Reading and Writing Files

1.

```
def find_dups(L):
    """ (list) -> set

    Return the number of duplicates numbers from L.

    >>> find_dups([1, 1, 2, 3, 4, 2])
    {1, 2}
    >>> find_dups([1, 2, 3, 4])
    set()
    """

    elem_set = set()
    dups_set = set()

    for entry in L:
        len_initial = len(elem_set)
        elem_set.add(entry)
        len_after = len(elem_set)
        if len_initial == len_after:
            dups_set.add(entry)

    return(dups_set)
```

2.

```
def mating_pairs(males, females):
    """ (set, set) -> set of tuple

    Return a set of tuples where each tuple contains a male from males and a
    female from females.

    >>> mating_pairs({'Anne', 'Beatrice', 'Cari'}, {'Ali', 'Bob', 'Chen'})
    {('Cari', 'Chen'), ('Beatrice', 'Bob'), ('Anne', 'Ali')}
    """

    pairs = set()
    num_gerbils = len(males)

    for i in range(num_gerbils):

        male = males.pop()
        female = females.pop()
        pairs.add((male, female),)

    return pairs
```

3.

```
def get_authors(filenames):
```

```
    """ (list of str) -> set of str

    Return a list of the authors in PDB files names appear in filenames.
    """

    authors = set()

    for filename in filenames:
        pdb_file = open(filename)

        for line in pdb_file:
            if line.lower().startswith('author'):
                author = line[6:].strip()
                authors.add(author)

    return authors
```

4.

```
def count_values(dictionary):
    """ (dict) -> int

    Return the number of unique values in dictionary.

    >>> count_values({'red': 1, 'green': 2, 'blue': 2})
    2
    """

    return len(set(dictionary.values()))
```

5.

```
def least_likely(particle_to_probability):
    """ (dict of {str: float}) -> str

    Return the particle from particle_to_probability with the lowest
probablity.

    >>> least_likely({'neutron': 0.55, 'proton': 0.21, 'meson': 0.03, 'muon':
0.07})
    'meson'
    """

    smallest = 1
    name = ''

    for particle in particle_to_probability:
        probability = particle_to_probability[particle]
        if probability < smallest:
            smallest = probability
            name = particle

    return particle
```

6.

```
def count_duplicates(dictionary):
    """ (dic) -> int

    Return the number of duplicate values in dictionary.

    >>> count_duplicates({'R': 1, 'G': 2, 'B': 2, 'Y': 1, 'P': 3})
    2
    """

    duplicates = 0
    values = list(dictionary.values())

    for item in values:

        # if an item appears at least 2 times, it is a duplicate
        if values.count(item) >= 2:
            duplicates = duplicates + 1

            # remove that item from the list
            num_occurrences = values.count(item)
            for i in range(num_occurrences):
                values.remove(item)

    return duplicates
```

7.

```
def is_balanced(color_to_factor):
    """ (dict of {str: float}) -> bool

    Return True if and only if color_to_factor represents a balanced color.

    >>> is_balanced({'R': 0.5, 'G': 0.4, 'B': 0.7})
    False
    >>> is_balanced({'R': 0.3, 'G': 0.5, 'B': 0.2})
    True
    """

    values = list(color_to_factor.values())
    total = sum(values)
    return total == 1.0
```

8.

```
def dict_interest(dict1, dict2):
    """ (dict, dict) -> dict

    Return a new dictionary that contains only the key/value pairs that occur
    in both dict1 and dict2.

    >>> dict_interest({'a': 1, 'b': 2, 'c': 3}, {'a': 1, 'd': 2, 'b': 2})
    {'a': 1, 'b': 2}
    """

    intersection = {}
```

```
    for (key, value) in dict1.items():
        if key in dict2 and value == dict2[key]:
            intersection[key] = value

    return intersection
```

9.

```
def db_headings(dict_of_dict):
    """ (dict of dict) -> set

    Return a set of the keys in the inner dictionaries in dict_of_dict.

    >>> db_headings({'A': {1: 'a', 2: 'b'}, 'B': {2: 'c', 3: 'd'}})
    {1, 2, 3}
    """

    inner_keys = set()

    for key in dict_of_dict:
        for inner_key in dict_of_dict[key]:
            inner_keys.add(inner_key)

    return inner_keys
```

10.

```
def db_consistent(dict_of_dict):
    """ (dict of dict) -> set

    Return whether all inner dictionaries in dict_of_dict contain the same
keys.

    >>> db_consistent({'A': {1: 'a', 2: 'b'}, 'B': {2: 'c', 3: 'd'}})
    False
    >>> db_consistent({'A': {1: 'a', 2: 'b'}, 'B': {2: 'c', 1: 'd'}})
    True
    """

    inner_keys_list = []

    # Build a list of list of keys
    for key in dict_of_dict:
        inner_keys = list(dict_of_dict[key].keys())
        inner_keys.sort()
        inner_keys_list.append(inner_keys)

    for i in range(1, len(inner_keys_list)):

        # If the number of keys is different.
        if len(inner_keys_list[0]) != len(inner_keys_list[i]):
            return False

        # If the keys don't match.
```

```
        for j in range(len(inner_keys_list[0])):
            if inner_keys_list[0][j] != inner_keys_list[i][j]:
                return False

    return True
```

11.

a.

```
def sparse_add(vector1, vector2):
    """ (dict of {int: int}, dict of {int: int} -> dict of {int: int})

    Return the sum of sparse vectors vector1 and vector2.

    >>> sparse_add({1: 3, 3: 4}, {2: 4, 3: 5, 5: 6})
    {1: 3, 2: 4, 3: 9, 5: 6}
    """

    sum_vector = vector1.copy()

    for key in vector2:
        if key in sum_vector:
            sum_vector[key] = sum_vector[key] + vector2[key]
        else:
            sum_vector[key] = vector2[key]

    return sum_vector
```

b.

```
def sparse_dot(vector1, vector2):
    """ (dict of {int: int}, dict of {int: int} -> dict of {int: int})

    Return the dot product of sparse vectors vector1 and vector2.

    >>> sparse_dot({1: 3, 3: 4}, {2: 4, 3: 5, 5: 6})
    20
    """

    dot = 0

    for key1 in vector1:
        if key1 in vector2:
            dot = dot + vector1[key1] * vector2[key1]

    return dot
```

c.

```
Since only non-zero entries are stored, will the last entry always be non-
zero?  If not, how will the last entry be represented in the dictionary?
```