnonononononoyesyesyesyesnoyesyesyesyesyesyesyesnono
nononononoyesyesyesyesnoyesnoyesnonoyesyesnonoyesyesnonoyesnoyesyesyesyesnoyesnoyesyesyes
noyesnonoyesyesyesyesyesyesnoyesyesyesyesyesnonoyesyesyesnonoyesnoyesyesnoyesnononoyesn
oyesnoyesnoyesyesnonononononononononoyesyesnonoyesnoyesnonoyesnoyesnoyesyesnononononoyesyesye
syesyesyesyesyesyesyesyesyesyesyesnoyesyesyesnonononoyesyesyesnoyesnoyesnonoyesyesnonoyes
nonoyesnoyesyesnonoyesnonononononoyesyesyesnonoyesyesyesnoyesyesyesyesyesyesyesnoyesyesnoyes
noyesnoyesyesnononononononononononoyesnoyesyesnoyesyesnonoyesyesyesyesyesyesyesnoyesyesnoye
snononoyesyesyesyesnoyesyesnononononoyesyesyesnonononoyesnoyesnononononononoyesyesyesyesnoyes
yesyesnononononoyesyesyesnononoyesnonoyesyesnoyesnoyesnoyesyesyesnononononoyesyesyesnon
onononoyesnonononononoyesnoyesnonononoyesnoyesnononononononoyesyesyesyesnononoyesnoyesyesye
syesnononoyesyesnoyesnononono'
assert ''.join(lookup_column_values('reason', math_header, math_data_rows)) == 'coursec
ourseotherhomehomereputationhomehomehomehomereputationreputationcoursecoursehomehomerep
utationreputationcoursehomereputationothercoursereputationcoursehomehomeotherhomehomeho
mereputationcoursecoursehomeotherhomereputationcoursereputationhomehomecoursecoursecour
secoursehomereputationhomeothercourseotherothercourseotherotherreputationreputationhome
courseothercoursereputationhomereputationcoursereputationcoursereputationreputationrepu
```

```
tationcoursereputationreputationhomehomecoursereputationhomecoursecoursehomereputationh
omehomereputationcoursereputationreputationreputationhomereputationhomehomereputationho
mereputationcoursereputationcourseotherothercoursehomecoursereputationcoursehomehomeoth
ercoursereputationhomecoursereputationcoursereputationhomecoursereputationcoursehomecou
rsecoursehomehomehomecoursereputationcoursecoursecoursecoursecoursecoursecoursecourseco
ursecoursecoursecoursereputationcoursecoursehomecoursehomecoursecoursecoursecoursecours
ereputationhomecoursecoursereputationcoursecoursecoursecoursecoursecoursecoursecourseco
ursecoursehomehomereputationcoursereputationreputationhomereputationcoursereputationrep
utationothercoursehomehomereputationreputationreputationotherothercoursereputationhomec
oursecourseotherreputationhomecoursehomehomehomereputationhomereputationcoursereputatio
nreputationhomecourseotherhomereputationreputationhomereputationhomeotherreputationrepu
tationhomehomecoursereputationreputationotherhomehomereputationcoursereputationcourseco
ursereputationcoursereputationreputationhomereputationhomehomecoursereputationcoursecou
rsecoursecoursecoursecoursecourseothercourseothercoursereputationothercoursecoursecours
ereputationreputationhomecoursehomecoursecoursehomehomereputationotherreputationreputat
ionreputationhomereputationhomehomereputationcoursehomehomereputationcoursehomehomerepu
tationhomecoursereputationotherreputationreputationreputationhomereputationreputationre
putationreputationhomereputationhomereputationhomehomehomereputationreputationhomereput
ationcoursereputationreputationreputationhomeothercoursereputationhomereputationcoursec
oursecoursecoursecoursecoursecoursecoursehomecoursereputationcoursecoursecoursecourseco
ursehomehomecoursecoursehomehomehomehomehomehomehomecourseothercoursecoursereputati
oncoursehomecoursecoursehomehomecourseotherreputationhomecoursecourseotherothercourseco
ursecourseotherreputationcourseotherhomeotherhomecoursereputationhomecoursecoursehomere
putationhomeotherhomeotherhomeotherreputationcoursecoursecoursecoursecoursecoursecourse
course'
assert ''.join(lookup_column_values('guardian', math_header, math_data_rows)) == 'mothe
rfathermothermotherfathermothermothermothermothermothermotherfatherfathermotherothermot
hermothermothermotherfathermotherfathermothermothermothermothermothermothermothermother
mothermothermothermothermotherfathermothermothermothermothermotherotherfatherfatherfath
ermothermothermothermotherfathermothermotherfathermothermothermothermothermotherfatherm
othermotherfatherfathermotherfathermothermothermothermotherfatherfathermothermothermoth
ermothermothermothermothermothermotherfathermothermothermotherfatherfathermothermotherf
athermothermothermotherfathermothermothermothermothermothermothermothermotherfathermoth
ermothermotherfathermotherfatherfathermothermotherfathermothermothermotherfatherfatherf
atherfatherfathermotherfatherfathermothermotherfathermotherothermothermotherfathermothe
rfatherfathermothermothermotherothermothermotherfatherfathermothermothermotherfatherfat
hermothermothermothermothermotherothermothermothermothermothermothermotherfathermotherm
othermotherfathermothermotherfatherfathermothermotherfathermothermothermothermothermoth
ermotherfathermothermothermothermotherfathermothermothermothermothermothermothermotherm
othermotherothermotherfatherfathermotherfathermothermothermothermothermothermothermothe
rmothermothermothermothermotherothermothermothermothermothermothermotherfatherfathermot
hermothermothermothermothermothermothermotherfathermothermotherfatherfathermotherfather
mothermotherfathermothermotherfathermothermothermothermotherotherfathermothermothermoth
ermothermotherothermothermothermothermothermothermothermotherfatherfathermothermothermo
thermothermothermothermotherfathermotherotherfathermothermothermothermothermothermother
fathermothermothermothermothermothermotherfathermothermothermothermothermothermothermot
hermothermothermotherotherotherfathermothermotherfathermotherfatherotherotherotherother
fatherotherotherotherotherotherotherothermothermotherfathermothermothermothermotherfath
erfathermothermothermothermothermothermotherfathermotherothermothermotherothermothermot
hermotherotherfathermotherfathermothermothermothermothermothermotherothermothermotherot
hermotherfatherfathermotherfatherfathermothermothermothermotherfathermothermothermother
fatherfatherotherfathermothermothermothermotherothermothermothermotherfathermotherfathe
rmotherfathermothermothermothermothermothermotherothermotherothermothermotherfather'
assert ''.join(lookup_column_values('famsup', math_header, math_data_rows)) == 'noyesno
yesyesyesnoyesyesyesyesyesyesyesyesyesyesyesyesnonoyesnoyesyesyesyesnoyesyesyesyesyesno
yesyesyesyesyesyesyesyesyesyesnoyesyesnoyesyesyesyesnoyesnonoyesyesyesyesyesyesnoyesnoy
esyesyesyesyesyesnoyesnoyesyesnonoyesyesyesnoyesnoyesnoyesyesnoyesyesnoyesyesyesyesyesy
esnoyesyesyesyesyesyesnoyesyesyesyesnoyesnononoyesyesnoyesnonoyesyesyesnononoyesyesnoye
syesyesnoyesyesnoyesnonoyesyesyesnoyesyesyesyesyesnononoyesyesyesnonoyesnoyesnoyesnonon
oyesnonoyesyesyesyesyesnoyesnononoyesyesyesnoyesyesyesyesyesnoyesnonononoyesnononoyesye
syesyesyesyesyesyesyesyesnoyesyesyesyesyesyesyesnonoyesyesyesyesyesnoyesyesnonoyesyesno
noyesyesyesyesnononononononoyesnonoyesnononoyesnonoyesnononoyesyesyesnonoyesyesnonoyesyes
nonoyesyesyesyesyesyesyesyesyesnonoyesnonononoyesnoyesnoyesnoyesyesyesyesnoyesyesyesyesye
syesyesnoyesyesyesyesyesnoyesnoyesnoyesnoyesnonoyesyesyesyesyesyesyesyesnononoyesyesyesyes
yesnonononononoyesyesyesnoyesnoyesyesnonoyesyesyesnoyesnoyesyesyesyesnonononononoyesyesnoye
snoyesyesyesnoyesnononononoyesnonoyesnononononononoyesnoyesnoyesnononono'
assert ''.join(lookup_column_values('nursery', math_header, math_data_rows)) == 'yesnoy
esyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesnoyesyesyesyesnoyes
yesnonoyesyesyesyesyesnonoyesyesyesyesyesyesnoyesyesyesyesyesyesyesyesyesyesyesyesnoyes
yesyesyesnoyesyesnoyesyesnoyesyesyesyesyesyesyesnonoyesyesnoyesyesnoyesyesnoyesyesyesyesy
esyesnoyesnoyesyesyesnoyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesnoyesyesyesye
syesyesnoyesyesyesnoyesyesyesyesyesnoyesyesyesyesyesyesyesyesyesnoyesyesyesyesyesyesyes
yesyesnonoyesyesyesyesyesyesyesyesnoyesyesnoyesnoyesyesnoyesyesnoyesyesyesyesyesyesyesye
syesnoyesyesyesnoyesyesyesyesyesyesyesyesyesnoyesnoyesyesyesyesyesyesyesyesyesyesyesyes
yesyesyesyesyesyesyesnoyesnonoyesyesyesyesyesyesyesyesnoyesyesyesyesyesnoyesnoyesyesnoy
```

```
esnoyesnonononoyesyesyesyesyesyesnononoyesyesyesyesyesyesyesnonoyesyesnoyesyesyesyes
yesyesyesnonoyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesyesnoyesyesyesyesyesnonoye
snononoyesyesyesnoyesyesyesyesnoyesyesnoyesyesnoyesyesyesnoyesyesyesyesyesyesyesyesy
esyesyesyesyesyesnoyesyesyesyesnoyesyesyesnoyesyesnoyesyesnoyesyesyesyesnoyesyesyesy
esyesnoyesyesnonononoyesyesyesyesyesnoyesyesyesnononoyes'
assert ''.join(lookup_column_values('sex', port_header, port_data_rows)) == 'FFFFFMMFMM
FFMMMFFFMMMMMMFFMMMMMMMMMFMMFFFMMMFFFMMFFFMFFFFMMFFFFFFFMFFFMMFMFMMFMFMMFMFFFFMMFFFFMFM
FFFMMMFMFFMMFMFFMMMMMMMFMFMFMFFMMFFFFFFMFMFMFMFFFMFFMFMMFFMFFFFFMMMMFMFMMFMMMMMMMMMMFFF
MMMFFMFMMMMMMFFFMMMFMFFMMMFMMFFFFFFFFFFFFFMFMFFFMFFFFFMFFFMMFFMMMMMMFFFFMMFFFFFMFFFMMMM
FMFFFMFMMMMMMMMMMFFFMFFFMMMFFFMMFFMMMFMFFFFMFFFFMFFFMMMMFFFFMFFMMMFFMMFFFMFMFFFMMMFFFFF
FFFMFFFFFFMFFFFFFFFMMMFFFMMFFMFFFFFFFMMFFFMMFFFFFFMMFMFMFFMFMMFFFFFFFFFMMFFFFFFFFFFMMMMM
MMMFFFMFFFFFFMFFFFFFFMFMFMMFFFMMFFMFFFFFFFFFFFFMFFFFFFFFMFFMMMFFFFFMFFFFFFFFFFMFFFMFMFMFFMM
MFMMMMFFFFFMMMFMFFMMMMFFMFMMMMFMMMMMMMMMFFMFMMMMMFFFFFFFFFFFFFFMFFMMMFFFFFFFFFFFFMFFFMFF
FFFMMFFFMFFFFFFFFFMMMMMFFFFFFMM'
assert ''.join(lookup_column_values('absences', port_header, port_data_rows)) == '42600
602002000061022600022680240200440428168001446242004002080200000200062020400101402422464
662602246042120216010448244121046461420624020084010862124122420210840626240424220026616
006260014640084220100000400000048440032860010660160084163002402161424204150106216104001
209442020000020040181614266410421416844807422101010742128084420022042444202822220860841
216102286615628001022182012100000020000200000000000222228289042140221222400000020420020
680680400618044001041660119020218510513101054110640050000404020040748086320403004021002
690110420420211040201601660412250020001282000800001902800806042201051802001181128501228
002000243542000600202049648912888404200050300320003008544002005200540202006400000038404
41110404000044466 4'
assert ''.join(lookup_column_values('G2', port_header, port_data_rows)) == '11111314131
212131612141213121417131481213121310111112111211111514121211141312131111151011111217121
213149121213141513151610131312151291015119111311111113911119111212111115109111313141213
111213139161310161010141015141110179141514111314131213119111014121191012121391111149131
111119101491181311131171210111211911910139138810101088698871017121691791314111299111012
181315139141391211111012151311151014121513139111513121212131111169101312101213916818111
015121313139121610131087881410111397914141512141511141191016101289128118131210121510101
212111316101112151011121281611111291216131714161291212158101011131713121312181315171618
191515131417181513716181015101012121615121512101112151412129108101311181414141810121412
111091514119131313141412161212161413121516111312111417151710151018171411171013111210101
781111106149107111116091391091110161315111088101411109101214111189131212121114811138107
991010107101099911810991314111316712814131191191717976681614578108691414991110107151211
141311101191191191511101813149111114109131299130910701188959111110991098861178610108131
213181718011131381201001811101101411101214131715121413101681051011817118141691871587179
91115121011'
assert ''.join(lookup_column_values('age', port_header, port_data_rows)) == '1817151516
161617151515151515151516161617161515161615161515161615151515161515161515161515151616151616 1
515161515151515161515151616161616151615161515161515161615151617161515151515151516151616161615
161615151616161616161515151515151615161516161515161516171515151616161515191616151817151 71
61616151517161818161616151515151616151617151515151516151816151917151718161616171715161 7
171616161616161716161716171617161617171617161617171617171616171716171616161717161716161
617191716181717171817171717161816181717181717171716161617161618181818171717161717181717
171717161617171717181617172218161816181617171817191817181819181717201818181718171817181
818191818171817171818181719181817171818171817181718181817171817191818181817171717192019
181817171818171819181817171717181919181717171718181717171717181918171818181817171817181
817181818181717171917211818171718212019171818191820181616151516151616171715161515161715
161715151615151716151616161616151617161615161515161815161516151616161515161516171917151
616171618191818191617161616171616171617161917201717161717181616181616181616161618161717
161716171716161615151516161717161717171716171716171917171616171718171816171616181819181
619161720181819171819181918171717181717181817181818171818181718181818191717181819181718
17171818181817171818171818181717181819181817181817191818171819181817 18'
assert ''.join(lookup_column_values('famsize', port_header, port_data_rows)) == 'GT3GT3
LE3GT3GT3LE3LE3GT3LE3GT3GT3GT3LE3GT3GT3GT3GT3GT3GT3LE3GT3GT3LE3LE3GT3GT3GT3GT3LE3GT3GT3
GT3GT3LE3GT3GT3LE3GT3GT3GT3LE3LE3GT3GT3LE3LE3LE3GT3GT3GT3LE3LE3LE3GT3LE3GT3GT3GT3LE3GT3
GT3GT3LE3GT3LE3LE3GT3GT3LE3LE3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3LE3LE3GT3GT3LE3GT3GT3
LE3GT3GT3LE3GT3LE3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3LE3LE3GT3GT3LE3GT3GT3GT3GT3
GT3GT3GT3GT3LE3GT3GT3GT3LE3GT3GT3GT3GT3GT3LE3GT3LE3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3GT3LE3
GT3GT3GT3GT3GT3LE3GT3LE3GT3LE3GT3GT3GT3GT3LE3LE3GT3LE3GT3GT3GT3LE3GT3GT3GT3GT3LE3LE3GT3
LE3GT3LE3GT3GT3GT3GT3GT3GT3GT3LE3GT3LE3LE3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3GT3LE3GT3GT3GT3
GT3GT3GT3GT3LE3GT3LE3GT3LE3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3LE3LE3GT3LE3GT3GT3
GT3GT3GT3GT3GT3GT3GT3LE3LE3GT3LE3GT3GT3GT3LE3GT3LE3GT3GT3LE3LE3GT3GT3GT3GT3LE3GT3GT3LE3
GT3LE3LE3LE3LE3LE3LE3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3GT3GT3GT3GT3LE3LE3LE3GT3LE3
GT3LE3GT3LE3GT3GT3LE3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3GT3GT3GT3GT3LE3GT3GT3GT3LE3GT3GT3GT3
GT3GT3LE3GT3LE3LE3LE3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3LE3LE3GT3GT3LE3GT3LE3LE3LE3GT3GT3GT3
LE3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3GT3GT3GT3GT3GT3LE3LE3LE3GT3LE3GT3GT3LE3GT3LE3GT3GT3GT3
GT3GT3GT3GT3LE3GT3GT3GT3GT3GT3LE3GT3GT3GT3LE3GT3GT3LE3GT3GT3GT3GT3GT3GT3GT3GT3LE3GT3LE3
LE3LE3GT3LE3LE3LE3GT3GT3LE3GT3GT3GT3LE3GT3GT3GT3GT3GT3GT3GT3GT3LE3LE3GT3GT3GT3GT3LE3LE3LE3
GT3GT3GT3GT3GT3LE3LE3GT3LE3GT3LE3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3GT3GT3LE3GT3GT3
GT3GT3GT3GT3GT3GT3LE3LE3LE3GT3LE3GT3GT3GT3GT3GT3LE3GT3LE3GT3GT3LE3GT3GT3GT3GT3GT3LE3
GT3LE3GT3GT3GT3GT3LE3LE3GT3LE3GT3LE3GT3LE3GT3GT3GT3GT3LE3GT3LE3GT3GT3LE3GT3LE3GT3GT3LE3
LE3LE3GT3LE3GT3LE3GT3GT3GT3LE3LE3GT3LE3LE3LE3GT3GT3GT3GT3LE3GT3GT3LE3GT3GT3LE3GT3GT3
```

```
LE3GT3GT3LE3GT3GT3GT3GT3LE3LE3GT3GT3LE3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3GT3LE3GT3GT3
LE3GT3GT3LE3GT3LE3GT3LE3GT3GT3GT3LE3GT3LE3GT3GT3LE3GT3GT3GT3LE3LE3GT3GT3GT3LE3LE3GT3GT3
GT3GT3GT3GT3LE3LE3LE3LE3GT3LE3GT3GT3LE3GT3GT3GT3LE3GT3GT3LE3GT3GT3LE3GT3GT3GT3GT3GT3GT3
GT3GT3GT3GT3GT3LE3GT3LE3LE3'

print("\n(Passed!)")
```

```
First five values of 'age' column in the math data:
['18', '17', '15', '15', '16']

Spot-checking some additional cases...

(Passed!)
```

Exercise 2 (1 points). Suppose we wish to get all *unique* values in a given column. Complete the function, `get_unique_values(col, header, data_rows)`, so that it returns these values, as a **list**.

> The ordering of unique values in your output does not matter.

In [8]: Student's answer (Top)

```python
def get_unique_values(col, header, data_rows):
    assert col in header
    return list(set(lookup_column_values(col, header, data_rows)))
```

In [9]: Grade cell: `exercise_2_test` Score: 1.0 / 1.0 (Top)

```python
# Test cell: `exercise_2_test`

print("Checking ages...")
ages = get_unique_values('age', math_header, math_data_rows)
assert type(ages) is list, "`get_unique_values()` should return a list!"
assert len(ages) == 8
assert all([str(a) in ages for a in range(15, 23)])

print("\nSpot checking some additional cases...")

if False: # Generate test cases
    from random import sample
    for col in sample(math_header, 5):
        values = get_unique_values(col, math_header, math_data_rows)
        print("""
values = get_unique_values('{}', math_header, math_data_rows)
assert len(values) == {}
for v in {}:
    assert v in values, "'{{}}' should be in the output, but isn't.".format(v)
""".format(col, len(values), values))

values = get_unique_values('Fedu', math_header, math_data_rows)
assert len(values) == 5
for v in ['4', '1', '3', '0', '2']:
    assert v in values, "'{}' should be in the output, but isn't.".format(v)

values = get_unique_values('traveltime', math_header, math_data_rows)
assert len(values) == 4
for v in ['4', '1', '3', '2']:
    assert v in values, "'{}' should be in the output, but isn't.".format(v)

values = get_unique_values('paid', math_header, math_data_rows)
assert len(values) == 2
for v in ['no', 'yes']:
    assert v in values, "'{}' should be in the output, but isn't.".format(v)

values = get_unique_values('school', math_header, math_data_rows)
assert len(values) == 2
for v in ['MS', 'GP']:
    assert v in values, "'{}' should be in the output, but isn't.".format(v)
```

```
values = get_unique_values('sex', math_header, math_data_rows)
assert len(values) == 2
for v in ['F', 'M']:
    assert v in values, "'{}' should be in the output, but isn't.".format(v)

print("\n(Passed!)")
```

```
Checking ages...

Spot checking some additional cases...

(Passed!)
```

A simple analysis task. The column `'Dalc'` contains the student's self-reported drinking frequency during the weekday. The values are 1 (very low amount of drinking) to 5 (very high amount of drinking); if your function above works correctly then you should see that in the output of the following cell:

In [10]:
```
print("Unique values of 'Dalc':", get_unique_values('Dalc', math_header, math_data_rows))
```

```
Unique values of 'Dalc': ['4', '5', '1', '2', '3']
```

Similarly, `Walc` is the self-reported drinking frequency on the same scale, but for the *weekend* (instead of *weekday*).

Now, suppose we wish to know whether there is a relationship between these drinking frequencies and the final math grade, which appears in column `'G3'` (on a scale of 0-20, where 0 is a "low" grade and 20 is a "high" grade).

Exercise 3 (4 points). Create a dictionary named **`dw_avg_grade`** that will help with this analysis. For this exercise, we only care about the math grades (`math_data_rows`); you can ignore the Portuguese grades (`port_data_rows`).

For your dictionary, `dw_avg_grade`, the keys and values should be defined as follows.

1. Each key should be a tuple, `(a, b)`, where `a` is the `'Dalc'` rating and `b` is the `'Walc'` rating. You should convert these ratings from strings to integers. You only need to consider keys that actually occur in the data.
2. Each corresponding value should be the average test score, rounded to two decimal places.

> **Hint.** To get you started, we've used your `lookup_column_values()` function to extract the relevant columns for analysis. From there, consider breaking this problem up into several parts:
>
> 1. Counting the number of occurrences of (`Dalc`, `Walc`) pairs.
> 2. Summing the grades for each pair.
> 3. Dividing the latter by the former to get the mean. Use [`round()` (https://docs.python.org/3/library/functions.html#round)](https://docs.python.org/3/library/functions.html#round) to do the rounding.

In [11]: Student's answer (Top)

```
from collections import defaultdict # Optional, but might help

# Relevant data to analyze:
Dalc_values = lookup_column_values('Dalc', math_header, math_data_rows)
Walc_values = lookup_column_values('Walc', math_header, math_data_rows)
G3_values = lookup_column_values('G3', math_header, math_data_rows)

# Your task: Build `dw_avg_grade` per the problem statement.

dw_counts = defaultdict(int)
dw_sums = defaultdict(int)
for d, w, g3 in zip(Dalc_values, Walc_values, G3_values):
    key = (int(d), int(w))
    value = int(g3)
    dw_counts[key] += 1
    dw_sums[key] += int(g3)

dw_avg_grade = {}
for key in dw_sums.keys():
```

```
    dw_avg_grade[key] = round(dw_sums[key] / dw_counts[key], 1)
```

In [12]: Grade cell: exercise_3_test Score: 4.0 / 4.0 (Top)

```
# Test cell: `exercise_3_test`

assert type(dw_avg_grade) is dict or type(dw_avg_grade) is defaultdict, "'dw_avg_grade'
should be a dictionary (or default dictionary)."

print("Your computed results:")
descending = sorted(dw_avg_grade.items(), key=lambda x: -x[1])
for (dalc, walc), g3 in descending:
    print("(Dalc={}, Walc={}): {}".format(dalc, walc, g3))

print("\nSpot checking some of these values...")
test_cases = [((3, 2), 12.0), ((1, 1), 10.8), ((2, 1), 0.0), ((3, 3), 9.75), ((3, 5), 1
0.0)]
for (d, w), g3 in test_cases:
    your_g3 = dw_avg_grade[(d, w)]
    msg = "({}, {}) == {}, but it should be {}".format(d, w, your_g3, g3)
    assert abs(your_g3 - g3) < 0.1, msg

print("\n(Passed!)")
```

```
Your computed results:
(Dalc=3, Walc=2): 12.0
(Dalc=4, Walc=4): 11.3
(Dalc=3, Walc=4): 11.2
(Dalc=1, Walc=3): 11.1
(Dalc=4, Walc=2): 11.0
(Dalc=1, Walc=1): 10.8
(Dalc=1, Walc=5): 10.8
(Dalc=5, Walc=5): 10.7
(Dalc=2, Walc=3): 10.7
(Dalc=1, Walc=2): 10.4
(Dalc=1, Walc=4): 10.3
(Dalc=3, Walc=5): 10.0
(Dalc=4, Walc=5): 9.8
(Dalc=3, Walc=3): 9.8
(Dalc=2, Walc=5): 9.2
(Dalc=2, Walc=2): 8.7
(Dalc=2, Walc=4): 8.3
(Dalc=4, Walc=3): 5.0
(Dalc=2, Walc=1): 0.0

Spot checking some of these values...

(Passed!)
```

Fin! If you've reached this point and all tests above pass, you are ready to submit your solution to this problem. Don't forget to save you work prior to submitting.

problem2 (Score: 10.0 / 10.0)

1. Test cell (Score: 2.0 / 2.0)
2. Test cell (Score: 2.0 / 2.0)
3. Test cell (Score: 1.0 / 1.0)
4. Test cell (Score: 3.0 / 3.0)

5. Test cell (Score: 2.0 / 2.0)

Important note! Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

Problem 2: DNA Sequence Analysis

This problem is about strings and regular expressions. It has three (3) exercises, numbered 0-2. They are worth a total of ten (10) points.

In [1]:
```
import re # You'll need this module
```

DNA Sequence Analysis

Your friend is a biologist who is studying a particular DNA sequence. The sequence is a string built from an alphabet of four possible letters, `A`, `G`, `C`, and `T`. Biologists refer to each of these letters a *base*.

Here is an example of a DNA fragment as a string of bases.

In [2]:
```
dna_seq = 'ATGGCAATAACCCCCCGTTTCTACTTCTAGAGGAGAAAAGTATTGACATGAGCGCTCCCGGCACAAGGGCCAAAGAAG
TCTCCAATTTCTTATTTCCGAATGACATGCGTCTCCTTGCGGGTAAATCACCGACCGCAATTCATAGAAGCCTGGGGGAACAGATAGGT
CTAATTAGCTTAAGAGAGTAAATCCTGGGATCATTCAGTAGTAACCATAAACTTACGCTGGGGCTTCTTCGGCGGATTTTTACAGTTAC
CAACCAGGAGATTTGAAGTAAATCAGTTGAGGATTTAGCCGCGCTATCCGGTAATCTCCAAATTAAAACATACCGTTCCATGAAGGCTA
GAATTACTTACCGGCCTTTTCCATGCCTGCGCTATACCCCCCACTCTCCCGCTTATCCGTCCGAGCGGAGGCAGTGCGATCCTCCGTT
AAGATATTCTTACGTGTGACGTAGCTATGTATTTTGCAGAGCTGGCGAACGCGTTGAACACTTCACAGATGGTAGGGATTCGGGTAAAG
GGCGTATAATTGGGGACTAACATAGGCGTAGACTACGATGGCGCCAACTCAATCGCAGCTCGAGCGCCCTGAATAACGTACTCATCTCA
ACTCATTCTCGGCAATCTACCGAGCGACTCGATTATCAACGGCTGTCTAGCAGTTCTAATCTTTTGCCAGCATCGTAATAGCCTCCAAG
AGATTGATGATAGCTATCGGCACAGAACTGAGACGGCGCCGATGGATAGCGGACTTTCGGTCAACCACAATTCCCCACGGGACAGGTCC
TGCGGTGCGCATCACTCTGAATGTACAAGCAACCCAAGTGGGCCGAGCCTGGACTCAGCTGGTTCCTGCGTGAGCTCGAGACTCGGGAT
GACAGCTCTTTAAACATAGAGCGGGGGCGTCGAACGGTCGAGAAAGTCATAGTACCTCGGGTACCAACTTACTCAGGTTATTGCTTGAA
GCTGTACTATTTTAGGGGGGGAGCGCTGAAGGTCTCTTCTTCTCATGACTGAACTCGCGAGGGTCGTGAAGTCGGTTCCTTCAATGGTT
AAAAAACAAAGGCTTACTGTGCGCAGAGGAACGCCCATCTAGCGGCTGGCGTCTTGAATGCTCGGTCCCCTTTGTCATTCCGGATTAAT
CCATTTCCCTCATTCACGAGCTTGCGAAGTCTACATTGGTATATGAATGCGACCTAGAAGAGGGCGCTTAAAATTGGCAGTGGTTGATG
CTCTAAACTCCATTTGGTTTACTCGTGCATCACCGCGATAGGCTGACAAAGGTTTAACATTGAATAGCAAGGCACTTCCGGTCTCAATG
AACGGCCGGGAAAGGTACGCGCGCGGTATGGGAGGATCAAGGGGCCAATAGAGAGGCTCCTCTCTCACTCGCTAGGAGGCAAATGTAAA
ACAATGGTTACTGCATCGATACATAAAACATGTCCATCGGTTGCCCAAAGTGTTAAGTGTCTATCACCCCTAGGGCCGTTTCCCGCATA
TAAACGCCAGGTTGTATCCGCATTTGATGCTACCGTGGATGAGTCTGCGTCGAGCGCGCCGCACGAATGTTGCAATGTATTGCATGAGT
AGGGTTGACTAAGAGCCGTTAGATGCGTCGCTGTACTAATAGTTGTCGACAGACCGTCGAGATTAGAAAATGGTACCAGCATTTTCGGA
GGTTCTCTAACTAGTATGGATTGCGGTGTCTTCACTGTGCTGCGGCTACCCATCGCCTGAAATCCAGCTGGTGTCAAGCCATCCCCTCT
CCGGGACGCCGCATGTAGTGAAACATATACGTTGCACGGGTTCACCGCGGTCCGTTCTGAGTCGACCAAGGACACAATCGAGCTCCGAT
CCGTACCCTCGACAAACTTGTACCCGACCCCCGGAGCTTGCCAGCTCCTCGGGTATCATGGAGCCTGTGGTTCATCGCGTCCGATATCA
AACTTCGTCATGATAAAGTCCCCCCCTCGGGAGTACCAGAGAAGATGACTACTGAGTTGTGCGAT'
print("=== Sequence (Number of bases: {}) ===\n\n{}".format(len(dna_seq), dna_seq))
```

```
=== Sequence (Number of bases: 2012) ===

ATGGCAATAACCCCCCGTTTCTACTTCTAGAGGAGAAAAGTATTGACATGAGCGCTCCCGGCACAAGGGCCAAAGAAGTCTCCAATTTC
TTATTTCCGAATGACATGCGTCTCCTTGCGGGTAAATCACCGACCGCAATTCATAGAAGCCTGGGGGAACAGATAGGTCTAATTAGCTT
AAGAGAGTAAATCCTGGGATCATTCAGTAGTAACCATAAACTTACGCTGGGGCTTCTTCGGCGGATTTTTACAGTTACCAACCAGGAGA
TTTGAAGTAAATCAGTTGAGGATTTAGCCGCGCTATCCGGTAATCTCCAAATTAAAACATACCGTTCCATGAAGGCTAGAATTACTTAC
CGGCCTTTTCCATGCCTGCGCTATACCCCCCACTCTCCCGCTTATCCGTCCGAGCGGAGGCAGTGCGATCCTCCGTTAAGATATTCTT
ACGTGTGACGTAGCTATGTATTTTGCAGAGCTGGCGAACGCGTTGAACACTTCACAGATGGTAGGGATTCGGGTAAAGGGCGTATAATT
GGGGACTAACATAGGCGTAGACTACGATGGCGCCAACTCAATCGCAGCTCGAGCGCCCTGAATAACGTACTCATCTCAACTCATTCTCG
GCAATCTACCGAGCGACTCGATTATCAACGGCTGTCTAGCAGTTCTAATCTTTTGCCAGCATCGTAATAGCCTCCAAGAGATTGATGAT
AGCTATCGGCACAGAACTGAGACGGCGCCGATGGATAGCGGACTTTCGGTCAACCACAATTCCCCACGGGACAGGTCCTGCGGTGCGCA
TCACTCTGAATGTACAAGCAACCCAAGTGGGCCGAGCCTGGACTCAGCTGGTTCCTGCGTGAGCTCGAGACTCGGGATGACAGCTCTTT
AAACATAGAGCGGGGGCGTCGAACGGTCGAGAAAGTCATAGTACCTCGGGTACCAACTTACTCAGGTTATTGCTTGAAGCTGTACTATT
TTAGGGGGGGAGCGCTGAAGGTCTCTTCTTCTCATGACTGAACTCGCGAGGGTCGTGAAGTCGGTTCCTTCAATGGTTAAAAAACAAAG
GCTTACTGTGCGCAGAGGAACGCCCATCTAGCGGCTGGCGTCTTGAATGCTCGGTCCCCTTTGTCATTCCGGATTAATCCATTTCCCTC
ATTCACGAGCTTGCGAAGTCTACATTGGTATATGAATGCGACCTAGAAGAGGGCGCTTAAAATTGGCAGTGGTTGATGCTCTAAACTCC
ATTTGGTTTACTCGTGCATCACCGCGATAGGCTGACAAAGGTTTAACATTGAATAGCAAGGCACTTCCGGTCTCAATGAACGGCCGGGA
AAGGTACGCGCGCGGTATGGGAGGATCAAGGGGCCAATAGAGAGGCTCCTCTCTCACTCGCTAGGAGGCAAATGTAAAACAATGGTTAC
```

```
TGCATCGATACATAAAACATGTCCATCGGTTGCCCAAAGTGTTAAGTGTCTATCACCCCTAGGGCCGTTTCCCGCATATAAACGCCAGG
TTGTATCCGCATTTGATGCTACCGTGGATGAGTCTGCGTCGAGCGCGCCGCACGAATGTTGCAATGTATTGCATGAGTAGGGTTGACTA
AGAGCCGTTAGATGCGTCGCTGTACTAATAGTTGTCGACAGACCGTCGAGATTAGAAAATGGTACCAGCATTTTCGGAGGTTCTCTAAC
TAGTATGGATTGCGGTGTCTTCACTGTGCTGCGGCTACCCATCGCCTGAAATCCAGCTGGTGTCAAGCCATCCCCTCTCCGGGACGCCG
CATGTAGTGAAACATATACGTTGCACGGGTTCACCGCGGTCCGTTCTGAGTCGACCAAGGACACAATCGAGCTCCGATCCGTACCCTCG
ACAAACTTGTACCCGACCCCCGGAGCTTGCCAGCTCCTCGGGTATCATGGAGCCTGTGGTTCATCGCGTCCGATATCAAACTTCGTCAT
GATAAAGTCCCCCCCTCGGGAGTACCAGAGAAGATGACTACTGAGTTGTGCGAT
```

In this problem, you will help your friend analyze this sequence.

Exercise 0 (2 point). Complete the function, `count_bases(s)`. It takes as input a DNA sequence as a string, `s`. It should compute the number of occurrences of each base (i.e., `'A'`, `'C'`, `'G'`, and `'T'`) in `s`. It should then return these counts in a dictionary whose keys are the bases.

In [3]: Student's answer (Top)

```python
def count_bases(s):
    assert type(s) is str
    assert all([b in ['A', 'C', 'G', 'T'] for b in s])
    from collections import Counter
    return dict(Counter(s))
```

In [4]: Grade cell: `exercise_0_test` Score: 2.0 / 2.0 (Top)

```python
# Test cell: `exercise_0_test`

base_counts = count_bases(dna_seq)
print("Your result:", base_counts)

assert type(base_counts) is dict, "`base_counts` is of type `{}`, not `dict`.".format(t
ype(base_counts))
assert len(base_counts) <= 4, "There can be at most 4 bases."
for b, c in [('A', 501), ('C', 507), ('G', 508), ('T', 496)]:
    assert base_counts[b] == c, "Base '{}' has a count of {} when it should be {}.".for
mat(b, base_counts[b], c)

print("\n(Passed!)")
```

```
Your result: {'G': 508, 'A': 501, 'C': 507, 'T': 496}

(Passed!)
```

Enzyme "scissors." Your friend is interested in what will happen to the sequence if she uses certain "restriction enzymes" to cut it. The enzymes work by scanning the DNA sequence from left to right for a particular pattern. It then cuts the DNA wherever it finds a match.

A biologist's notation. Your friend does not know about regular expressions. Instead, she uses a special notation (https://en.wikipedia.org/wiki/Nucleic_acid_sequence) that other biologists use to describe base patterns. These are "extra letters" that have a special meaning.

For example, the special letter `N` denotes any base, i.e., any single occurrence of an `A`, `C`, `G`, or `T`. Therefore, when a biologist writes, `ANT`, that means `AAT`, `ACT`, `AGT`, or `ATT`.

Here is the complete set of special letters:

- `R`: Either `G` or `A`
- `Y`: Either `T` or `C`
- `K`: Either `G` or `T`
- `M`: Either `A` or `C`
- `S`: Either `G` or `C`
- `W`: Either `A` or `T`
- `B`: Anything but `A` (i.e., `G`, `T`, or `C`)
- `D`: Anything but `C`

- `H`: Anything but `G`
- `V`: Anything but `T`
- `N`: Anything, i.e., `A`, `C`, `G`, or `T`

Exercise 1 (4 points). Given a string in the biologist's notation, complete the function `bio_to_regex(pattern_bio)` so that it returns an equivalent pattern in Python's regular expression language.

If your function is correct, then the following code would also work:

```
assert re.search(bio_to_regex('ANT'), 'AGATTA') is not None
```

That's because `ANT` matches `ATT`, which is contained in `AGATTA`.

In [5]: Student's answer (Top)

```
def bio_to_regex(pattern_bio):
    # A handy conversion table, to map bio letters to regex subpatterns:
    translation_table = {'R': '[AG]', 'Y': '[TC]', 'K': '[GT]', 'M': '[AC]', 'S': '[GC]
', 'W': '[AT]',
                         'B': '[^A]', 'D': '[^C]', 'H': '[^G]', 'V': '[^T]', 'N': '.'}

    # Here is the most compact solution we came up with:
    translator = str.maketrans(translation_table)
    return pattern_bio.translate(translator)

    # However, the following loop-based code would also work:
    pattern_regex = ''
    for c in pattern_bio:
        if c in translate:
            pattern_regex += translation_table[c]
        else:
            pattern_regex += c
    return pattern_regex
```

In [6]: Grade cell: `exercise_1_test_0` Score: 2.0 / 2.0 (Top)

```
# Test cell: `exercise_1_test_0`

assert re.search(bio_to_regex('ANT'), 'AGATTA') is not None
assert set(re.findall(bio_to_regex('ANTAAT'), dna_seq)) == {'ATTAAT', 'ACTAAT'}
assert set(re.findall(bio_to_regex('GCRWTG'), dna_seq)) == {'GCGTTG', 'GCAATG'}
assert len(re.findall(bio_to_regex('CDCHA'), dna_seq)) == 18

print("\n(Passed first group of tests!)")
```

```
(Passed first group of tests!)
```

In [7]: Grade cell: `exercise_1_test_1` Score: 1.0 / 1.0 (Top)

```
# Test cell: `exercise_1_test_1`
if False:
    for c in {'Y', 'K', 'M', 'S', 'B', 'D', 'V'}:
        from random import sample
        x = ''.join([sample('ACGT', 1)[0] for _ in range(2)])
        y = ''.join([sample('ACGT', 1)[0] for _ in range(2)])
        pattern = '{}{}{}'.format(x, c, y)
        ans = set(re.findall(bio_to_regex(pattern), dna_seq))
        print("assert set(re.findall(bio_to_regex('{}'), dna_seq)) == {}".format(patter
n, ans))

assert set(re.findall(bio_to_regex('GABAT'), dna_seq)) == {'GACAT', 'GAGAT', 'GATAT'}
assert set(re.findall(bio_to_regex('GAVCA'), dna_seq)) == {'GACCA', 'GAACA'}
assert set(re.findall(bio_to_regex('TGYGG'), dna_seq)) == {'TGTGG', 'TGCGG'}
assert set(re.findall(bio_to_regex('GCKAA'), dna_seq)) == {'GCGAA'}
assert set(re.findall(bio_to_regex('ATSCA'), dna_seq)) == {'ATCCA'}
assert set(re.findall(bio_to_regex('GCMTT'), dna_seq)) == {'GCCTT', 'GCATT'}
assert set(re.findall(bio_to_regex('AGDCC'), dna_seq)) == {'AGTCC', 'AGACC'}
```

```
print("\n(Passed second set of tests!)")
```

```
(Passed second set of tests!)
```

Restriction sites. When an enzyme cuts the string, it does it in a certain location with respect to the target pattern. This information is encoded as a *restriction site*.

The way a biologist specifies the restriction site is with a special notation that embeds the cut in the pattern. For example, there is one enzyme that has a restriction site of the form, `ANT|AAT`, where the vertical bar, `'|'`, shows where the enzyme will split the sequence. So, if the input DNA sequence were

```
GCATAGTAATGTATTAATGGC
```

then there would two matches:

```
GCATAGTAATGTATTAATGGC
    ^^^^^^  ^^^^^^
    match!  match!
```

Furthermore, there would be two cuts, since this enzyme splits its pattern in the middle (between `ANT` and `AAT`):

```
GCATAGT|AATGTATT|AATGGC
    ^^^ ^^^  ^^^ ^^^
```

That would result in three fragments: `GCATAGT`, `AATGTATT`, and `AATGGC`.

Exercise 2 (5 points). Complete the function, `sim_cuts(site_pattern, s)`, below. The first argument, `site_pattern`, is the biologist's restriction site pattern, e.g., `ANT|AAT`, where there may be an embedded cut. The second argument, `s`, is the DNA sequence to cut. The function should return the fragments in the sequence order.

For the preceding example,

```
sim_cuts('ANT|AAT', 'GCATAGTAATGTATTAATGGC') == ['GCATAGT', 'AATGTATT', 'AATGGC']
```

> **Note.** There are *two* test cells, below. Both must pass for full credit, but if only one passes, you'll at least get some partial credit.

Solutions. There are several ways to attack this problem. Here are a four ideas. The first two are "expected" in the sense that you could have come up with them knowing only the content of Topic 5 (regular expressions). The latter two introduce some more advanced ideas.

Idea 0: Match and span. You can solve this problem using only the regular expression ideas covered in Topic 5. Here is a scheme.

1. Record where cuts occur within the `site_pattern`. Let's refer to these as "cut offsets."
2. Find occurrences of `site_pattern` *without* the cuts.
3. Iterate over all matches. For each one, cut the input using knowledge of the offsets.

The function `sim_cuts__0()` implements this scheme. We were expecting solutions that resembled it.

In [8]:
```
def sim_cuts__0(site_pattern, s):
    # Find location of cuts within `site_pattern`
    cut_sites = re.finditer('\|', site_pattern)
    cut_offsets = [m.span()[0] for m in cut_sites]

    # Generate a regex to match `site_pattern` _without_ cuts
    site_pattern_sans_cuts = ''.join(site_pattern.split('|'))
    site_regex = bio_to_regex(site_pattern_sans_cuts)

    # Main cutting loop
    cuts = [] # Holds final cuts
    offset_s = 0 # Start of current cut (beginning of `s`)
    for match in re.finditer(site_regex, s):
        match_start = match.span()[0] # Start of match
        for k in cut_offsets:
            offset_e = match_start + k # End of current cut
```

```
            cuts.append(s[offset_s:offset_e])
            offset_s = offset_e
    cuts.append(s[offset_s:]) # Last cut
    return cuts

sim_cuts__0('ANT|AAT', 'GCATAGTAATGTATTAATGGC')
```

Out[8]: ['GCATAGT', 'AATGTATT', 'AATGGC']

Idea 1: Variation on a theme. This next idea is just a minor twist on the first idea. Instead of cutting as it goes, it transforms the input string into a new one that has a special delimiter within it to mark the cut locations. This implementation uses `':'` as that delimiter. You can then just split this transformed input on it.

In [9]:

```
def sim_cuts__1(site_pattern, s):
    # Find location of cuts within `site_pattern`
    cut_sites = re.finditer('\|', site_pattern)
    cut_offsets = [m.span()[0] for m in cut_sites]

    # Generate a regex to match `site_pattern` _without_ cuts
    site_pattern_sans_cuts = ''.join(site_pattern.split('|'))
    site_regex = bio_to_regex(site_pattern_sans_cuts)

    # Transform `s` to contain a special cut delimiter, ':'
    s_new = ''
    offset_s = 0 # Start of current cut (beginning of `s`)
    for match in re.finditer(site_regex, s):
        match_start = match.span()[0] # Start of match
        for k in cut_offsets:
            offset_e = match_start + k # End of current cut
            s_new += s[offset_s:offset_e] + ':'
            offset_s = offset_e
    s_new += s[offset_s:] # Last cut
    return s_new.split(':')

sim_cuts__1('ANT|AAT', 'GCATAGTAATGTATTAATGGC')
```

Out[9]: ['GCATAGT', 'AATGTATT', 'AATGGC']

Idea 2: Global grouping. This idea is based on exploiting the concept of groups in regular expressions. For instance, consider converting

```
site_pattern == 'ANT|AAT'
```

into

```
site_regex == '^(.*)(A[ACGT]T)(AAT)(.*)$'
```

After all, **Exercise 1** was about transforming your friend's pattern into a regular expression; so, it's natural to ask how to transform this new type of pattern, too. Here, `site_regex` has four parts, where the cut would go in the middle.

The code below generalizes this idea to any number of cuts, and even works if the input has no cuts.

However, to make it work you need to know that how `(.*)` will match when there are multiple options. It's tricky because regular expressions match greedily. For instance, consider the following:

```
pattern = '(.*)(cat)(.*)'
    text = 'caat/cat/caat/cat/caat'
    matched_groups = re.match(pattern, text).groups()
    print(matched_groups) # ???
```

What does this print? After all, this `pattern` can match `text` in two ways. It turns out the first wildcard group, `(.*)`, will match as much as it can, so

```
matched_groups == ('caat/cat/caat/', 'cat', '/caat')
```

Thus, to find all `cat` instances you'd need to keep matching against `matched_groups[0]` until no matches remain. The solution below does just that.

In [10]:

```
def sim_cuts__2(site_pattern, s):
    site_parts = site_pattern.split('|')
    if len(site_parts) <= 1: # No cuts
```

```
        return s

    regex_parts = [bio_to_regex(p) for p in site_parts]
    site_regex = '^(.*)(' + ')('.join(regex_parts) + ')(.*)$'
    uncut, cuts = s, [] # not yet cut, already cut
    while uncut:
        match = re.match(site_regex, uncut)
        if match is not None:
            segments = list(match.groups())
            uncut = ''.join(segments[:2])
            new_cuts = segments[2:-2] + [''.join(segments[-2:])]
        else: # No more matches
            new_cuts = [uncut]
            uncut = '' # Terminates loop
        cuts = new_cuts + cuts
    return cuts

sim_cuts__2('ANT|AAT', 'GCATAGTAATGTATTAATGGC')
```

Out[10]: `['GCATAGT', 'AATGTATT', 'AATGGC']`

This method is probably less efficient than the previous two because of the greedily matched prefix. However, it does not involve any tricky indexing, and so is (arguably) a little simpler.

Idea 3: Substitution. This solution is more compact but uses an advanced feature of regular expressions, which are substitutions using `re.sub()` with backreferences (https://docs.python.org/3/library/re.html#re.sub).

It's easiest to see by example. Suppose you know a string has `cat` somewhere in it, and you wish swap everything in front of `cat` with everything behind. For example, `dogcatllama` would become `llamacatdog`. Essentially, you want to find `(.*)cat(.*)` and then swap the prefix wildcard with the suffix. Using `re.sub()`, you can do that as follows:

In [11]:
```
pattern = r'(.*)cat(.*)'
replacement_template = r'\2cat\1'
text = 'dogcatllama'
new_text = re.sub(pattern, replacement_template, text)
print(text, '=>', new_text)
```

```
dogcatllama => llamacatdog
```

The groups in `pattern` are assigned logical numbers, starting at 1; a `replacement_template` refers to these groups by `\1`, which would be the first matching group, and `\2`, which is the second.

So, applying that idea to this exercise, here is what we might try to do given the biology pattern, `ANT|AAT`.

1. Convert the biology pattern into an equivalent *grouped* regular expression *without* the cut, e.g., `ANT|AAT` becomes `(A.T)(AAT)`.
2. Generate a replacement template with a special delimiter that marks the cut. For instance, we could use `:` as a delimiter, since it won't appear in the biologist's string. Then the replacement template might be `\1:\2`.
3. Use `re.sub()` to replace instances of `(A.T)(AAT)` with `\1:\2`. For example, `CACTAATG` would become `CACT:AATG`.
4. Split the transformed string at each instance of the special delimiter.

The `sim_cuts__3()` function below implements this idea. It should work for any number of cuts.

In [12]:
```
def sim_cuts__3(site_pattern, s):
    # Generate a grouped regular expression, `regex_pattern`
    cut_parts = site_pattern.split('|')
    pure_bio_parts = ['(' + p + ')' for p in cut_parts]
    pure_bio_pattern = ''.join(pure_bio_parts)
    regex_pattern = bio_to_regex(pure_bio_pattern)

    # Generate a replacement pattern, `repl_pattern`
    repl_parts = [r'\{}'.format(i+1) for i in range(len(cut_parts))]
    repl_pattern = ':'.join(repl_parts)

    # Substitute to insert the delimiter, `:`
    s_with_cuts = re.sub(regex_pattern, repl_pattern, s)

    # Cut at all instances of `:`
    return s_with_cuts.split(':')
```

```
sim_cuts__3('ANT|AAT', 'GCATAGTAATGTATTAATGGC')
```

Out[12]: ['GCATAGT', 'AATGTATT', 'AATGGC']

One attractive property of this solution is that it does not involve any explicit loops, which can make it a little easier to analyze.

You probably came up with your own interesting solution! Pick your favorite and try it below.

In [13]: Student's answer (Top)

```python
def sim_cuts(site_pattern, s):
    return sim_cuts__3(site_pattern, s)
```

In [14]: Grade cell: `exercise_2_test_0` Score: 3.0 / 3.0 (Top)

```python
# Test cell: `exercise_2_test_0`

def check_sim_cuts(bio_pattern, s, true_cuts):
    print("\nChecking: '{}'...".format(bio_pattern))
    your_cuts = sim_cuts(bio_pattern, s)
    print("   Your result ({} fragments): {}".format(len(your_cuts), your_cuts))
    print("   True result ({}): {}".format(len(true_cuts), true_cuts))
    assert your_cuts == true_cuts, "Did not match!"
    print("   ==> Matched!")

# Check a simple case:
check_sim_cuts('ANT|AAT', 'GCATAGTAATGTATTAATGGC', ['GCATAGT', 'AATGTATT', 'AATGGC'])

print("\n(Passed first test of Exercise 2; two more to go in the next cell.)")
```

```
Checking: 'ANT|AAT'...
   Your result (3 fragments): ['GCATAGT', 'AATGTATT', 'AATGGC']
   True result (3): ['GCATAGT', 'AATGTATT', 'AATGGC']
   ==> Matched!

(Passed first test of Exercise 2; two more to go in the next cell.)
```

In [15]: Grade cell: `exercise_2_test_1` Score: 2.0 / 2.0 (Top)

```python
# Test cell: `exercise_2_test_1`

check_sim_cuts('ANT|AAT', dna_seq, ['ATGGCAATAACCCCCCGTTTCTACTTCTAGAGGAGAAAAGTATTGACATG
AGCGCTCCCGGCACAAGGGCCAAAGAAGTCTCCAATTTCTTATTTCCGAATGACATGCGTCTCCTTGCGGGTAAATCACCGACCGCA
ATTCATAGAAGCCTGGGGGAACAGATAGGTCTAATTAGCTTAAGAGAGTAAATCCTGGGATCATTCAGTAGTAACCATAAACTTACG
CTGGGGCTTCTTCGGCGGATTTTTACAGTTACCAACCAGGAGATTTGAAGTAAATCAGTTGAGGATTTAGCCGCGCTATCCGGTAAT
CTCCAAATTAAAACATACCGTTCCATGAAGGCTAGAATTACTTACCGGCCTTTTCCATGCCTGCGCTATACCCCCCCACTCTCCCGC
TTATCCGTCCGAGCGGAGGCAGTGCGATCCTCCGTTAAGATATTCTTACGTGTGACGTAGCTATGTATTTTGCAGAGCTGGCGAACG
CGTTGAACACTTCACAGATGGTAGGGATTCGGGTAAAGGGCGTATAATTGGGGACTAACATAGGCGTAGACTACGATGGCGCCAACT
CAATCGCAGCTCGAGCGCCCTGAATAACGTACTCATCTCAACTCATTCTCGGCAATCTACCGAGCGACTCGATTATCAACGGCTGTC
TAGCAGTTCTAATCTTTTGCCAGCATCGTAATAGCCTCCAAGAGATTGATGATAGCTATCGGCACAGAACTGAGACGGCGCCGATGG
ATAGCGGACTTTCGGTCAACCACAATTCCCCACGGGACAGGTCCTGCGGTGCGCATCACTCTGAATGTACAAGCAACCCAAGTGGGC
CGAGCCTGGACTCAGCTGGTTCCTGCGTGAGCTCGAGACTCGGGATGACAGCTCTTTAAACATAGAGCGGGGGCGTCGAACGGTCGA
GAAAGTCATAGTACCTCGGGTACCAACTTACTCAGGTTATTGCTTGAAGCTGTACTATTTTAGGGGGGGAGCGCTGAAGGTCTCTTC
TTCTCATGACTGAACTCGCGAGGGTCGTGAAGTCGGTTCCTTCAATGGTTAAAAAACAAAGGCTTACTGTGCGCAGAGGAACGCCCA
TCTAGCGGCTGGCGTCTTGAATGCTCGGTCCCCTTTGTCATTCCGGATT',
 'AATCCATTTCCCTCATTCACGAGCTTGCGAAGTCTACATTGGTATATGAATGCGACCTAGAAGAGGGCGCTTAAAATTGGCAGTG
GTTGATGCTCTAAACTCCATTTGGTTTACTCGTGCATCACCGCGATAGGCTGACAAAGGTTTAACATTGAATAGCAAGGCACTTCCG
GTCTCAATGAACGGCCGGGAAAGGTACGCGCGCGGTATGGGAGGATCAAGGGGCCAATAGAGAGGCTCCTCTCTCACTCGCTAGGAG
GCAAATGTAAAACAATGGTTACTGCATCGATACATAAAACATGTCCATCGGTTGCCCAAAGTGTTAAGTGTCTATCACCCCTAGGGC
CGTTTCCCGCATATAAACGCCAGGTTGTATCCGCATTTGATGCTACCGTGGATGAGTCTGCGTCGAGCGCGCCGCACGAATGTTGCA
ATGTATTGCATGAGTAGGGTTGACTAAGAGCCGTTAGATGCGTCGCTGTACT',
 'AATAGTTGTCGACAGACCGTCGAGATTAGAAAATGGTACCAGCATTTTCGGAGGTTCTCTAACTAGTATGGATTGCGGTGTCTTC
ACTGTGCTGCGGCTACCCATCGCCTGAAATCCAGCTGGTGTCAAGCCATCCCCTCTCCGGGACGCCGCATGTAGTGAAACATATACG
TTGCACGGGTTCACCGCGGTCCGTTCTGAGTCGACCAAGGACACAATCGAGCTCCGATCCGTACCCTCGACAAACTTGTACCCGACC
CCCGGAGCTTGCCAGCTCCTCGGGTATCATGGAGCCTGTGGTTCATCGCGTCCGATATCAAACTTCGTCATGATAAAGTCCCCCCCT
CGGGAGTACCAGAGAAGATGACTACTGAGTTGTGCGAT'])
check_sim_cuts('GCRW|TG', dna_seq, ['ATGGCAATAACCCCCCGTTTCTACTTCTAGAGGAGAAAAGTATTGACATG
AGCGCTCCCGGCACAAGGGCCAAAGAAGTCTCCAATTTCTTATTTCCGAATGACATGCGTCTCCTTGCGGGTAAATCACCGACCGCA
ATTCATAGAAGCCTGGGGGAACAGATAGGTCTAATTAGCTTAAGAGAGTAAATCCTGGGATCATTCAGTAGTAACCATAAACTTACG
```

```
CTGGGGCTTCTTCGGCGGATTTTTACAGTTACCAACCAGGAGATTTGAAGTAAATCAGTTGAGGATTTAGCCGCGCTATCCGGTAAT
CTCCAAATTAAAACATACCGTTCCATGAAGGCTAGAATTACTTACCGGCCTTTTCCATGCCTGCGCTATACCCCCCACTCTCCCGC
TTATCCGTCCGAGCGGAGGCAGTGCGATCCTCCGTTAAGATATTCTTACGTGTGACGTAGCTATGTATTTTGCAGAGCTGGCGAACG
CGT',
 'TGAACACTTCACAGATGGTAGGGATTCGGGTAAAGGGCGTATAATTGGGGACTAACATAGGCGTAGACTACGATGGCGCCAACTC
AATCGCAGCTCGAGCGCCCTGAATAACGTACTCATCTCAACTCATTCTCGGCAATCTACCGAGCGACTCGATTATCAACGGCTGTCT
AGCAGTTCTAATCTTTTGCCAGCATCGTAATAGCCTCCAAGAGATTGATGATAGCTATCGGCACAGAACTGAGACGGCGCCGATGGA
TAGCGGACTTTCGGTCAACCACAATTCCCCACGGGACAGGTCCTGCGGTGCGCATCACTCTGAATGTACAAGCAACCCAAGTGGGCC
GAGCCTGGACTCAGCTGGTTCCTGCGTGAGCTCGAGACTCGGGATGACAGCTCTTTAAACATAGAGCGGGGGCGTCGAACGGTCGAG
AAAGTCATAGTACCTCGGGTACCAACTTACTCAGGTTATTGCTTGAAGCTGTACTATTTTAGGGGGGGAGCGCTGAAGGTCTCTTCT
TCTCATGACTGAACTCGCGAGGGTCGTGAAGTCGGTTCCTTCAATGGTTAAAAAACAAAGGCTTACTGTGCGCAGAGGAACGCCCAT
CTAGCGGCTGGCGTCTTGAATGCTCGGTCCCCTTTGTCATTCCGGATTAATCCATTTCCCTCATTCACGAGCTTGCGAAGTCTACAT
TGGTATATGAATGCGACCTAGAAGAGGGCGCTTAAAATTGGCAGTGGTTGATGCTCTAAACTCCATTTGGTTTACTCGTGCATCACC
GCGATAGGCTGACAAAGGTTTAACATTGAATAGCAAGGCACTTCCGGTCTCAATGAACGGCCGGGAAAGGTACGCGCGCGGTATGGG
AGGATCAAGGGGCCAATAGAGAGGCTCCTCTCTCACTCGCTAGGAGGCAAATGTAAAACAATGGTTACTGCATCGATACATAAAACA
TGTCCATCGGTTGCCCAAAGTGTTAAGTGTCTATCACCCCTAGGGCCGTTTCCCGCATATAAACGCCAGGTTGTATCCGCATTTGAT
GCTACCGTGGATGAGTCTGCGTCGAGCGCGCCGCACGAATGTTGCAA',
 'TGTATTGCATGAGTAGGGTTGACTAAGAGCCGTTAGATGCGTCGCTGTACTAATAGTTGTCGACAGACCGTCGAGATTAGAAAAT
GGTACCAGCATTTTCGGAGGTTCTCTAACTAGTATGGATTGCGGTGTCTTCACTGTGCTGCGGCTACCCATCGCCTGAAATCCAGCT
GGTGTCAAGCCATCCCCTCTCCGGGACGCCGCATGTAGTGAAACATATACGTTGCACGGGTTCACCGCGGTCCGTTCTGAGTCGACC
AAGGACACAATCGAGCTCCGATCCGTACCCTCGACAAACTTGTACCCGACCCCCGGAGCTTGCCAGCTCCTCGGGTATCATGGAGCC
TGTGGTTCATCGCGTCCGATATCAAACTTCGTCATGATAAAGTCCCCCCCTCGGGAGTACCAGAGAAGATGACTACTGAGTTGTGCG
AT'])

print("\n(Passed second tests of Exercise 2!)")
```

```
Checking: 'ANT|AAT'...
   Your result (3 fragments): ['ATGGCAATAACCCCCCGTTTCTACTTCTAGAGGAGAAAAGTATTGACATGAGCGCTC
CCGGCACAAGGGCCAAAGAAGTCTCCAATTTCTTATTTCCGAATGACATGCGTCTCCTTGCGGGTAAATCACCGACCGCAATTCATAGA
AGCCTGGGGGAACAGATAGGTCTAATTAGCTTAAGAGAGTAAATCCTGGGATCATTCAGTAGTAACCATAAACTTACGCTGGGGCTTCT
TCGGCGGATTTTTACAGTTACCAACCAGGAGATTTGAAGTAAATCAGTTGAGGATTTAGCCGCGCTATCCGGTAATCTCCAAATTAAAA
CATACCGTTCCATGAAGGCTAGAATTACTTACCGGCCTTTTCCATGCCTGCGCTATACCCCCCACTCTCCCGCTTATCCGTCCGAGCG
GAGGCAGTGCGATCCTCCGTTAAGATATTCTTACGTGTGACGTAGCTATGTATTTTGCAGAGCTGGCGAACGCGTTGAACACTTCACAG
ATGGTAGGGATTCGGGTAAAGGGCGTATAATTGGGGACTAACATAGGCGTAGACTACGATGGCGCCAACTCAATCGCAGCTCGAGCGCC
CTGAATAACGTACTCATCTCAACTCATTCTCGGCAATCTACCGAGCGACTCGATTATCAACGGCTGTCTAGCAGTTCTAATCTTTTGCC
AGCATCGTAATAGCCTCCAAGAGATTGATGATAGCTATCGGCACAGAACTGAGACGGCGCCGATGGATAGCGGACTTTCGGTCAACCAC
AATTCCCCACGGGACAGGTCCTGCGGTGCGCATCACTCTGAATGTACAAGCAACCCAAGTGGGCCGAGCCTGGACTCAGCTGGTTCCTG
CGTGAGCTCGAGACTCGGGATGACAGCTCTTTAAACATAGAGCGGGGGCGTCGAACGGTCGAGAAAGTCATAGTACCTCGGGTACCAAC
TTACTCAGGTTATTGCTTGAAGCTGTACTATTTTAGGGGGGGAGCGCTGAAGGTCTCTTCTTCTCATGACTGAACTCGCGAGGGTCGTG
AAGTCGGTTCCTTCAATGGTTAAAAAACAAAGGCTTACTGTGCGCAGAGGAACGCCCATCTAGCGGCTGGCGTCTTGAATGCTCGGTCC
CCTTTGTCATTCCGGATT', 'AATCCATTTCCCTCATTCACGAGCTTGCGAAGTCTACATTGGTATATGAATGCGACCTAGAAGAGGG
CGCTTAAAATTGGCAGTGGTTGATGCTCTAAACTCCATTTGGTTTACTCGTGCATCACCGCGATAGGCTGACAAAGGTTTAACATTGAA
TAGCAAGGCACTTCCGGTCTCAATGAACGGCCGGGAAAGGTACGCGCGCGGTATGGGAGGATCAAGGGGCCAATAGAGAGGCTCCTCTC
TCACTCGCTAGGAGGCAAATGTAAAACAATGGTTACTGCATCGATACATAAAACATGTCCATCGGTTGCCCAAAGTGTTAAGTGTCTAT
CACCCCTAGGGCCGTTTCCCGCATATAAACGCCAGGTTGTATCCGCATTTGATGCTACCGTGGATGAGTCTGCGTCGAGCGCGCCGCAC
GAATGTTGCAATGTATTGCATGAGTAGGGTTGACTAAGAGCCGTTAGATGCGTCGCTGTACT', 'AATAGTTGTCGACAGACCGTCGA
GATTAGAAAATGGTACCAGCATTTTCGGAGGTTCTCTAACTAGTATGGATTGCGGTGTCTTCACTGTGCTGCGGCTACCCATCGCCTGA
AATCCAGCTGGTGTCAAGCCATCCCCTCTCCGGGACGCCGCATGTAGTGAAACATATACGTTGCACGGGTTCACCGCGGTCCGTTCTGA
GTCGACCAAGGACACAATCGAGCTCCGATCCGTACCCTCGACAAACTTGTACCCGACCCCCGGAGCTTGCCAGCTCCTCGGGTATCATG
GAGCCTGTGGTTCATCGCGTCCGATATCAAACTTCGTCATGATAAAGTCCCCCCCTCGGGAGTACCAGAGAAGATGACTACTGAGTTGT
GCGAT']
   True result (3): ['ATGGCAATAACCCCCCGTTTCTACTTCTAGAGGAGAAAAGTATTGACATGAGCGCTCCCGGCACAAG
GGCCAAAGAAGTCTCCAATTTCTTATTTCCGAATGACATGCGTCTCCTTGCGGGTAAATCACCGACCGCAATTCATAGAAGCCTGGGGG
AACAGATAGGTCTAATTAGCTTAAGAGAGTAAATCCTGGGATCATTCAGTAGTAACCATAAACTTACGCTGGGGCTTCTTCGGCGGATT
TTTACAGTTACCAACCAGGAGATTTGAAGTAAATCAGTTGAGGATTTAGCCGCGCTATCCGGTAATCTCCAAATTAAAACATACCGTTC
CATGAAGGCTAGAATTACTTACCGGCCTTTTCCATGCCTGCGCTATACCCCCCACTCTCCCGCTTATCCGTCCGAGCGGAGGCAGTGC
GATCCTCCGTTAAGATATTCTTACGTGTGACGTAGCTATGTATTTTGCAGAGCTGGCGAACGCGTTGAACACTTCACAGATGGTAGGGA
TTCGGGTAAAGGGCGTATAATTGGGGACTAACATAGGCGTAGACTACGATGGCGCCAACTCAATCGCAGCTCGAGCGCCCTGAATAACG
TACTCATCTCAACTCATTCTCGGCAATCTACCGAGCGACTCGATTATCAACGGCTGTCTAGCAGTTCTAATCTTTTGCCAGCATCGTAA
TAGCCTCCAAGAGATTGATGATAGCTATCGGCACAGAACTGAGACGGCGCCGATGGATAGCGGACTTTCGGTCAACCACAATTCCCCAC
GGGACAGGTCCTGCGGTGCGCATCACTCTGAATGTACAAGCAACCCAAGTGGGCCGAGCCTGGACTCAGCTGGTTCCTGCGTGAGCTCG
AGACTCGGGATGACAGCTCTTTAAACATAGAGCGGGGGCGTCGAACGGTCGAGAAAGTCATAGTACCTCGGGTACCAACTTACTCAGGT
TATTGCTTGAAGCTGTACTATTTTAGGGGGGGAGCGCTGAAGGTCTCTTCTTCTCATGACTGAACTCGCGAGGGTCGTGAAGTCGGTTC
CTTCAATGGTTAAAAAACAAAGGCTTACTGTGCGCAGAGGAACGCCCATCTAGCGGCTGGCGTCTTGAATGCTCGGTCCCCTTTGTCAT
TCCGGATT', 'AATCCATTTCCCTCATTCACGAGCTTGCGAAGTCTACATTGGTATATGAATGCGACCTAGAAGAGGGCGCTTAAAAT
TGGCAGTGGTTGATGCTCTAAACTCCATTTGGTTTACTCGTGCATCACCGCGATAGGCTGACAAAGGTTTAACATTGAATAGCAAGGCA
CTTCCGGTCTCAATGAACGGCCGGGAAAGGTACGCGCGCGGTATGGGAGGATCAAGGGGCCAATAGAGAGGCTCCTCTCTCACTCGCTA
GGAGGCAAATGTAAAACAATGGTTACTGCATCGATACATAAAACATGTCCATCGGTTGCCCAAAGTGTTAAGTGTCTATCACCCCTAGG
GCCGTTTCCCGCATATAAACGCCAGGTTGTATCCGCATTTGATGCTACCGTGGATGAGTCTGCGTCGAGCGCGCCGCACGAATGTTGCA
ATGTATTGCATGAGTAGGGTTGACTAAGAGCCGTTAGATGCGTCGCTGTACT', 'AATAGTTGTCGACAGACCGTCGAGATTAGAAAA
TGGTACCAGCATTTTCGGAGGTTCTCTAACTAGTATGGATTGCGGTGTCTTCACTGTGCTGCGGCTACCCATCGCCTGAAATCCAGCTG
GTGTCAAGCCATCCCCTCTCCGGGACGCCGCATGTAGTGAAACATATACGTTGCACGGGTTCACCGCGGTCCGTTCTGAGTCGACCAAG
GACACAATCGAGCTCCGATCCGTACCCTCGACAAACTTGTACCCGACCCCCGGAGCTTGCCAGCTCCTCGGGTATCATGGAGCCTGTGG
TTCATCGCGTCCGATATCAAACTTCGTCATGATAAAGTCCCCCCCTCGGGAGTACCAGAGAAGATGACTACTGAGTTGTGCGAT']
```

```
   ==> Matched!

Checking: 'GCRW|TG'...
   Your result (3 fragments): ['ATGGCAATAACCCCCCGTTTCTACTTCTAGAGGAGAAAAGTATTGACATGAGCGCTC
CCGGCACAAGGGCCAAAGAAGTCTCCAATTTCTTATTTCCGAATGACATGCGTCTCCTTGCGGGTAAATCACCGACCGCAATTCATAGA
AGCCTGGGGGAACAGATAGGTCTAATTAGCTTAAGAGAGTAAATCCTGGGATCATTCAGTAGTAACCATAAACTTACGCTGGGGCTTCT
TCGGCGGATTTTTACAGTTACCAACCAGGAGATTTGAAGTAAATCAGTTGAGGATTTAGCCGCGCTATCCGGTAATCTCCAAATTAAAA
CATACCGTTCCATGAAGGCTAGAATTACTTACCGGCCTTTTCCATGCCTGCGCTATACCCCCCCACTCTCCCGCTTATCCGTCCGAGCG
GAGGCAGTGCGATCCTCCGTTAAGATATTCTTACGTGTGACGTAGCTATGTATTTTGCAGAGCTGGCGAACGCGT', 'TGAACACTTC
ACAGATGGTAGGGATTCGGGTAAAGGGCGTATAATTGGGGACTAACATAGGCGTAGACTACGATGGCGCCAACTCAATCGCAGCTCGAG
CGCCCTGAATAACGTACTCATCTCAACTCATTCTCGGCAATCTACCGAGCGACTCGATTATCAACGGCTGTCTAGCAGTTCTAATCTTT
TGCCAGCATCGTAATAGCCTCCAAGAGATTGATGATAGCTATCGGCACAGAACTGAGACGGCGCCGATGGATAGCGGACTTTCGGTCAA
CCACAATTCCCCACGGGACAGGTCCTGCGGTGCGCATCACTCTGAATGTACAAGCAACCCAAGTGGGCCGAGCCTGGACTCAGCTGGTT
CCTGCGTGAGCTCGAGACTCGGGATGACAGCTCTTTAAACATAGAGCGGGGGCGTCGAACGGTCGAGAAAGTCATAGTACCTCGGGTAC
CAACTTACTCAGGTTATTGCTTGAAGCTGTACTATTTTAGGGGGGGAGCGCTGAAGGTCTCTTCTTCTCATGACTGAACTCGCGAGGGT
CGTGAAGTCGGTTCCTTCAATGGTTAAAAAACAAAGGCTTACTGTGCGCAGAGGAACGCCCATCTAGCGGCTGGCGTCTTGAATGCTCG
GTCCCCTTTGTCATTCCGGATTAATCCATTTCCCTCATTCACGAGCTTGCGAAGTCTACATTGGTATATGAATGCGACCTAGAAGAGGG
CGCTTAAAATTGGCAGTGGTTGATGCTCTAAACTCCATTTGGTTTACTCGTGCATCACCGCGATAGGCTGACAAAGGTTTAACATTGAA
TAGCAAGGCACTTCCGGTCTCAATGAACGGCCGGGAAAGGTACGCGCGCGGTATGGGAGGATCAAGGGGCCAATAGAGAGGCTCCTCTC
TCACTCGCTAGGAGGCAAATGTAAAACAATGGTTACTGCATCGATACATAAAACATGTCCATCGGTTGCCCAAAGTGTTAAGTGTCTAT
CACCCCTAGGGCCGTTTCCCGCATATAAACGCCAGGTTGTATCCGCATTTGATGCTACCGTGGATGAGTCTGCGTCGAGCGCGCCGCAC
GAATGTTGCAA', 'TGTATTGCATGAGTAGGGTTGACTAAGAGCCGTTAGATGCGTCGCTGTACTAATAGTTGTCGACAGACCGTCGA
GATTAGAAAATGGTACCAGCATTTTCGGAGGTTCTCTAACTAGTATGGATTGCGGTGTCTTCACTGTGCTGCGGCTACCCATCGCCTGA
AATCCAGCTGGTGTCAAGCCATCCCCTCTCCGGGACGCCGCATGTAGTGAAACATATACGTTGCACGGGTTCACCGCGGTCCGTTCTGA
GTCGACCAAGGACACAATCGAGCTCCGATCCGTACCCTCGACAAACTTGTACCCGACCCCCGGAGCTTGCCAGCTCCTCGGGTATCATG
GAGCCTGTGGTTCATCGCGTCCGATATCAAACTTCGTCATGATAAAGTCCCCCCCTCGGGAGTACCAGAGAAGATGACTACTGAGTTGT
GCGAT']
   True result (3): ['ATGGCAATAACCCCCCGTTTCTACTTCTAGAGGAGAAAAGTATTGACATGAGCGCTCCCGGCACAAG
GGCCAAAGAAGTCTCCAATTTCTTATTTCCGAATGACATGCGTCTCCTTGCGGGTAAATCACCGACCGCAATTCATAGAAGCCTGGGGG
AACAGATAGGTCTAATTAGCTTAAGAGAGTAAATCCTGGGATCATTCAGTAGTAACCATAAACTTACGCTGGGGCTTCTTCGGCGGATT
TTTACAGTTACCAACCAGGAGATTTGAAGTAAATCAGTTGAGGATTTAGCCGCGCTATCCGGTAATCTCCAAATTAAAACATACCGTTC
CATGAAGGCTAGAATTACTTACCGGCCTTTTCCATGCCTGCGCTATACCCCCCCACTCTCCCGCTTATCCGTCCGAGCGGAGGCAGTGC
GATCCTCCGTTAAGATATTCTTACGTGTGACGTAGCTATGTATTTTGCAGAGCTGGCGAACGCGT', 'TGAACACTTCACAGATGGTA
GGGATTCGGGTAAAGGGCGTATAATTGGGGACTAACATAGGCGTAGACTACGATGGCGCCAACTCAATCGCAGCTCGAGCGCCCTGAAT
AACGTACTCATCTCAACTCATTCTCGGCAATCTACCGAGCGACTCGATTATCAACGGCTGTCTAGCAGTTCTAATCTTTTGCCAGCATC
GTAATAGCCTCCAAGAGATTGATGATAGCTATCGGCACAGAACTGAGACGGCGCCGATGGATAGCGGACTTTCGGTCAACCACAATTCC
CCACGGGACAGGTCCTGCGGTGCGCATCACTCTGAATGTACAAGCAACCCAAGTGGGCCGAGCCTGGACTCAGCTGGTTCCTGCGTGAG
CTCGAGACTCGGGATGACAGCTCTTTAAACATAGAGCGGGGGCGTCGAACGGTCGAGAAAGTCATAGTACCTCGGGTACCAACTTACTC
AGGTTATTGCTTGAAGCTGTACTATTTTAGGGGGGGAGCGCTGAAGGTCTCTTCTTCTCATGACTGAACTCGCGAGGGTCGTGAAGTCG
GTTCCTTCAATGGTTAAAAAACAAAGGCTTACTGTGCGCAGAGGAACGCCCATCTAGCGGCTGGCGTCTTGAATGCTCGGTCCCCTTTG
TCATTCCGGATTAATCCATTTCCCTCATTCACGAGCTTGCGAAGTCTACATTGGTATATGAATGCGACCTAGAAGAGGGCGCTTAAAAT
TGGCAGTGGTTGATGCTCTAAACTCCATTTGGTTTACTCGTGCATCACCGCGATAGGCTGACAAAGGTTTAACATTGAATAGCAAGGCA
CTTCCGGTCTCAATGAACGGCCGGGAAAGGTACGCGCGCGGTATGGGAGGATCAAGGGGCCAATAGAGAGGCTCCTCTCTCACTCGCTA
GGAGGCAAATGTAAAACAATGGTTACTGCATCGATACATAAAACATGTCCATCGGTTGCCCAAAGTGTTAAGTGTCTATCACCCCTAGG
GCCGTTTCCCGCATATAAACGCCAGGTTGTATCCGCATTTGATGCTACCGTGGATGAGTCTGCGTCGAGCGCGCCGCACGAATGTTGCA
A', 'TGTATTGCATGAGTAGGGTTGACTAAGAGCCGTTAGATGCGTCGCTGTACTAATAGTTGTCGACAGACCGTCGAGATTAGAAAA
TGGTACCAGCATTTTCGGAGGTTCTCTAACTAGTATGGATTGCGGTGTCTTCACTGTGCTGCGGCTACCCATCGCCTGAAATCCAGCTG
GTGTCAAGCCATCCCCTCTCCGGGACGCCGCATGTAGTGAAACATATACGTTGCACGGGTTCACCGCGGTCCGTTCTGAGTCGACCAAG
GACACAATCGAGCTCCGATCCGTACCCTCGACAAACTTGTACCCGACCCCCGGAGCTTGCCAGCTCCTCGGGTATCATGGAGCCTGTGG
TTCATCGCGTCCGATATCAAACTTCGTCATGATAAAGTCCCCCCCTCGGGAGTACCAGAGAAGATGACTACTGAGTTGTGCGAT']
   ==> Matched!

(Passed second tests of Exercise 2!)
```

Fin! If you've reached this point and all tests above pass, your biologist friend thanks you and you are ready to submit your solution to this problem. Don't forget to save you work prior to submitting.

Portions of this problem were inspired by a fun book called Python for Biologists (https://pythonforbiologists.com/python-books).

problem3 (Score: 10.0 / 10.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 1.0 / 1.0)
3. Test cell (Score: 3.0 / 3.0)
4. Test cell (Score: 5.0 / 5.0)

Important note! Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

Problem 3: Scans

This problem is about a useful computational primitive known as a *scan*. It has four (4) exercises, which are worth a total of ten (10) points.

By way of set up, the module will revolve around a possibly new function for you called `accumulate()`, which is available in the itertools (https://docs.python.org/3/library/itertools.html) module. Run the following code cell to preload it.

In [1]:
```
from itertools import accumulate

SAVE_ACCUMULATE = accumulate # You may ignore this line, which some test cells will use
```

Background: Cumulative sums

Consider a sequence of n values, $[x_0, x_1, x_2, \ldots, x_{n-1}]$. Its *cumulative sum* (or *running sum*) is

$$[x_0, \underbrace{x_0 + x_1}, \underbrace{x_0 + x_1 + x_2}, \ldots, \underbrace{x_0 + x_1 + x_2 + \cdots + x_{n-1}}]$$

.

For example, the list

```
[5, 3, -4, 20, 2, 9, 0, -1]
```

has the following cumulative sum:

```
[5, 8, 4, 24, 26, 35, 35, 34]
```

The `accumulate()` function makes it easy to compute cumulative sums.

Exercise 0 (1 point). Run the following code cell. (Yes, that's it -- one free point with no coding required!)

In [2]: Grade cell: `exercise_0_test` Score: 1.0 / 1.0 (Top)

```
L = list(accumulate([5, 3, -4, 20, 2, 9, 0, -1]))
print(L)

print("\n(Passed!)")
```

```
[5, 8, 4, 24, 26, 35, 35, 34]

(Passed!)
```

> Note: The `accumulate()` function returns a certain object, which is why `list()` is used to convert its result into a list.

Scans

A cumulative sum is one example of a more general primitive also known as a *scan*.

Let $f(x, y)$ be any associative function of two values. Associative means that $f(f(x, y), z) = f(x, f(y, z))$. For example, addition is associative: suppose that $f(x, y) = x + y$; then it is true that $(x + y) + z = x + (y + z)$. The *scan* of a sequence with respect to f is

$$\begin{aligned}\mathrm{scan}([x_0, x_1, \ldots, x_{n-1}], f) = [\ & x_0, \\ & f(x_0, x_1) \\ & f(f(x_0, x_1), x_2) \\ & \vdots, \\ & f(\cdots(f(x_0, x_1), x_2), \cdots), x_{n-1}) \\ ]\end{aligned}$$

The `accumulate()` function lets you implement these kinds of scans easily, too. For example, convince yourself that computing the minimum of two values, $\min(x, y)$, is also associative. Then you can implement a *minimum scan*, or *min-scan* for short, as follows:

Exercise 1 (1 point). Run the following cell and make sure its output makes sense to you. (Yes, that's it -- no coding required.)

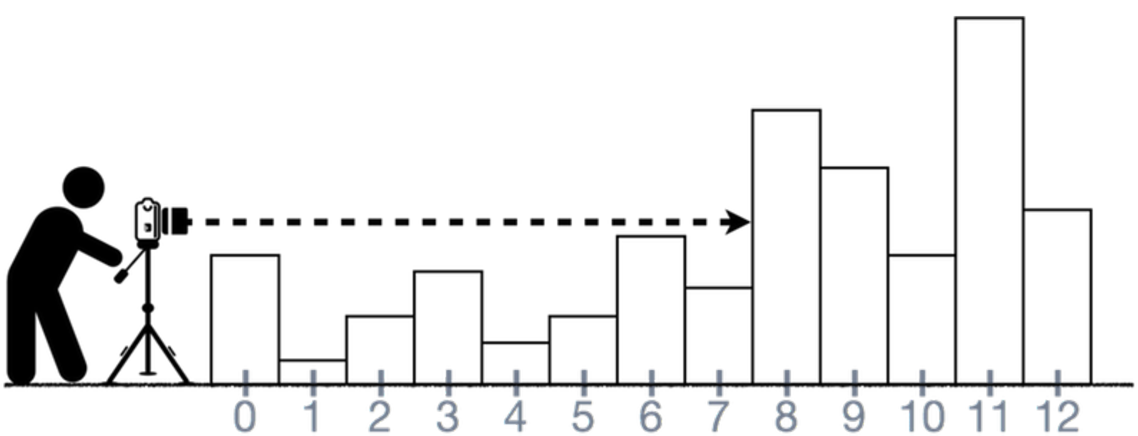
In [3]: Grade cell: `exercise_1_test` Score: 1.0 / 1.0 (Top)

```python
def min_scan(X):
    return list(accumulate(X, min))

print('min_scan({}) == {}'.format(L, min_scan(L)))

print("\n(Passed!)")
```

```
min_scan([5, 8, 4, 24, 26, 35, 35, 34]) == [5, 5, 4, 4, 4, 4, 4, 4]

(Passed!)
```

Exercise 2: Line-of-sight (3 points). Suppose a camera is fixed at a certain height in front of a bunch of obstacles of varying heights at different locations. You wish to determine all locations in which the camera's view is obstructed.

For example, in this cartoon, the camera's view is obstructed at all positions starting at 8:



Let `h` be the height of the camera and `X` be a list such that `X[i]` is the height of the obstacle at position `i`. Use `accumulate()` to complete the function, `get_obstruction(X, h)`, so that it returns a list of all positions at which the camera is obstructed. (For data corresponding to the above figure, this function would return `[8, 9, 10, 11, 12]`.

> An obstacle obstructs the camera if its height is *greater than* ***or*** *equal to* the height of the camera. For this exercise, your code must use `accumulate()` in some way.

In [4]: Student's answer (Top)

```
def get_obstruction(X, h):
    X_max = accumulate(X, max)
    return [i for i, x in enumerate(X_max) if x >= h]
```

In [5]: Grade cell: `exercise_2_test` Score: 3.0 / 3.0 (Top)

```
# Test cell: `exercise_2_test`

def check_get_obstruction(X, h):
    global accumulate
    print("Testing: h={}, X={}".format(h, X))
    try:
        del accumulate
        pos_test = get_obstruction(X, h)
    except NameError as n:
        if n.args[0] != "name 'accumulate' is not defined":
            raise n
    finally:
        accumulate = SAVE_ACCUMULATE

    pos = get_obstruction(X, h)
    print("\t==> Your code reports these positions as 'obstructed': {}".format(pos))
    for i in range(len(X)):
        msg = "Position i={} is incorrectly labeled."
        assert X[i] < h or i in pos, msg

# Test 0: Roughly the figure
X_test = [5, 1, 3, 3.5, 2, 3, 5.9, 4, 10, 8, 5, 12, 6]
h_test = 6
check_get_obstruction(X_test, h_test)

# Test 1: Random test cases
for _ in range(8):
    from random import randint
    h = randint(1, 5)
    n = randint(1, 10)
    X = [randint(0, 10) for _ in range(n)]
    check_get_obstruction(X, h)

print("\n(Passed!)")
```

```
Testing: h=6, X=[5, 1, 3, 3.5, 2, 3, 5.9, 4, 10, 8, 5, 12, 6]
        ==> Your code reports these positions as 'obstructed': [8, 9, 10, 11, 12]
Testing: h=3, X=[6, 7, 2, 5, 3, 6, 2, 9, 4]
        ==> Your code reports these positions as 'obstructed': [0, 1, 2, 3, 4, 5, 6, 7, 8
]
Testing: h=3, X=[10, 9, 4, 6]
        ==> Your code reports these positions as 'obstructed': [0, 1, 2, 3]
Testing: h=3, X=[8, 2, 5, 6]
        ==> Your code reports these positions as 'obstructed': [0, 1, 2, 3]
Testing: h=2, X=[9, 7, 10, 0, 6, 9, 6, 10, 1, 4]
        ==> Your code reports these positions as 'obstructed': [0, 1, 2, 3, 4, 5, 6, 7, 8
, 9]
Testing: h=4, X=[4, 4, 2, 0, 7, 4, 1]
        ==> Your code reports these positions as 'obstructed': [0, 1, 2, 3, 4, 5, 6]
Testing: h=1, X=[3, 6, 3, 2, 9, 3]
        ==> Your code reports these positions as 'obstructed': [0, 1, 2, 3, 4, 5]
Testing: h=4, X=[3, 2, 8, 0]
```

```
        ==> Your code reports these positions as 'obstructed': [2, 3]
Testing: h=4, X=[6, 1, 10, 10, 2, 9, 0]
        ==> Your code reports these positions as 'obstructed': [0, 1, 2, 3, 4, 5, 6]

(Passed!)
```

Application: When to buy a stock? Suppose you have the price of a stock on n consecutive days. For example, here is a list of stock prices observed on 14 consecutive days (assume these are numbered from 0 to 13, corresponding to the indices):

```
prices = [13, 11, 10, 8, 5, 8, 9, 6, 7, 7, 10, 7, 4, 3]
```

Suppose you buy on day `i` and sell on day `j`, where `j > i`. Then `prices[j] - prices[i]` measures your *profit* (or *loss*, if negative).

Exercise 3 (5 points). Implement a function, `max_profit(prices)`, to compute the best possible profit you could have made given a list of prices.

In the example above, that profit turns out to be 5. That's because you can buy on day 4, whose price is `prices[4] == 5`, and then sell on day 10, whose price is `prices[10] == 10`; it turns out there is no other combination will beat that profit.

There are two constraints on your solution:

1. You must use `accumulate()`. There is a (relatively) simple and fast solution that does so.
2. If only a loss is possible, your function should return 0.

In [6]: Student's answer (Top)

```python
def max_profit(prices):
    lowest_prices = accumulate(prices, min)
    gains = [p - l for p, l in zip(prices, lowest_prices)]
    max_gain = max([0] + gains)
    return max_gain
```

In [7]: Grade cell: `exercise_3_test` Score: 5.0 / 5.0 (Top)

```python
# Test cell: `exercise_3_test`

def check_profit(prices):
    global accumulate
    print("\nTesting: prices={}".format(prices))
    try:
        del accumulate
        profit_test = max_profit(prices)
    except NameError as n:
        if n.args[0] != "name 'accumulate' is not defined":
            raise n
    finally:
        accumulate = SAVE_ACCUMULATE

    profit = max_profit(prices)
    print("\t==> Your code's maximum profit: {}".format(profit))

    # Do an exhaustive search -- a correct, but highly inefficient, algorithm
    true_max = 0
    i_max, j_max = -1, -1
    for i in range(len(prices)):
        for j in range(i, len(prices)):
            gain_ij = prices[j] - prices[i]
            if gain_ij > true_max:
                i_max, j_max, true_max = i, j, gain_ij
    if i_max >= 0 and j_max >= 0:
        explain = "Buy on day {} at price {} and sell on {} at {}.".format(i_max, price
s[i_max],
                                                                            j_max, price
s[j_max])
    else:
        explain = "No buying options!"
    print("\t==> True max profit: {} ({})".format(true_max, explain))
    assert profit == true_max, "Your code's calculation does not match."
```

```
check_profit([13, 11, 10, 8, 5, 8, 9, 6, 7, 7, 10, 7, 4, 3])
check_profit([5, 4, 3, 2, 1])
check_profit([1, 2, 3, 4, 5])

for _ in range(8): # Random test cases
    from random import randint
    num_days = randint(1, 10)
    prices = [randint(1, 20) for _ in range(num_days)]
    check_profit(prices)

print("\n(Passed!)")
```

```
Testing: prices=[13, 11, 10, 8, 5, 8, 9, 6, 7, 7, 10, 7, 4, 3]
        ==> Your code's maximum profit: 5
        ==> True max profit: 5 (Buy on day 4 at price 5 and sell on 10 at 10.)

Testing: prices=[5, 4, 3, 2, 1]
        ==> Your code's maximum profit: 0
        ==> True max profit: 0 (No buying options!)

Testing: prices=[1, 2, 3, 4, 5]
        ==> Your code's maximum profit: 4
        ==> True max profit: 4 (Buy on day 0 at price 1 and sell on 4 at 5.)

Testing: prices=[1, 19, 1, 2, 18, 4, 9, 18, 3]
        ==> Your code's maximum profit: 18
        ==> True max profit: 18 (Buy on day 0 at price 1 and sell on 1 at 19.)

Testing: prices=[8, 8, 13, 1, 5, 10]
        ==> Your code's maximum profit: 9
        ==> True max profit: 9 (Buy on day 3 at price 1 and sell on 5 at 10.)

Testing: prices=[2, 16, 19, 16, 19]
        ==> Your code's maximum profit: 17
        ==> True max profit: 17 (Buy on day 0 at price 2 and sell on 2 at 19.)

Testing: prices=[15, 7, 8, 18, 6, 15, 2, 17, 15]
        ==> Your code's maximum profit: 15
        ==> True max profit: 15 (Buy on day 6 at price 2 and sell on 7 at 17.)

Testing: prices=[7, 1, 9, 9]
        ==> Your code's maximum profit: 8
        ==> True max profit: 8 (Buy on day 1 at price 1 and sell on 2 at 9.)

Testing: prices=[16, 15, 14]
        ==> Your code's maximum profit: 0
        ==> True max profit: 0 (No buying options!)

Testing: prices=[16, 2, 12, 11, 3, 6, 15, 10, 18, 1]
        ==> Your code's maximum profit: 16
        ==> True max profit: 16 (Buy on day 1 at price 2 and sell on 8 at 18.)

Testing: prices=[7, 4, 10, 14, 6, 20, 9, 9]
        ==> Your code's maximum profit: 16
        ==> True max profit: 16 (Buy on day 1 at price 4 and sell on 5 at 20.)

(Passed!)
```

Fin! If you've reached this point and all tests above pass, you are ready to submit your solution to this problem. Don't forget to save you work prior to submitting.

English

Download on the App Store

ANDROID APP ON Google play

POWERED BY OPEN edX