CS234 S17 - Assignment 1
Coverage: Python review, Sections 1-2; Modules 1 and 2.

This assignment consists of both a written component and a programming component. Please read the course website carefully to ensure that you submit each component correctly.

Several of the questions will make use of the ADT Pointset, which stores points as triples \((x, y, \text{feature})\), where \(x\) and \(y\) are integers in the range from 0 to \(\text{rows}-1\) and 0 to \(\text{cols}-1\), respectively, and \text{feature} is a string. The operations supported by ADT Pointset are the following, where in each case \(P\) is a pointset, \(\text{rows}\) and \(\text{cols}\) are positive integers, \(x\) is an integer in the range from 0 to \(\text{rows}-1\), \(y\) is an integer in the range from 0 to \(\text{cols}-1\), and \text{feature} is a string. This ADT allows only one point at a particular \((x, y)\) location, so the addition of a point may remove one previously stored. (ADT Pointset and the implementation are credited to Naomi Nishimura, with minor modifications.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Returns</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreatePointset(rows, cols)</td>
<td>a new empty pointset with limits rows and cols</td>
<td></td>
</tr>
<tr>
<td>IsEmptyPointset(P)</td>
<td>true if empty, else false</td>
<td></td>
</tr>
<tr>
<td>NumRows(P)</td>
<td>the number of rows in (P)</td>
<td></td>
</tr>
<tr>
<td>NumCols(P)</td>
<td>the number of columns in (P)</td>
<td></td>
</tr>
<tr>
<td>FeatureAtPoint(P, x, y)</td>
<td>feature at ((x, y)) or false if none</td>
<td></td>
</tr>
<tr>
<td>FindX(P, feature)</td>
<td>(x) value of a point with feature or false if none</td>
<td></td>
</tr>
<tr>
<td>FindY(P, feature)</td>
<td>(y) value of a point with feature or false if none</td>
<td></td>
</tr>
<tr>
<td>AddPoint(P, x, y, feature)</td>
<td>((x, y, \text{feature})) added to (P) replacing ((x, y, -)) if any</td>
<td></td>
</tr>
<tr>
<td>RemovePoint(P, x, y)</td>
<td>If a point exists at ((x, y)), it is removed from (P); if no point exists there, (P) is unchanged</td>
<td></td>
</tr>
</tbody>
</table>

Written component
For full marks, you are expected to provide a brief justification of any answer you provide.

W1. [3 marks] In a sentence or two, explain what the pseudocode function \(\text{vac}(x, y, m)\) produces.

function \(\text{vac}(x, y, m)\):
    for \(i\) from \(x\) to \(x + m\)
        for \(j\) from \(y\) to \(y + m\)
            if FeatureAtPoint(\(i, j\)) \(\neq\) False:
                return \(\text{False}\)
    return \(\text{True}\)
W2. [6 marks] In this question you will analyze the worst-case running time of the pseudocode function vac. For each subquestion, give the running time in asymptotic notation using a function on $n$, the number of items in $P$. Be sure to justify your answers.

(a) $m$ is in $O(1)$, and the cost of FeatureAtPoint is in $O(1)$.
(b) $m$ is in $O(n)$, and the cost of FeatureAtPoint is in $O(1)$.
(c) $m$ is in $O(n)$, and the cost of FeatureAtPoint is in $O(n)$.

W3. [5 marks] Using the pseudocode interface above, write a function Move that consumes a pointset, a feature, and $x$ and $y$ values. (You can assume that the pointset contains exactly one point with the matching feature.) The function will move the feature to the new location specified by the $x$ and $y$ values, and the feature will no longer be present at its original location.

W4. [6 marks] Answer each subquestion, explaining/justifying your answer.

(a) Will a given algorithm A with running time in $O(n)$ always provide better performance than a given algorithm B in $O(n^2)$?
(b) Algorithm C consists of a sequence of calls to four functions with running times in $\Theta(n^2), \Theta(\log n), \Theta(n^2),$ and $\Theta(n^2)$. (There are no branches/loops/etc. in C; it consists only of the four function calls, and constant time operations). Algorithm D is in $O(n^3)$. Which algorithm is preferable in general (in terms of performance), or is there not enough information to tell?

Programming component
Please read the course website carefully to ensure that you are using the correct version of Python and the correct style. Approximately 80% of the marks of each question will be allocated to correctness, with the remaining 20% to style, including aspects of the design recipe.

The programming questions for this assignment use the ADT Pointset, which has been implemented for you in the provided module pointset.py. Your code should work with the following code interface:

```python
## Pointset() produces a newly constructed empty Pointset with
## __init__: Int Int -> Pointset
def __init__(self, x_limit, y_limit):

## self.empty() produces True if self is empty.
## empty: Pointset -> Bool
def empty(self):

## self.num_rows() returns the x_limit (i.e., the number of rows in
## the pointset).
## empty: Pointset -> Int
def num_rows(self):
```
## self.num_cols() returns the y_limit (i.e., the number of columns in the pointset).
## empty: Pointset -> Int
def num_cols(self):

## self.feature_at_point(x, y) produces the feature at point (x, y),
##    if any, or otherwise False.
## feature_at_point: Pointset Int Int -> (anyof Str False)
## Requires: 0 <= x < x_limit 0 <= y < y_limit
def feature_at_point(self, x, y):

## self.find_x(feature) produces the x position of a point
##    with give feature, if any, or otherwise False.
## find_x: Pointset Str -> (anyof Int False)
def find_x(self, feature):

## self.find_y(feature) produces the y position of a point
##    with given feature, if any, or otherwise False.
## find_y: Pointset Str -> (anyof Int False)
def find_y(self, feature):

## add_point(x, y, feature) adds (x, y, feature) to self,
##    removing (x, y, -) if previously present.
## Effects: Mutates self.
## add_point: Pointset Int Int Str -> None
## Requires: 0 <= x < x_limit 0 <= y < y_limit
def add_point(self, x, y, feature):

## remove_point(x, y) removes any existing point at (x,y) from self,
##    with no effect if no such point exists.
## Effects: Mutates self.
## remove_point: Pointset Int Int -> None
## Requires: 0 <= x < x_limit 0 <= y < y_limit
def remove_point(self, x, y):

For example, if you wish to create a new pointset P, use

P = Pointset()

and if you wish to find the x position of a school in P, use

P.find_x("school")

Notice that adding a feature at (x, y) will replace any previous feature with values x and y.

P1. [10 marks] Write a program (stored in the file collision.py) that does the following:
• Reads in data from the file points1.txt and adds all data into a Pointset. (Please use 10 rows and 10 columns for your pointset.) Each line contains the x, y, and feature values for one entry.
• Reads in data from the file points2.txt, with the same format as points1.txt. (How you want to process/store points2.txt, in a Pointset or otherwise, is up to you.)
• For each point in points2.txt, determines if there is a point in points1.txt that occupies the same x, y coordinates (i.e., a ‘collision’). For each collision print a message on a single line, using the following structure:

 Collision: x 4 y 18

where 4 would be the x value of the collision, 18 would be the y value of the collision, and there is a single space separating each element after the colon. One line is printed per collision; if there are multiple collisions they should be printed in the same order as encountered in the file points2.txt.

• If there are no collisions in the file, then a single message should be printed:

 No collisions

NOTE: Your program will be tested automatically, which means that it is extremely important that what you print to the screen appears exactly as given here. Variations in spelling or punctuation will result in tests being failed, and hence loss of marks.

Two samples files have been provided. The contents of the sample points1.txt:
1 2 first
2 3 second
1 5 verylong
2 4 test
1 6 test2

The contents of the sample points2.txt:
3 3 a
1 2 b
6 5 c
2 3 d
1 1 e
1 2 f

The output from running the required program on these two files would be:
Collision: x 1 y 2
Collision: x 2 y 3
Collision: x 1 y 2

Note that your code will be tested with different input files; your testing should be more extensive than simply running this pair of files.

P2. [10 marks] Write a function row_ruler that consumes a Pointset and a positive integer row number i in the range 0 ≤ i < x_limit, and returns the feature in row i with the longest string length. If there are multiple features of the same length, return the first; if there are no features in row i, return None.

When using a pointset corresponding to that of points1.txt (and stored in the pointset set1), row_ruler(set1, 0) would return None, and row_ruler(set1, 1) would return ‘verylong’.