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

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

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

In [ ]:
```
# Iterate over keys in a dictionary. (Since python3.6,
# dicts are iterated in order of key insertion)
for key in table:
    print("key:", key)
```

In [ ]:
```
# Iterate over values in a dictionary
for val in table.values():
    print("val:", val)
```

In [ ]:
```
# Iterate over items in a dictionary
for key, val in table.items():
    print("key:", key, "-- val:", val)
```

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

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

get and setdefault

The `get` method enables one to return a default value if the key is not in the dictionary.

`setdefault` is a useful method for dictionaries that saves a lot of initialization code:

`d.setdefault(key, val)` is equivalent to:

```
if key not in d:
    d[key] = val
d[key]
```

In [ ]:
```
# get default value if key not in table
table = {}
val = table.get(32, [])
print(val)
```

In [ ]:
```
# get default value if key not in table
table = {}
table[32] = table.get(32, []).append(1)
print(table[32])
```

In [ ]:
```
# fixes bug from above
table = {}
table[32] = table.get(32, [])
table[32].append(1)
print(table[32])
```

In [ ]:
```
# simpler using setdefault
table = {}
table.setdefault(32, []).append(1)
print(table[32])
```

Practice

Write a procedure `remove_first_duplicate` that, given a list whose elements are integers or strings, removes (returns a new list with) the *first* occurrence of the element that first gets repeated in the list removed. If there are no duplicates, the output should be equal to the original list. Below are two example inputs you can use to verify your solution.

In [ ]:
```
inp = [0, 12, 7, 12, 0, 12, 12, 34, 7, 23]
# expected = [0, 7, 12, 0, 12, 12, 34, 7, 23]
inp_str = ['zero', 'twelve', 'seven', 'twelve', 'zero', 'twelve', 'twelve',
           'thirty four', 'seven', 'twenty three']
```

In [ ]:
```
def remove_first_duplicate(data):
    pass

print(inp)
print(remove_first_duplicate(inp))
```