Dictionaries

Dictionaries are like sets, except that the "elements" of the dictionary are treated as keys, and a value is associated with that key. As in sets, the keys to dictionaries must be immutable and hashable objects. However, the values associated with the key can be anything, and can be mutable.

In [1]:
```
table = {} #Create empty dictionary
table[27] = 'my value'
table["dog"] = [1, 2, "three"]
print(table)
```

```
{27: 'my value', 'dog': [1, 2, 'three']}
```

In [2]:
```
table[27] = 3
print("table:", table)
```

```
table: {27: 3, 'dog': [1, 2, 'three']}
```

In [3]:
```
#Is key in a dictionary?
if 'dog' in table:
    table['cat'] = 'unhappy'
print("table:", table)
```

```
table: {27: 3, 'cat': 'unhappy', 'dog': [1, 2, 'three']}
```

In [4]:
```
# Iterative over elements in a dictionary. Any order can occur!
for key, val in table.items():
    print("key:", key, "val:", val)
```

```
key: 27 val: 3
key: cat val: unhappy
key: dog val: [1, 2, 'three']
```

In [5]:
```
# Remove element from a dictionary
del table[27]  #Exception if key is not in dictionary
print("table:", table)
```

```
table: {'cat': 'unhappy', 'dog': [1, 2, 'three']}
```

In [6]:
```
# Dictionary comprehensions also possible
{n: n**3 for n in range(8)}
```

Out[6]:
```
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343}
```

Some fun with sets and dictionaries...

Problem:

Return a new list with the 2nd instance of the first element that is repeated in the input list removed. The rest of the list should remain unchanged (be the same as the input).

Goal:

Our goal is to use this problem to build some familiarity with **sets and dictionaries** -- so the code versions below are written to uses sets and/or dictionaries. This is definitely not the only way to solve the problem, but it happens to also be the most efficient away. (At the very end, we'll also show a more "direct", but less efficient, solution which does not use sets or dictionaries.)

In [7]:
```
inp = [0, 12, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]

inps = ['zero', 'twelve', 'twelve', 'zero', 'twelve', 'twelve',
        'thirty four', 'fifty six', 'twenty three', 'eleven',
        'forty five', 'two', 'three', 'four', 'eleven', 'ten', 'twelve']
```

Version 1

In [8]:
```
def remove2ndInstancev1(input):
    print(input)

    #Create a dictionary with frequencies
    freqd = {}
    for val in input:
        freqd[val] = freqd.get(val, 0) + 1
        #The get() method returns the value of the key in the dictionary
        #if it exists, and otherwise returns the default value (2nd param)

    #Look for first repeated element
    for val in input:
        if freqd[val] >= 2:
            repeated = val
            break
    print("repeated:", repeated)

    #Look through the list to find where the 2nd instance of the repeat is
    index = len(input)
    count = 0
    for i in range(len(input)):
        if input[i] == repeated:
            count += 1
        if count == 2:
            index = i
            break
    print("index:", index)

    output = input[:]        #make copy of input
    #Remove this 2nd instance if it exists from a copy of the input
    if index < len(output):
        output.pop(index)
    return output

remove2ndInstancev1(inp)
```

```
[0, 12, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]
repeated: 0
index: 3
```

Out[8]:
```
[0, 12, 12, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]
```

In [9]:
```
output = remove2ndInstancev1(inps)  #works also for lists of other kinds of elements
print(output)
```

```
['zero', 'twelve', 'twelve', 'zero', 'twelve', 'twelve', 'thirty four', 'fifty six', 'twenty thre
e', 'eleven', 'forty five', 'two', 'three', 'four', 'eleven', 'ten', 'twelve']
repeated: zero
index: 3
['zero', 'twelve', 'twelve', 'twelve', 'twelve', 'thirty four', 'fifty six', 'twenty three', 'eleve
n', 'forty five', 'two', 'three', 'four', 'eleven', 'ten', 'twelve']
```

Revised Spec

Our procedure removed `0` (`'zero'`) from the list at index 3, since that is the first repeat of the "earliest" element. Is that what we want? Our specification is perhaps **ambiguous**. Arguably, we might want to remove `12` at index 2 since `12` appeared twice before `0` appeared twice. Let's clarify our spec: by 'first repeated element' we mean the element that appears twice **first**. Thus

```
inp = [0, 12, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]
```

should give

```
expect = [0, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12] # remove second 12
```

Version 2 -- remove the second appearance of the first element that makes a second appearance

In [10]:
```
def remove2ndInstancev2(input):
    print(input)

    #Create a dictionary whose values are frequencies AND indices of elements.
    #We store either the index of the first element or for repeated elements
    #we store the index of the 2nd instance (first repeat).
    #Ignore additional repeats.
    freqd = {}

    for i in range(len(input)):
        val = input[i]
        if val in freqd:
            freqd[val][0] += 1
            if freqd[val][0] == 2:
                freqd[val][1] = i
        else:
            freqd[val] = [1, i]
    print("\nfreqd =", freqd)

    #Get the index of the 2nd instance of the earliest appearing repeat.
    index = len(input)
    for val in input:
        entry = freqd[val]
        if entry[0] >= 2:
            index = min(index, entry[1])

    #Remove this 2nd instance
    output = input[:]        #make copy of input
    if index < len(output):
        output.pop(index)
    return output

remove2ndInstancev2(inp)
```

```
[0, 12, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]

freqd = {0: [2, 3], 34: [1, 6], 3: [1, 12], 4: [1, 13], 23: [1, 8], 56: [1, 7], 2: [1, 11], 10: [1,
15], 11: [2, 14], 12: [5, 2], 45: [1, 10]}
```

Out[10]:
```
[0, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]
```

Hurray! We correctly detect the second `12` as the first repeated element, and remove that.

But can we do better in terms of writing prettier and tighter code -- perhaps something using **sets** rather than **dictionaries**?

Version 3 -- simplified using a set rather than a dictionary, to detect first repeat

In [11]:
```
def remove2ndInstancev3(input):
    print(input)

    index = len(input)
    freq_set = set()
    for i, val in enumerate(input):
        if val in freq_set:
            index = i
            break
        freq_set.add(val)

    #Remove this 2nd instance
    output = input[:index] + input[index+1:]  # list slicing rather than list pop
    return output

remove2ndInstancev3(inp)
```

```
[0, 12, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]
```

Out[11]:
```
[0, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]
```

Version 4 -- simpler but less efficient

Here's a very short solution to the original problem (with the first interpretation of the spec). It's interesting to think about how this works, and its virtues and/or failings compared to the earlier versions. This operates directly on the input list, without making any auxiliary sets and dictionaries, so is a more direct solution of the earlier problem. But it doesn't use sets and dictionaries, and as a result, it is slow for large inputs (taking quadratic time instead of linear). Think about why!

In [12]:
```
def remove2ndInstancev4(input):
    print(input)
    for i in range(len(input)):
        ii = input.index(input[i],i+1)
        if ii != -1:
            return input[:ii] + input[ii+1:]
    return input[:]

remove2ndInstancev4(inp)
```

```
[0, 12, 12, 0, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]
```

Out[12]:
```
[0, 12, 12, 12, 12, 34, 56, 23, 11, 45, 2, 3, 4, 11, 10, 12]
```

Consider it a challenge to you to revise this to go with our revised spec, i.e., to remove the first repeated element (12)!