Ministry of Education, Singapore
Computing Teachers' Content Upgrading Course

Practical Assessment 2 - Trial Test

Time allowed: 3 hours

Instructions to candidates:

1. This is an **open-book** exam.
2. Answer all **three** questions.
3. You may complete your solutions in any IDE first before copy them into this Jupyter Notebook for submission.
4. Submit this Jupyter Notebook online before test ends. You may submit multiple times, but only last submission is accepted. https://driveuploader.com/upload/xTTmmKYr6C/ (https://driveuploader.com/upload/xTTmmKYr6C/)
5. Please note that the sample test program may not be enough to test your program. Your programs will be tested with other inputs and they should exhibit the required behaviours to get full credit.

Question 1

A class `Reverser` accepts multiple string inputs from user and is able to return them by characters in reverse order. For exmaple, following sample code shows how the Time Machine works.

```
        rv = Reverser()
        rv.push('Hello')
        rv.push('World')
        rv.pop()           # returns 1 character ['d']
        rv.pop(3)          # returns 3 characters ['l', 'r', 'o']
        rv.pop(10)         # returns remaining characters ['W', 'o', 'l', 'l', 'e',
'H']
```

A) Implement the `Reverser` class as either stack, queue, linked list or binary tree. The implementation must fulfill following requirements:

- Create an empty list `_data` in initializer to save characters.
- Write a method `push()` which takes in a string input, converts it into characters and saves them to `_data` .
- Write a method `pop()` which returns characters in reverse order.
  - Return max N characters if input parameter is N;
  - Return 1 character if input parameter is None.

In [1]:

```python
class Reverser:
    def __init__(self):
        self._data = []

    def push(self, item):
        self._data.extend(list(item))

    def pop(self, n = None):
        if n is None:
            return list(self._data.pop()) if self._data else None
        result = []
        while n > 0 and self._data:
            result.append(self._data.pop())
            n = n - 1
        return result

```

Test Case

Sample Output:

```
['d']
['l', 'r', 'o']
['W', 'o', 'l', 'l', 'e', 'H']
```

In [2]:

```python
rv = Reverser()
rv.push('Hello')
rv.push('World')
print(rv.pop())
print(rv.pop(3))
print(rv.pop(10))
```

```
['d']
['l', 'r', 'o']
['W', 'o', 'l', 'l', 'e', 'H']
```

B) Implement a class `Reverser2` which inherits from `Reverser` class and adds following functions.

- Write a method `peek()` which returns next character to be pop().
  - **Note:** It does not perform actual backtrack.
- Write a method `size()` which returns current number of characters saved in the list.
- Write a method `is_empty()` which returns `True` if there the list is empty, otherwise return `False` .

In [3]:

```
class Reverser2(Reverser):

    def peek(self):
        return self._data[-1] if self._data else None

    def size(self):
        return len(self._data)

    def is_empty(self):
        return len(self._data)==0

```

Test Case

Sample Output:

```
11
d
['d', 'l', 'r', 'o', 'W', ' ', 'o', 'l', 'l', 'e', 'H']
True
```

In [4]:

```
rv = Reverser2()
rv.push('Hello World')
print(rv.size())
print(rv.peek())
print(rv.pop(100))
print(rv.is_empty())
```

```
11
d
['d', 'l', 'r', 'o', 'W', ' ', 'o', 'l', 'l', 'e', 'H']
True
```

Question 2

The file `"olympics-medals.csv"` gives total number of Olympic medals won by participating countries from 1896 to 2008.

A) Write a function `load_file()` to read the file and returned records in a list.

- Each record (data point) in the list is a tuple of values, e.g. `('URS', 'Soviet Union', '838', 'Gold')` .
- Do NOT include header row in the returned list.

In [5]:

```python
import csv

def load_file(file):
    result = []
    with open(file) as f:
        reader = csv.reader(f)
        header = next(reader)
        for r in reader:
            result.append(r)
    return result
```

Test Case

Sample output

```
Records:  414
[('USA', 'United States', '2088', 'Gold'), ('URS', 'Soviet Union', '838', 'Gold'),
...]
```

In [6]:

```python
file_name = 'olympics-medals.csv'
table = load_file(file_name)
print("Records: ", len(table))
print(table[:4])
```

```
Records:  414
[['USA', 'United States', '2088', 'Gold'], ['URS', 'Soviet Union', '838', 'G
old'], ['GBR', 'United Kingdom', '498', 'Gold'], ['FRA', 'France', '378', 'G
old']]
```

B) The medals count (column 3) in all records must be a valid numerical value. Write a function `clean_medal_count()` to return a new table with only records with valid medal count.

- Return new table containing only records with valid medal count.

In [7]:

```python
def clean_medal_count(table):
    valid = [r for r in table if r[2].isnumeric()]
    return valid

```

Test Case

Sample output:

```
Valid records: 334
```

In [8]:

```python
table = load_file(file_name)
table2 = clean_medal_count(table)
print("Valid records: {}".format(len(table2)))
```

```
Valid records: 334
```

C) Write a function `list_medals()` to return medals records of a country.

- It returns all records which matches the country name.

In [9]:

```python
def list_medals(table, country):
    return [r for r in table if r[1] == country]
```

Test Case

Sample Output:

```
[('SIN', 'Singapore', '4', 'Silver')]
```

In [10]:

```python
table = load_file(file_name)
table2 = clean_medal_count(table)
records = list_medals(table2, 'Singapore')
print(records)
```

```
[['SIN', 'Singapore', '4', 'Silver']]
```

D) Write a function `top3_country()` to find the top 3 countries which have most medals in a category, e.g. Bronze.

In [11]:

```python
def top3_country(table, cat):
    result = [r for r in table if r[3] == cat]
    result = sorted(result, key=lambda x:int(x[2]), reverse=True)
    return result[:3]
```

Test Case

Sample output

```
Gold [('USA', 'United States', '2088', 'Gold'), ('URS', 'Soviet Union', '838', 'Go
ld'), ('GBR', 'United Kingdom', '498', 'Gold')]
```

In [12]:

```
table = load_file(file_name)
table2 = clean_medal_count(table)
category = 'Gold'
result = top3_country(table2, category)
print(category, result)
```

```
Gold [['USA', 'United States', '2088', 'Gold'], ['URS', 'Soviet Union', '83
8', 'Gold'], ['GBR', 'United Kingdom', '498', 'Gold']]
```

Question 3

A) A class `Point` represents a point on a Cartisian coordinate. Implement this class which fulfils following criterias:

- It has 2 instance variables, `_x` and `_y`, representing x coordinate value and y coordinate value respectively.
- Use 2 properties `x` and `y` to encapsulate its instance variables `_x` and `_y` respectively.
  - Both properties are readable and writable.
  - Both properties only accepts numeric value, i.e. it will not set the property if incoming value is not numeric.
- Implement a method `dist()` which returns its distance from origin using formula `math.sqrt(x*x+y*y)`.
- Implement its `__repr__()` method which returns a string in the format of `(_x,_y)`, e.g. "(3,4)".

Hint:

- Use `isnumeric()` method of string object to check if a string is numeric value.

In [13]:

```python
import math

class Point:
    def __init__(self, x, y):
        self._x = x
        self._y = y

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, val):
        if str(val).isnumeric():
            self._x = val

    @property
    def y(self):
        return self._y

    @y.setter
    def y(self, val):
        if str(val).isnumeric():
            self._y = val

    def dist(self):
        return math.sqrt(self._x * self._x + self._y * self._y)

    def __repr__(self):
        return '({},{})'.format(self._x, self._y)

```

Test Case

Sample output:

```
(3,4) 5.0
(3,5)
```

In [14]:

```python
i = Point(3, 4)
print(i, i.dist())
i.x = 'Orange'
i.y = 5
print(i)
```

```
(3,4) 5.0
(3,5)
```

B) Implement the class `Line` which is a collection of points (`Point` instances) on the same line.

- It has a instance variable `_points` of list type.
- Its initializer takes in 2 points, which defines the line, and save to the list.

- Define a static method `on_same_line()` which returns `True` if the 3 points are on the same line, otherwise reurns `False` . It uses formular `(y2-y1)/(x2-x1)=(y3-y1)/(x3-x1)` .
- Define a instance method `add_point()` which adds a point to list if that point is on the same line as other points.
- Implement its `__str__()` method which returns a string representation of its item list. E.g. `[(3,4), (6,8), (12,16)]`

In [15]:

```
class Line:

    def __init__(self, p1, p2):
        self._points = [p1, p2]

    @staticmethod
    def on_same_line(p1, p2, p3):
        return (p2.y-p1.y)/(p2.x-p1.x)==(p3.y-p1.y)/(p3.x-p1.x)

    def add_point(self, p):
        if Line.on_same_line(self._points[0], self._points[1], p):
            self._points.append(p)

    def __str__(self):
        return str(self._points)

```

Test Case

Sample output:

```
[(3,4), (6,8), (12,16)]
```

In [16]:

```
line = Line(Point(3,4), Point(6,8))
line.add_point(Point(12,16))
print(line)
```

```
[(3,4), (6,8), (12,16)]
```