U1NSCAY6R : its a bit that gets marked when the file is archived - which has some weird definitions
U5UQKCC06 : I'd have to check our admin, I'm not sure what system we're using. I think it's hosted by another company
U1NSCAY6R : sorry running out the door but if this isnt turned on on the share, you wont get back access times: <https://gallery.technet.microsoft.com/How-to-enable-File-and-af674be4>
U5UQKCC06 : Thanks for the info <@U1NSCAY6R>
This is probably a silly question, but do you by chance know if this was enabled if it would retro date everything accessed previously? This is probably a pipe dream, but maybe the data is stored locally on everyone's own PCs?

U3G7RJP61 : Anyone ever worked with Stripe to create a UK managed account?
U1NSCAY6R : <@U5UQKCC06> if auditing is the issue and it wasn't on then you probably wont. Unless someone replaced a file with their local version then maybe
U1NSCAY6R : Have you tried `os.path.getmtime(path)`?
U5UQKCC06 : I haven't. I'll test it
U5UQKCC06 : Same thing, pretty sure both methods are retrieving the same attribute
U2BS4M1RV : Anyone on tonight? Or maybe tag me if you see this in the morning?
How would I get the index of a namedtuple field through code without looking at the definition? This is so I can write a code that will use this named tuple by field name rather than index so if the indexes change later down the line with code changes it won't break, and for easier reading?

```
Query = namedtuple('Query', 'dt record_type query client')

def _counts_generic(queries: list, index_to_count=0, include: list=None,
                    exclude: list=None) -&gt; dict:
    if include is None:
        include = []
    if exclude is None:
        exclude = []

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

def counts_client(queries: list, include: list=None, exclude: list=None) \
        -&gt; dict:
    return _counts_generic(queries, 3, include, exclude)```

With the counts_client function, I am sending a hardcoded index right now, but I would like to send Query.client. As expected, using Query.client doesn't work here and raises a TypeError for tuple indices must be integers or slices, not property.

U5VGKQ2SY : 1st, this:```
 if include is None:
        include = []
    if exclude is None:
        exclude = []
```
to:
```
if not include:
   include = []
if not exclude:
   exclude = []
```

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) -> 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) \
        -> 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) -> 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)\
        -> 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?