IMPERIAL COLLEGE LONDON

EXAMINATIONS 2024-2025
(MOCK)

MSc in Artificial Intelligence
MRes in Artificial Intelligence and Machine Learning
for Internal Students of the Imperial College of Science, Technology and Medicine

# PAPER COMP70053

# PYTHON PROGRAMMING

Friday 29th November 2024, 16:15
Duration: 75 minutes

*Answer ALL questions*

Paper contains 4 questions

# Question 1 (5 marks)

In `q1/q1.py`, please complete the function `odd_even_swap(word)` to convert a `str` such that each character at an odd-numbered position $n$ is swapped with the next even numbered character at position $n + 1$. If the string is of an odd length, the last character will remain where it is.

The function should return a `str` where the characters in `word` have been swapped as described.

## Sample input and output

| `word` | **returns** |
|---|---|
| `"abcdefghi"` | `"badcfehgi"` |
| `"Python Programming"` | `"yPhtnoP orrgmaimgn"` |

# Question 2 (10 marks)

In `q2/q2.py`, please complete the function `closest_pair(coord_list)` which takes a set of coordinates in $K$-dimensional space and returns the pair of points that are closest to each other as measured using Euclidean distance.

`coord_list` is a `list` of `tuples`, where each `tuple` represents the coordinate of a point in $K$-dimensional space. Assume $K \geq 1$.

The function should return a `set` of two `tuples`, representing the pair of closest points.

The Euclidean distance $D$ of two points $a$ and $b$ in $K$-dimensional space is defined as $D(a, b) = \sqrt{\sum_k^K (x_{a,k} - x_{b,k})^2}$.

## Sample input and output

| `coord_list` | **returns** |
|---|---|
| `[`<br>`  (1, 2),`<br>`  (3, 5),`<br>`  (5, 4),`<br>`  (2, 1),`<br>`  (5, 2)`<br>`]` | `{`<br>`  (1, 2),`<br>`  (2, 1)`<br>`}` |
| `[`<br>`  (1, 1, 1),`<br>`  (2, 3, 1),`<br>`  (3, 3, 2),`<br>`  (1, 3, 3),`<br>`  (4, 1, 1)`<br>`]` | `{`<br>`  (2, 3, 1),`<br>`  (3, 3, 2)`<br>`}` |

# Question 3 (15 marks)

In `q3/q3.py`, please complete the function `top5_bigram_frequency(filename)` to return a dictionary containing the frequency count of the top 5 bigrams in `filename`.

An bigram is a sequence of two consecutive words. For example, in the sentence `"my term was very intense"`, the set of valid bigrams are `"my term"`, `"term was"`, `"was very"`, `"very intense"`.

Your function should:

- first convert the text contained in the file `filename` to be all **lowercase**.
- then return a `dict` of the top 5 most frequent bigrams and their frequency.

You can assume that each word in the file is separated by a space.

You may also assume that each line in `filename` is independent; there is no need to consider the last word from the previous line to be part of the bigram. For example, for the following two lines, `"buns one"` is not a bigram.

```
Line 1: hot cross buns hot cross buns
Line 2: one a penny two a penny hot cross buns
```

## Sample input and output

Using `"baby.txt"` provided in `q3/`, `top5_bigram_frequency("baby.txt")` should return:

```
{'baby baby': 36,
 'like baby': 14,
 'yeah yeah': 12,
 'baby oh': 9,
 "i 'm": 9}
```

Using `"frankenstein.txt"`,`top5_bigram_frequency("frankenstein.txt")` should return:

```
{'of the': 527,
 'of my': 272,
 'in the': 263,
 'i was': 228,
 'i had': 218}
```

# Question 4 (20 marks)

In `q4/q4.py`, please complete the function `get_winner(board)` to compute the winner of a game of Tic-Tac-Toe given the state of a `board`.

`board` is a two-dimensional `list` representing an $N \times N$ grid, where $N$ can be an arbitrary number between 3 and 50 (both inclusive). Each cell can be filled with either `"x"`, `"o"` or `" "` (a blank space). You can assume that `board` is always valid.

The function returns one of the following: `"x"`, `"o"`, `"draw"` or `None`.

The winner of the game is where either `"x"` or `"o"` has occupied $N$ consecutive grid cells in a single row, column or diagonal. If all cells are occupied without a winner, return `"draw"`. Otherwise, return `None` to indicate that the game has not yet finished.

## Sample input and output

```
board = [
['x','o',' ','o','x'],
['x','x','o',' ',' '],
['o','o','x',' ',' '],
[' ','o','o','x','o'],
[' ',' ',' ',' ','x']
]
```

**Output:** `"x"`

```
board = [
['o','x','o','x'],
['o','o','o','x'],
['x','x','x','o'],
['x','o','x','o']
]
```

**Output:** `"draw"`

```
board = [
[' ','x',' '],
['o','o','o'],
['x','x',' ']
]
```

**Output:** `"o"`

```
board = [
[' ',' ',' ','o',' ',' ','o'],
[' ',' ',' ',' ',' ',' ',' '],
[' ',' ',' ',' ',' ',' ',' '],
['x',' ','x','x',' ',' ',' '],
[' ',' ',' ',' ',' ',' ',' '],
[' ',' ',' ',' ',' ',' ',' '],
['x',' ',' ',' ',' ',' ','o']
]
```

**Output:** `None`