U2BS4M1RV : Oh, good catch. Oops. PyCharm should have told me that one.
U5VGKQ2SY : ``` counter = Counter()

```
    for entry in queries:
        if _query_filter(entry.client, include, exclude):
            counter[entry[index_to_count]] += 1
    return counter
```

U5VGKQ2SY : walk me through this
U2BS4M1RV : Counter is the Counter from collections. Queries is a list of the Query namedtuple. _query_filter is essentially grep and returns a boolean as to whether to include that entry in the counter or not.
index_to_count is the index of the named tuple. That is where I would much prefer to say Query.query rather than entry[index_to_count]

U2BS4M1RV : Or, Query.client, I think it was in that example.
U2BS4M1RV : I have two functions, at present, that use the _counts_generic, rather than each repeating the whole function and specifying entry.query or entry.client from the namedtuple.
U2BS4M1RV : ```def counts_query(queries: list, include: list=None, exclude: list=None) -&gt; dict:

```
    """
    Counts queries and returns a Counter of all domains queries
    Filters are literal and must match exactly
    :param queries: list of Query namedtuples
    :param include: list of items to include, works as whitelist
    :param exclude: list of items to exclude, works as blacklist
    :return: Counter keyed to dns query
    """
    return _counts_generic(queries, 2, include, exclude)


def counts_client(queries: list, include: list=None, exclude: list=None) \
        -&gt; dict:
    """
    Counts client requests and returns a Counter of all clients
    Filters are literal and must match exactly
    :param queries: list of Query namedtuples
    :param include: list of items to include, works as whitelist
    :param exclude: list of items to exclude, works as blacklist
    :return: Counter keyed to client ip query
    """
    return _counts_generic(queries, 3, include, exclude)


def _counts_generic(queries: list, index_to_count=0, include: list=None,
                    exclude: list=None) -&gt; dict:
    if not include:
        include = []
    if not exclude:
        exclude = []
    counter = Counter()
    for entry in queries:
        if _query_filter(entry[index_to_count], include, exclude):
            counter[entry[index_to_count]] += 1
    return counter


def _query_filter(entry: str, include: list = None, exclude: list = None)\
        -&gt; bool:
    if include:
        if entry in include and entry not in exclude:
            return True
```

~~~~
    else:
        if entry not in exclude:
            return True
    return False
```

Don't know if more code helps.

U5VGKQ2SY : ```    if include:
        if entry in include and entry not in exclude:
            return True
    else:
        if entry not in exclude:
            return True
    return False
```
Does this work the same if:
```
if include:
        if entry in include and entry not in exclude:
            return True
elif entry not in exclude:
            return True
else:
    return False
```
?

U2BS4M1RV : Yeah, it would. I shouldn't program at night. :slightly_smiling_face:
U2BS4M1RV : Though, in yours the else isn't necessary.
U5VGKQ2SY : technically, this could all be brought down to:```
if entry not in include:
    return True
else:
   return False
```
right?

U2BS4M1RV : No. Include is a whitelist, if include is empty it should give all entries not in exclude.
U5VGKQ2SY : because in both of the first 2 if's you are demanding that `entry` not be an element of exclude
U5VGKQ2SY : okay,```
def _query_filter(entry: str, include: list = None, exclude: list = None)\
        -&gt; bool:
    if include:
        if entry in include and entry not in exclude:
            return True
    else:
        if entry not in exclude:
            return True
    return False
```

U5VGKQ2SY : this is yoru orginal code
U5VGKQ2SY : so if `include is None`, it goes to the `else` and checks that `entry` be an element of `exclude`
U5VGKQ2SY : right?
U5VGKQ2SY : sorry... NOT an element of `exclude`
U2BS4M1RV : Right.
U5VGKQ2SY : but if include NOT None, you are checking that `entry` not an element of `exclude`
U2BS4M1RV : ```    if include:
        if entry in include and entry not in exclude:
~~~~

```
            return True
    elif entry not in exclude:
        return True
    return False
```

I could remove the if include and include it in the next if

U5VGKQ2SY : I'm just saying that `if include:` doesn't seem to have any bearing on what the condition does.
U5VGKQ2SY : because entry still has to NOT be an element in `exclude`