BY: Zhu_Ming

<CS1010S>

Tutorial 7

Lists and Dictionaries Processing

Lecture_Recap.tut07

Lecture Recap

1. Tuples
2. List
3. Dictionary
4. Linear Data Structure

1. Dictionary

- A <u>mutable</u> sequence of key-value pair

```
dct = {1 : “a”, 2 : “b”}
dct[3] = “c” # {1:”a”, 2:”b”, 3:”c”}
dct[1] = “z” # {1:”z”, 2:”b”, 3:”c”}
```

- Is a <u>reference</u> type
- Key must be immutable data type
- Value could be anything
- Curly brackets and colon
  e.g. {1 : 2}

Important

Dictionary only handle Int, Float, String, Boolean, None, Tuple as key!!

Recap!!

Primitive Type: (int, str, float, bool, none)
- fundamental data structure that <u>predefined</u>
- SAME identity!!

```
a = “same”
b = “same”
a == b # True
a is b # True
```

Reference Type:
- Look alike ⇐/⇒ Same Identity
- Same Identity ⇒ Look alike

```
dct1 = {1:2}
dct2 = {1:2}
dct1 == dct2 # True
dct1 is dct2 # False
```

Lecture_Recap.tut07

Dict Method : keys, values, items

```
dct = {1 : ”a”,
       2 : ”b”,
       3 : ”c”}

dct.keys()            # dict_keys([1, 2, 3])
dct.values()    # dict_values([“a”, “b”, “c”])
dct.items()           # dict_items([(1, “a”), (2, “b”), (3, “c”)])
```

Important!

Take note that **dct.items()** return key-value pair in form of **Tuple**

Dict as Iterator

```
dct = {1 : ”a”,
       2 : ”b”,
       3 : ”c”}
for key, value in dct.items():
    print(key, value)

>>> 1 “a”
>>> 2 “b”
>>> 3 “c”
```

PythonTips!

```
for idx in range(len(dct)):
    print(idx)

for idx, pair in enumerate(dct.items()):
    key, value = pair
    print(idx, key, value)
```

COMMON MISTAKE!!

NEVER MODIFY YOUR ITERATING DICTIONARY!!!

- ## Dict Method

```
dct[key]                  # getting the value
dct[key] = value          # assign/reassign value
```

```
dct = {1 : ”a”,
       2 : ”b”,
       3 : ”c”}

dct[1]
>>> “a”
dct[1] = “abc” # dct = {1:”abc”, 2:”b”, 3:”c”}
```

Important!

Searching in dictionary takes no time, O(1)!!

If you are interested to Order of growth of Dictionary, refer to this.

Lecture_Recap.tut07

Dict Method : update

```
dct1 = {1:2}
dct2 = {1:10, 3:4}
dct1.update(dct2)

dct1 >>> {1:10, 3:4}
```

Important

Take note that it is update of the dictionary (avoid thinking as merge of two dict, there is no concatenation / merge of two dict)

Dict Method : copy, clear, pop

```
dct.copy()
dct.clear()
dct.pop(key) # return value
dct1 = {1:[1,2]}
dct2 = {1:10, 3:4}
dct3 = dct1.copy()   # dct3 >>> {1:[1,2]}
dct3 is dct1         # False
dct3[1] is dct1[1]   # True
dct1.clear()         # dct1 >>> {}
dct2.pop(1)          # return 10, dct2 >>> {3:4}
```

PythonTips!

Take note that pop & update,UPDATE the dictionary (IN PLACE)! It return None!

copy is shallow copy!!

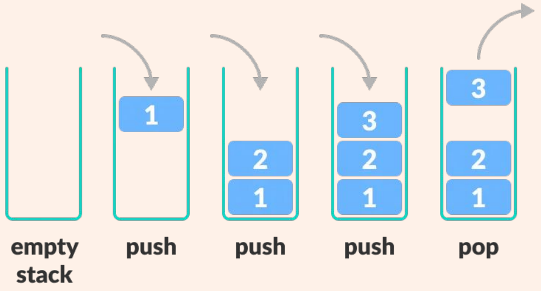
- **Dict Method : sort & reverse**

Important!

Order/sequences **DOES NOT** matter in Dictionary

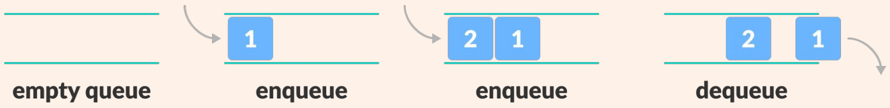
2. Linear Data Structure

- Stack
  - First In Last Out (FILO)
  - Can only access the **top-most** item
  - Operations: pop, push, peek



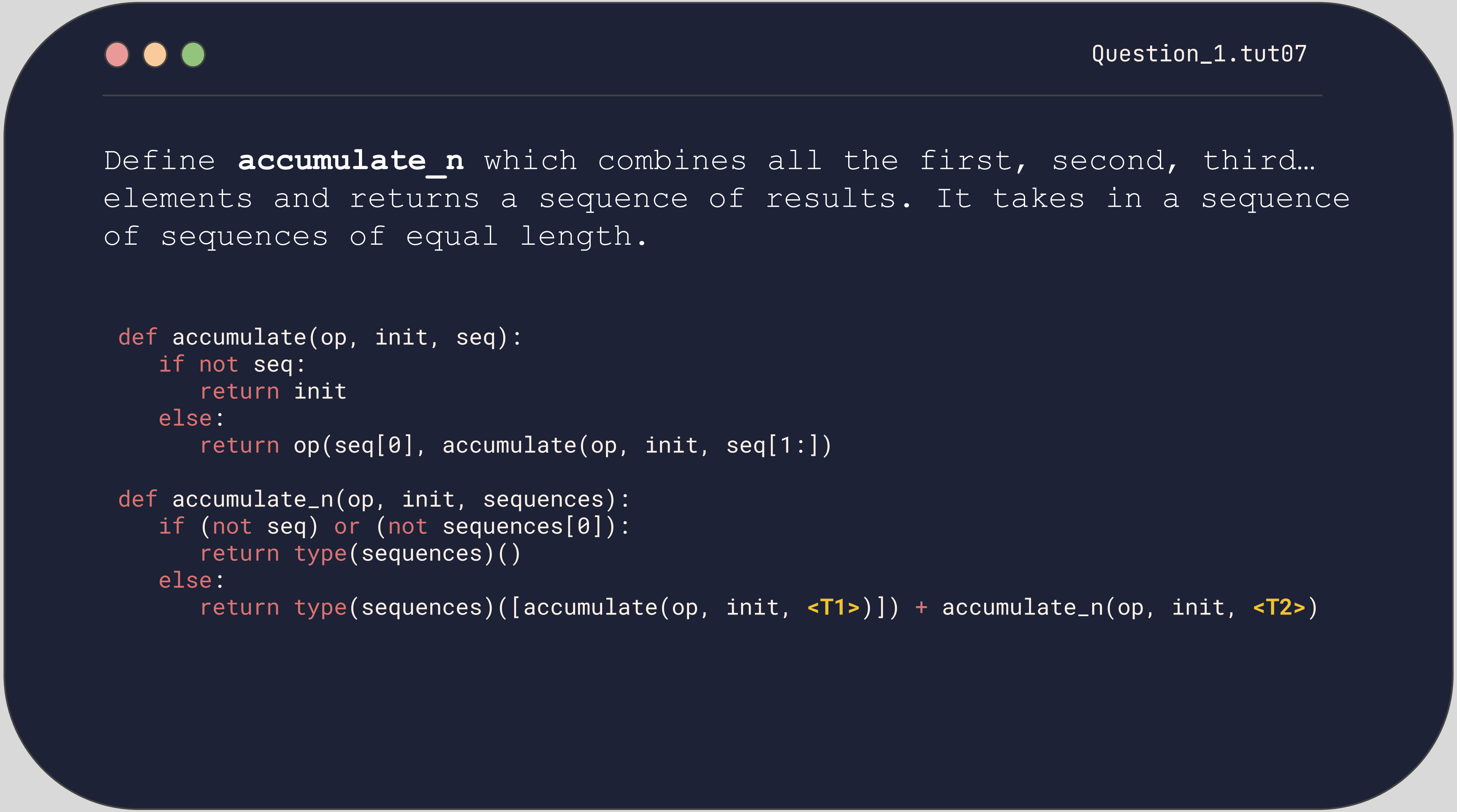
Animation

- Queue
  - First In First Out (FIFO)
  - Can only access the **front-most** item
  - Operations: enqueue, dequeue, peek



BY: Zhu_Ming

Any Questions?

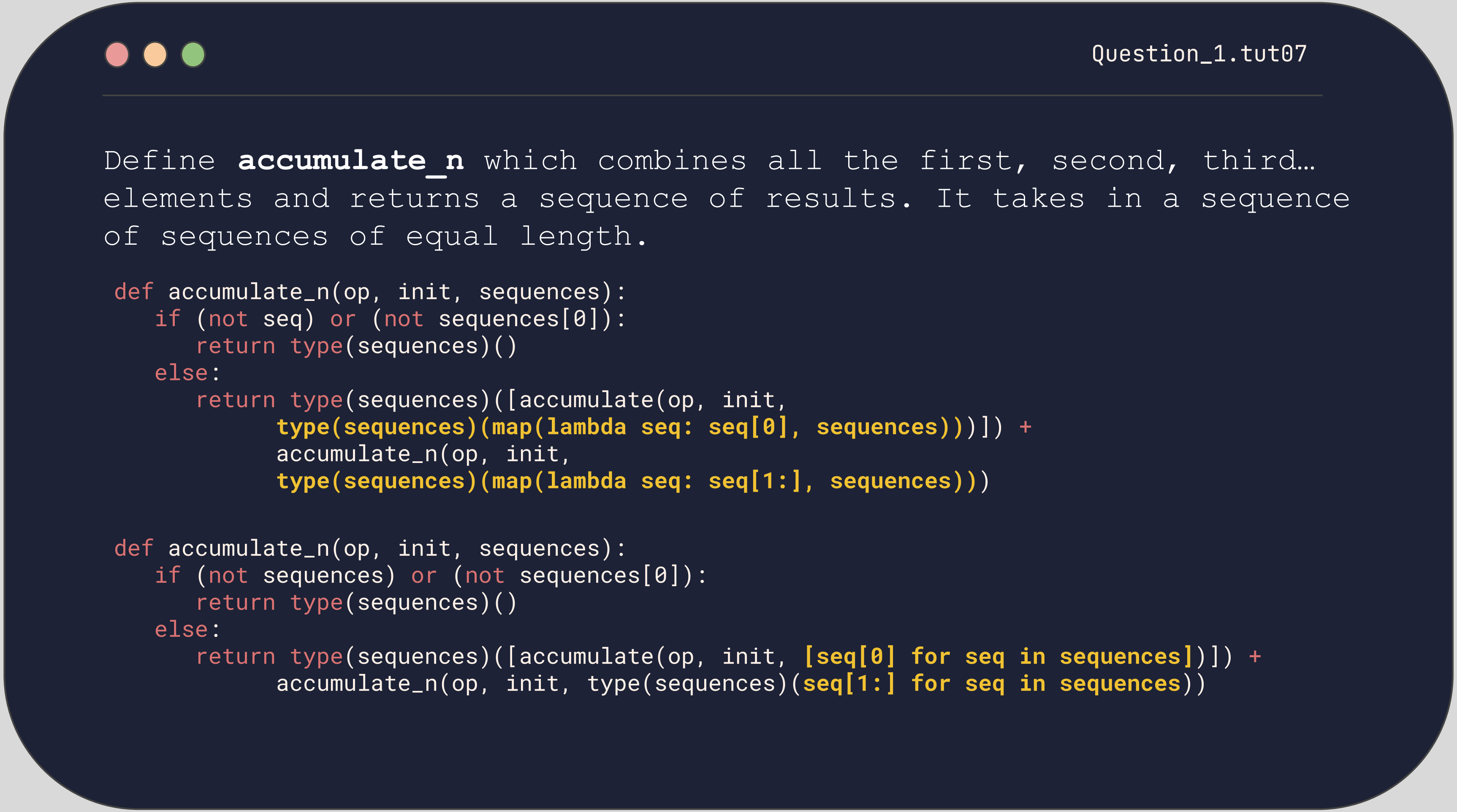


Question_1.tut07

Define **accumulate_n** which combines all the first, second, third… elements and returns a sequence of results. It takes in a sequence of sequences of equal length.

```
def accumulate(op, init, seq):
    if not seq:
        return init
    else:
        return op(seq[0], accumulate(op, init, seq[1:])

def accumulate_n(op, init, sequences):
    if (not seq) or (not sequences[0]):
        return type(sequences)()
    else:
        return type(sequences)([accumulate(op, init, <T1>)]) + accumulate_n(op, init, <T2>)
```



Define **accumulate_n** which combines all the first, second, third… elements and returns a sequence of results. It takes in a sequence of sequences of equal length.

```
def accumulate_n(op, init, sequences):
   if (not seq) or (not sequences[0]):
      return type(sequences)()
   else:
      return type(sequences)([accumulate(op, init,
            type(sequences)(map(lambda seq: seq[0], sequences)))]) +
            accumulate_n(op, init,
            type(sequences)(map(lambda seq: seq[1:], sequences)))
```

```
def accumulate_n(op, init, sequences):
   if (not sequences) or (not sequences[0]):
      return type(sequences)()
   else:
      return type(sequences)([accumulate(op, init, [seq[0] for seq in sequences])]) +
            accumulate_n(op, init, type(sequences)(seq[1:] for seq in sequences))
```

Question_2.tut07

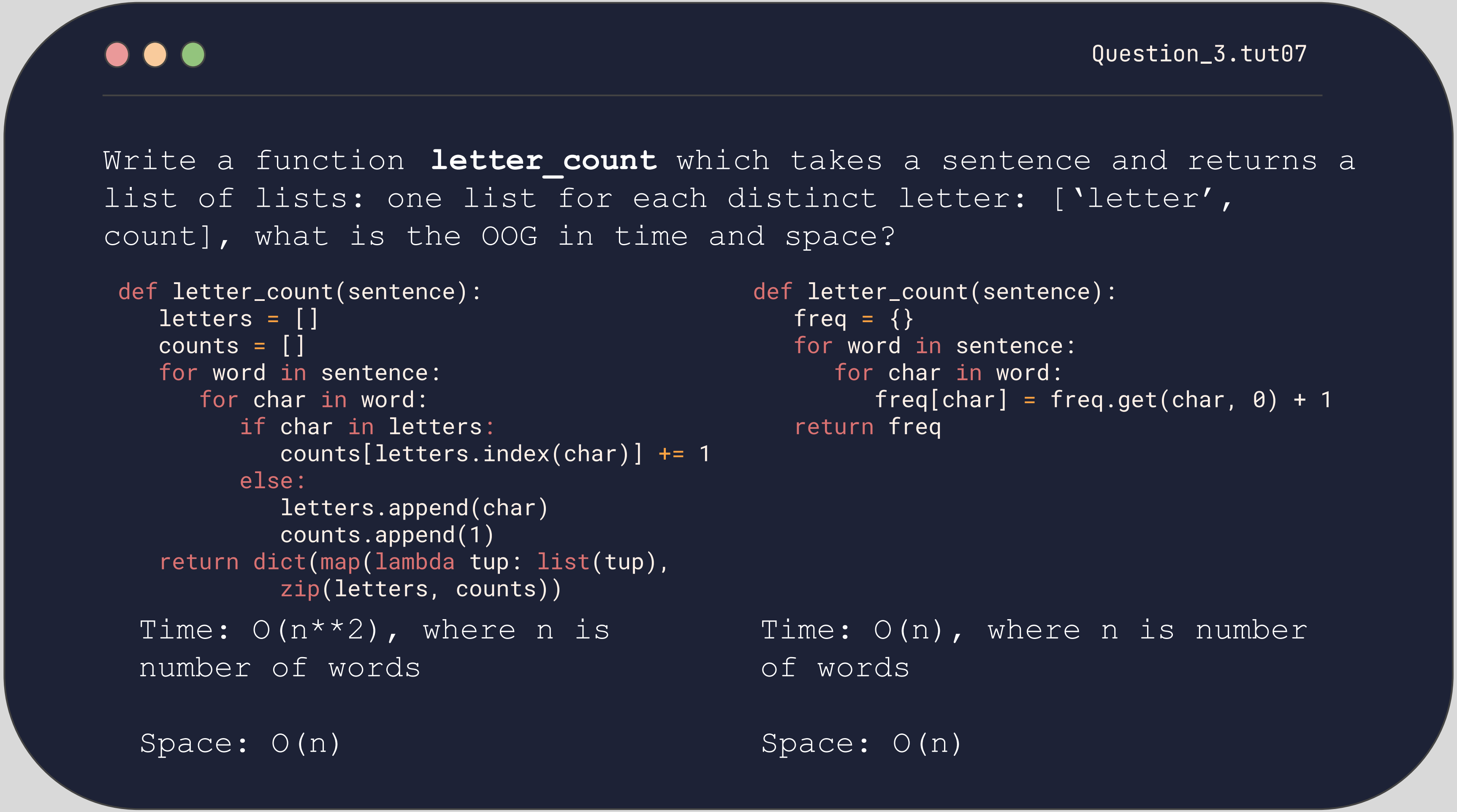
Write a function **count_sentence** which takes in a sentence representation and returns a list: **[number of words, number of letters]**, what is the OOG in time and space?

```python
def count_sentence(sentence):
   letter_count = 0
   for word in sentence:
      letter_count += len(word) # counting actual characters
   letter_count += len(sentence) - 1 # counting whitespaces
   return [len(sentence), letter_count]
```

```python
def count_sentence(sentence):
   sentence_len = len(sentence)
   letters_count = sum(len(ws) for word in sentence)
   return [sentence_len, letters_count + sentence_len - 1]
```

Time: O(n), where n is number of words

Space: O(1)



Question_3.tut07

Write a function **letter_count** which takes a sentence and returns a list of lists: one list for each distinct letter: [‘letter’, count], what is the OOG in time and space?

```
def letter_count(sentence):
    letters = []
    counts = []
    for word in sentence:
        for char in word:
            if char in letters:
                counts[letters.index(char)] += 1
            else:
                letters.append(char)
                counts.append(1)
    return dict(map(lambda tup: list(tup),
             zip(letters, counts))
```

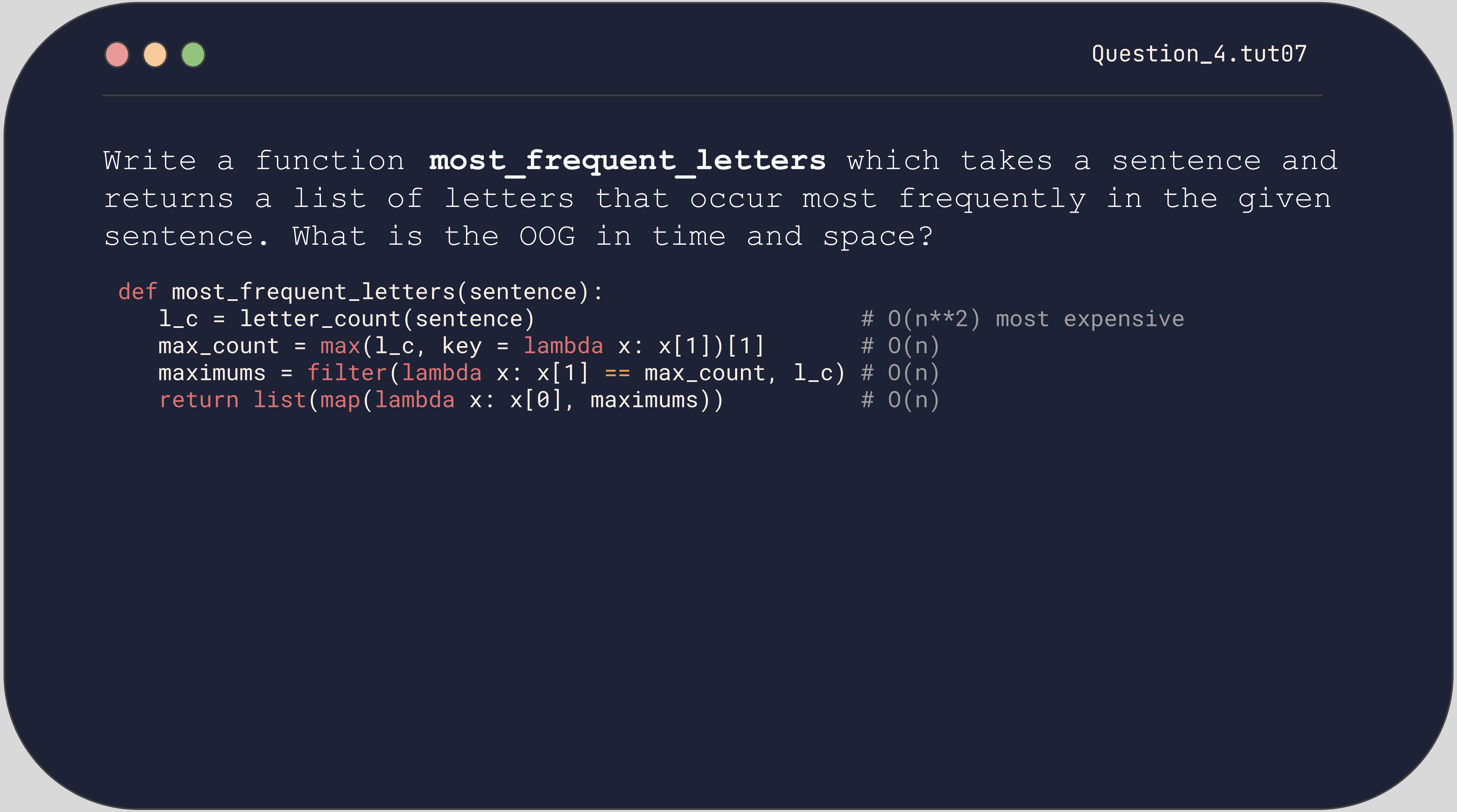
Time: O(n**2), where n is number of words

Space: O(n)

```
def letter_count(sentence):
    freq = {}
    for word in sentence:
        for char in word:
            freq[char] = freq.get(char, 0) + 1
    return freq
```

Time: O(n), where n is number of words

Space: O(n)



Question_4.tut07

Write a function **most_frequent_letters** which takes a sentence and returns a list of letters that occur most frequently in the given sentence. What is the OOG in time and space?

```
def most_frequent_letters(sentence):
   l_c = letter_count(sentence)                          # O(n**2) most expensive
   max_count = max(l_c, key = lambda x: x[1])[1]         # O(n)
   maximums = filter(lambda x: x[1] == max_count, l_c) # O(n)
   return list(map(lambda x: x[0], maximums))          # O(n)
```

Question_5.tut07

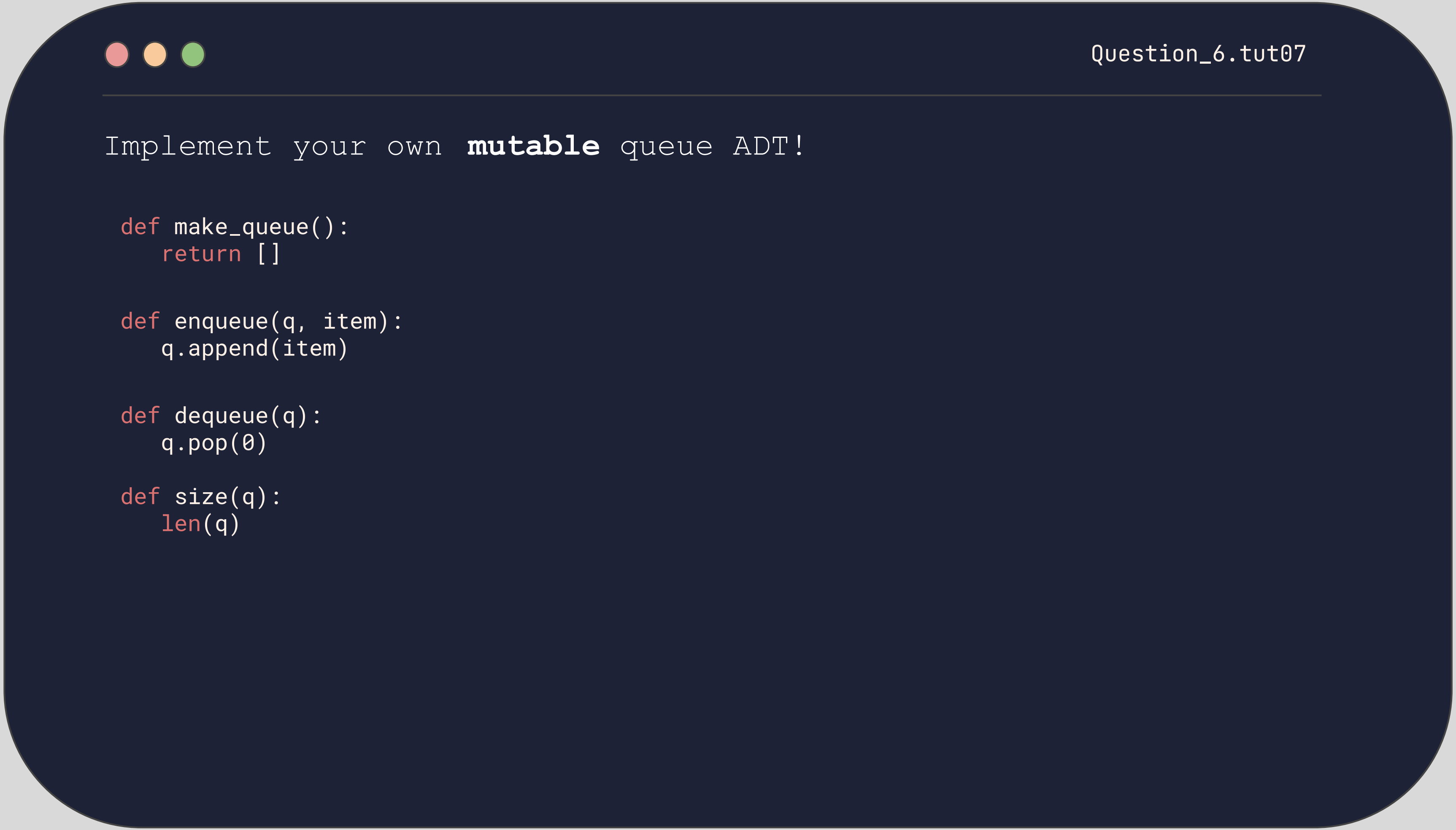
Re-implement the function **letter_count_dict** to return a dictionary of distinct letters and their count in the sentence. How has the OOG in time and space for **letter_count** and **most_frequent_letters** changed?

```
def letter_count(sentence):
    letters = []
    counts = []
    for word in sentence:
        for char in word:
            if char in letters:
                counts[letters.index(char)] += 1
            else:
                letters.append(char)
                counts.append(1)
    return list(map(lambda tup: list(tup),
                zip(letters, counts))
```

```
def letter_count_dict(sentence):
    freq = {}
    for word in sentence:
        for char in word:
            if char in freq:
                freq[char] += 1
            else:
                freq[char] = 1
    return dict(list(pair) for pair in freq.items())
```

Time: O(n**2), where n is number of words

Space: O(n)



Question_6.tut07

Implement your own **mutable** queue ADT!

```python
def make_queue():
    return []

def enqueue(q, item):
    q.append(item)

def dequeue(q):
    q.pop(0)

def size(q):
    len(q)
```

Question_6.tut07

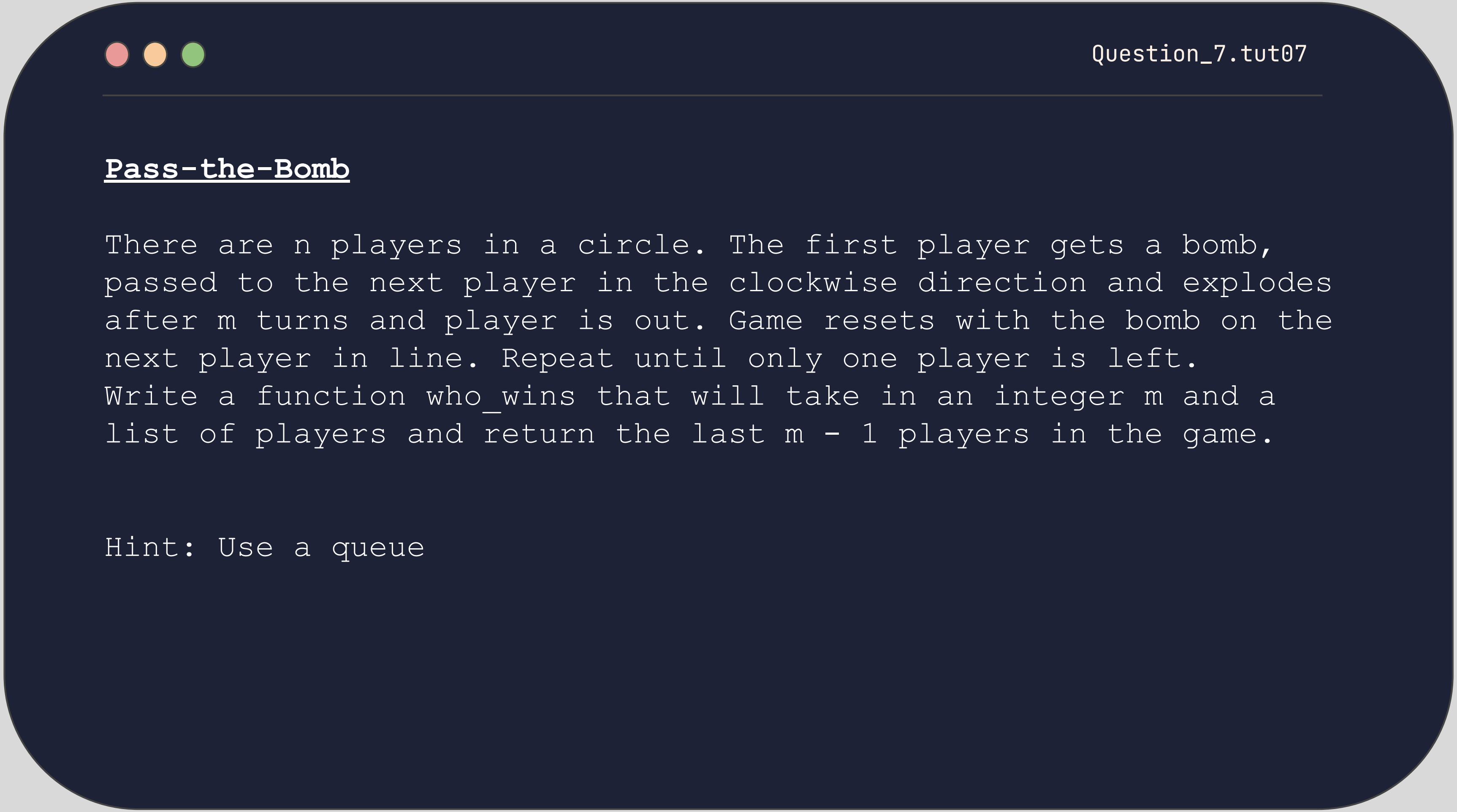
Implement your own **mutable** queue ADT!

```
def make_queue():
    return []

def enqueue(q, item):
    q.append(item)

def dequeue(q):
    return q.pop(0)

def size(q):
    return len(q)
```

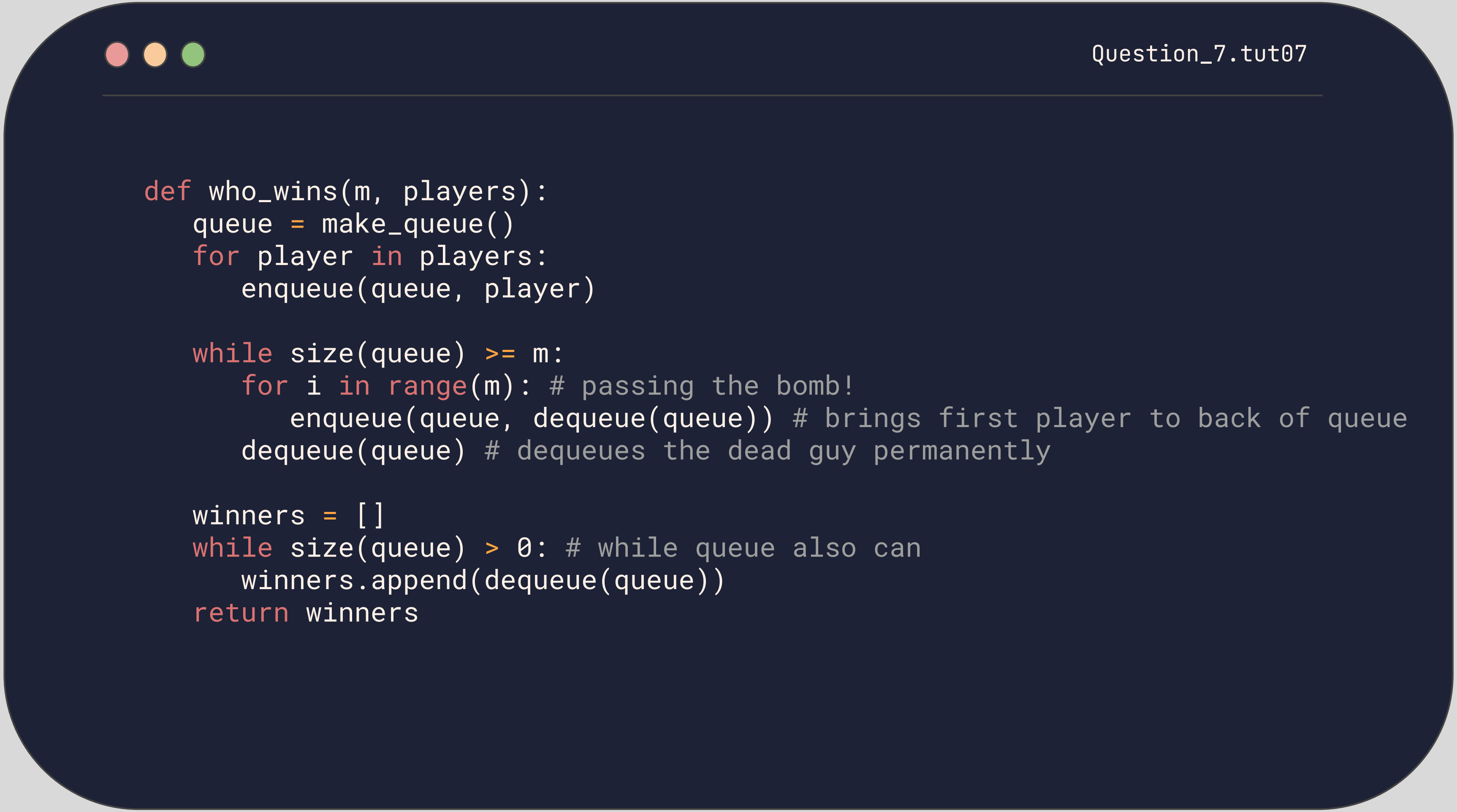


Question_7.tut07

Pass-the-Bomb

There are n players in a circle. The first player gets a bomb, passed to the next player in the clockwise direction and explodes after m turns and player is out. Game resets with the bomb on the next player in line. Repeat until only one player is left.
Write a function who_wins that will take in an integer m and a list of players and return the last m - 1 players in the game.

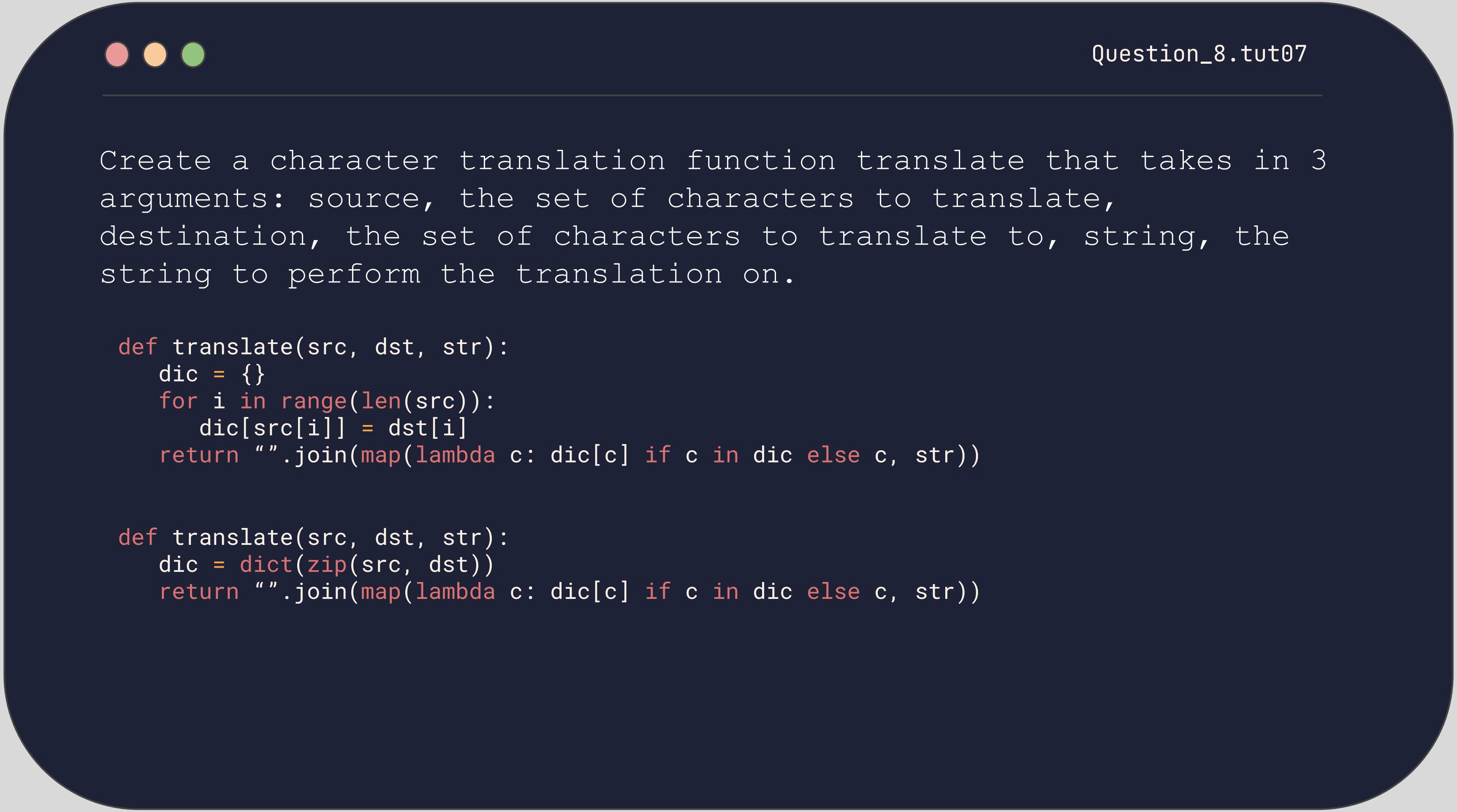
Hint: Use a queue



```
def who_wins(m, players):
   queue = make_queue()
   for player in players:
      enqueue(queue, player)

   while size(queue) >= m:
      for i in range(m): # passing the bomb!
         enqueue(queue, dequeue(queue)) # brings first player to back of queue
      dequeue(queue) # dequeues the dead guy permanently

   winners = []
   while size(queue) > 0: # while queue also can
      winners.append(dequeue(queue))
   return winners
```

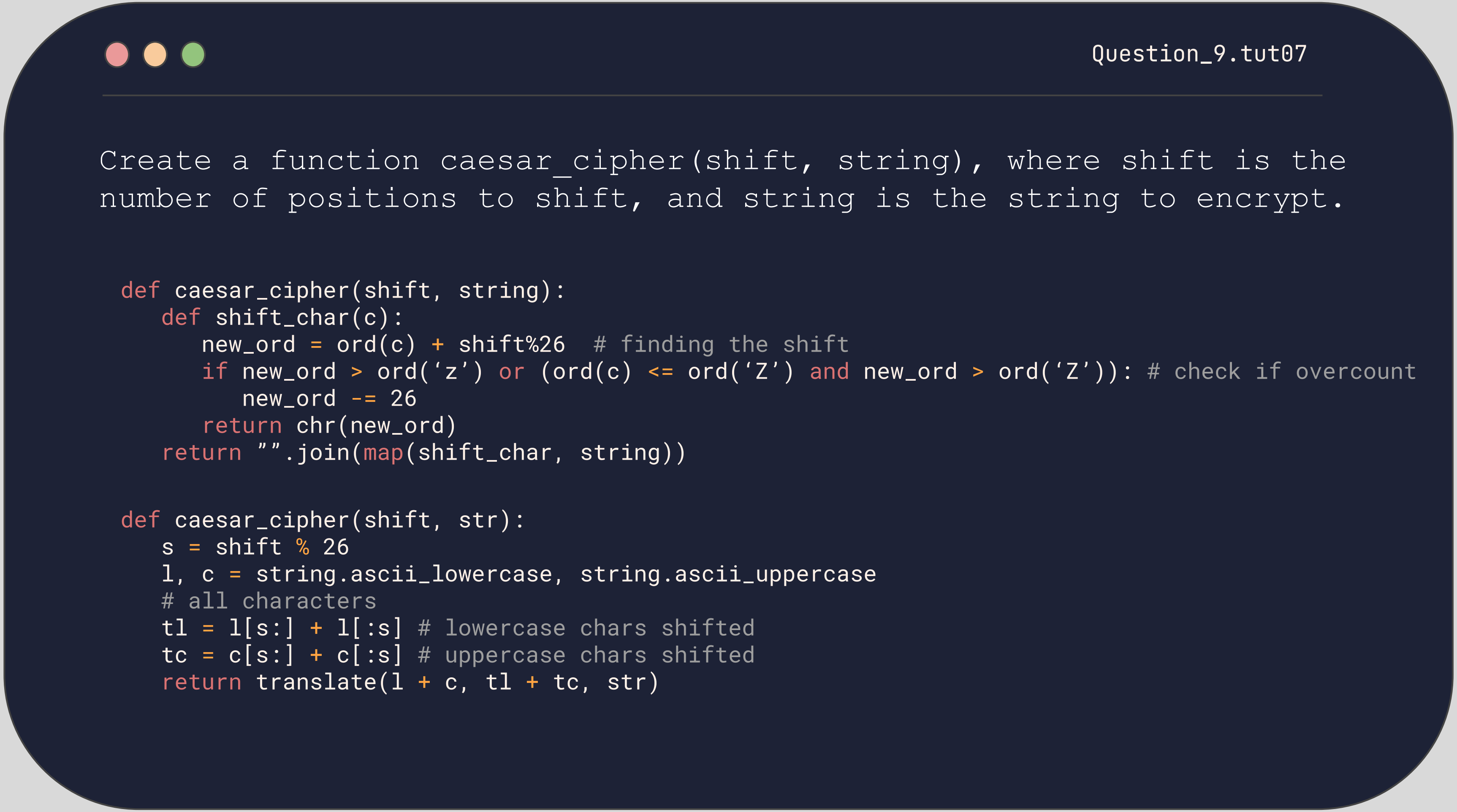


Question_8.tut07

Create a character translation function translate that takes in 3 arguments: source, the set of characters to translate, destination, the set of characters to translate to, string, the string to perform the translation on.

```
def translate(src, dst, str):
   dic = {}
   for i in range(len(src)):
      dic[src[i]] = dst[i]
   return “”.join(map(lambda c: dic[c] if c in dic else c, str))
```

```
def translate(src, dst, str):
   dic = dict(zip(src, dst))
   return “”.join(map(lambda c: dic[c] if c in dic else c, str))
```



Question_9.tut07

Create a function caesar_cipher(shift, string), where shift is the number of positions to shift, and string is the string to encrypt.

```python
def caesar_cipher(shift, string):
    def shift_char(c):
        new_ord = ord(c) + shift%26  # finding the shift
        if new_ord > ord(‘z’) or (ord(c) <= ord(‘Z’) and new_ord > ord(‘Z’)): # check if overcount
            new_ord -= 26
        return chr(new_ord)
    return ””.join(map(shift_char, string))

def caesar_cipher(shift, str):
    s = shift % 26
    l, c = string.ascii_lowercase, string.ascii_uppercase
    # all characters
    tl = l[s:] + l[:s] # lowercase chars shifted
    tc = c[s:] + c[:s] # uppercase chars shifted
    return translate(l + c, tl + tc, str)
```

Thank You!!

The End

See you next lesson

BY: Zhu_Ming

Any Questions?