1 point

1. Consider this code:

```
>>> d = {'a': 1, 'b': 2}
>>> # CODE MISSING HERE
>>> d
{'a': 1, 'c': 3, 'b': 2}
```

Write the missing assignment statement that that modifies the dictionary as shown. (Just write the assignment statement; don't write the **>>>** part.)

```
d['c'] = 3
```

---

1 point

2. Consider this code:

```
>>> d = {'a': 1, 'b': 2}
>>> # CODE MISSING HERE
>>> d
{'a': 1, 'b': 3}
```

Write the missing assignment statement that modifies the dictionary as shown. (Just write the assignment statement; don't write the **>>>** part.)

```
d['b'] = 3
```

---

1 point

3. Consider this code:

```
>>> d = {'a': [1, 3], 'b': [5, 7]}
# CODE MISSING HERE
>>> d
{'a': [1, 2, 3], 'b': [5, 7]}
```

Select the option(s) that would lead to this result. Hint: call `help` on `insert`, `append`, and `sort`.

- [ ] `d['A'].insert(1, 2)`
- [ ] `d['a'].append(2)`
- [ ] `d['a'].sort()`

- [ ] `d['a'].insert(2, 1)`
- [ ] `d['a'].insert(1, 2)`

---

1 point

4. Consider this assignment statement:

```
1  d = {'a': 1, 'c': 3, 'b': 2}
```

Select the expression(s) that evaluate to `True`.

- [ ] `'b' in "d"`
- [ ] `2 in d`
- [ ] `"b" in d`
- [ ] `'b' in d`

---

1 point

5. Consider this code:

```
1  d = {'a': [1, 3], 'b': [5, 7, 9], 'c': [11]}
```

Select the expression(s) that evaluate to **3**.

- [ ] `len(d) - 3`
- [ ] `len(d['b'])`
- [ ] `len(d)`
- [ ] `len(d['a' + 'c'])`

---

1 point

6. Consider this code:

```
1  tup = (1, 2, 3)
```

Select the expression(s) and statement(s) below that **result in an error**.

- [ ] `subtup = tup[0:2]`
- [ ] `tup[0:2] == (10, 30)`
- [ ] `tup.reverse()`
- [ ] `tup[-2] = 4`

---

1 point

7. Select the expression(s) that can be used as dictionary keys.

- [ ] `('single',)`
- [ ] `(1, 'fred', 2.0)`
- [ ] `{1: 2, 3: 4}`
- [ ] `['a', 'b']`

---

1 point

8. Consider this code:

```
d = {1: ['a', 'b', 'c'], 2: ['d', 'e'], 3: []}
```

Select the code fragment(s) that set variable **total** to the number of items in all the lists that occur as values in **d**.

- [ ]
```
L = []
for k in d:
    L.append(k)

total = len(L)
```

- [ ]
```
total = 0
for k in d:
    total = total + len(d[k])
```

- [ ]
```
total = 0
for k in d:
    total = total + k
```

- [ ]
```
L = []
for k in d:
    L.extend(d[k])

total = len(L)
```

---

1 point

9. This dictionary has 3 keys that are all the same. **Enter this in the Python shell:**

```
1  {1: 10, 1: 20, 1: 30}
```

Submit what the code above evaluates to; don't submit your answers to the thought questions below.

What we want you to think about: We haven't covered this situation in the videos; what do *you* think should happen? Do you think this should this cause an error? Should it discard some of the key/value pairs? If so, which one do you think it should keep? People who create programming languages have to make these kinds of decisions, and often there isn't a clear good choice.

{1: 30}

1 point

10. Consider this code:

```
1  L = [['apple', 3], ['pear', 2], ['banana', 3]]
2  d = {}
3  for item in L:
4      d[item[0]] = item[1]
```

What does this code do?

- Populates dictionary `L` where each key is the first item of each inner list of `d` and each value is the second item of that inner list.
- Reorders the items in the inner lists of `L`.
- Removes the items from `L` and populates dictionary `d` where each key is the first item of each inner list of `L` and each value is the second item of that inner list.
- Populates dictionary `d` where each key is the first item of each inner list of `L` and each value is the second item of that inner list.

1 point

11. Consider this code:

```
def eat(d):
    '''(dict of {str: int}) -> bool

    Each key in d is a fruit and each value is the quantity of     that fruit
      .

    REST OF DESCRIPTION MISSING HERE

    >>> eat({'apple': 2, 'banana': 3, 'pear': 3, 'peach': 1})
    True
    >>> eat({'apple': 0, 'banana': 0})
    False
    '''
    ate = False
    for fruit in d:
        if d[fruit] > 0:
            d[fruit] = d[fruit] - 1
            ate = True

    return ate
```

Select the most appropriate description below.

- Remove from `d` all fruits that have a value of 0 associated with them and return `True` if and only if the were no such fruits.
- Try to eat one of each fruit: reduce by 1 all quantities greater than 0 associated with each fruit in `d` and return `True` if and only if any fruit was eaten.
- Return `True` if and only if any fruit was eaten.
- Reduce by 1 all quantities greater than 0 associated with each fruit in `d`.

---

1 point

12. Consider the code:

```
def contains(v, d):
    ''' (object, dict of {object: list}) -> bool

    Return whether v is an element of one of the list values in    d.
    >>> contains('moogah', {1: [70, 'blue'], 2: [1.24, 'moogah'    , 90], 3
      .14: [80, 100]})
    True
    >>> contains('moogah', {'moogah': [1.24, 'frooble', 90], 3.    14: [80,
      100]})
    False
    '''

    found = False # Whether we have found v in a list in d.

    # CODE MISSING HERE

    return found
```

Select the code fragment(s) that make the function above match its docstring description.

```
for k in d:
    if v == k:
        found = True
```

```
for k in d:
    for i in range(len(d[k])):
        found = (d[k][i] == v)
```

```
for k in d:
    for i in range(len(d[k])):
        if d[k][i] == v:
            found = True
```

```
for k in d:
    if v in d[k]:
        found = True
```

I understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account. Learn more about Coursera's Honor Code

Simon Uribe-Convers

Submit Quiz
