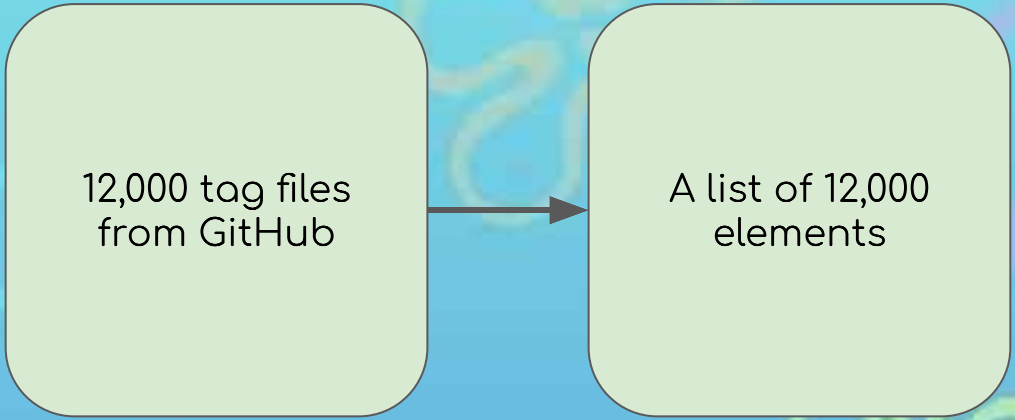
Organizing Plankton Classifications

Process Overview, Sat 14 Nov 2020

STEP 1
Get the tag files



Install the PyGithub package

```
pip install PyGithub
```

pip ★ The package manager for Python

PyGithub ★ The package we want to install

From the github module, only import the Github variable

```
from github import Github
```

We don't need all of the features of the github module,

we only need the Github features!

★ Some features we will use from the Github variable:

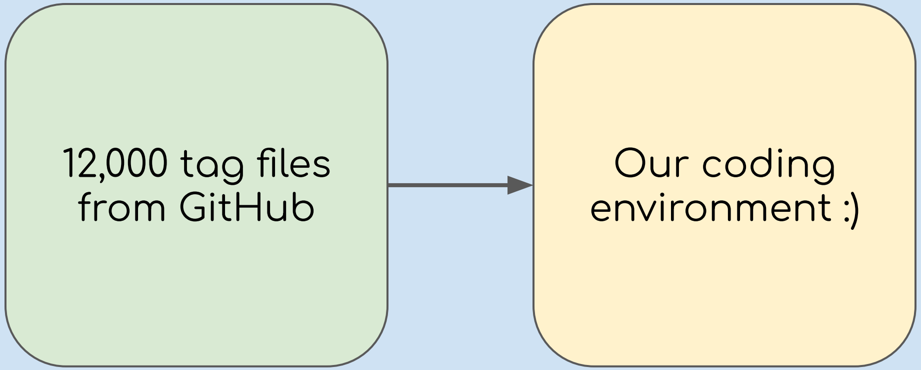
`Github(), get_repo(), get_contents()`

Populate the list of all tag files

⋆Access the repo⋆

★ Create a **`Github()`** object and use **`get_repo()`** to access Dr. Taniguchi's repository via its URL.

★ Use **`get_contents()`** to get the contents of the repository.



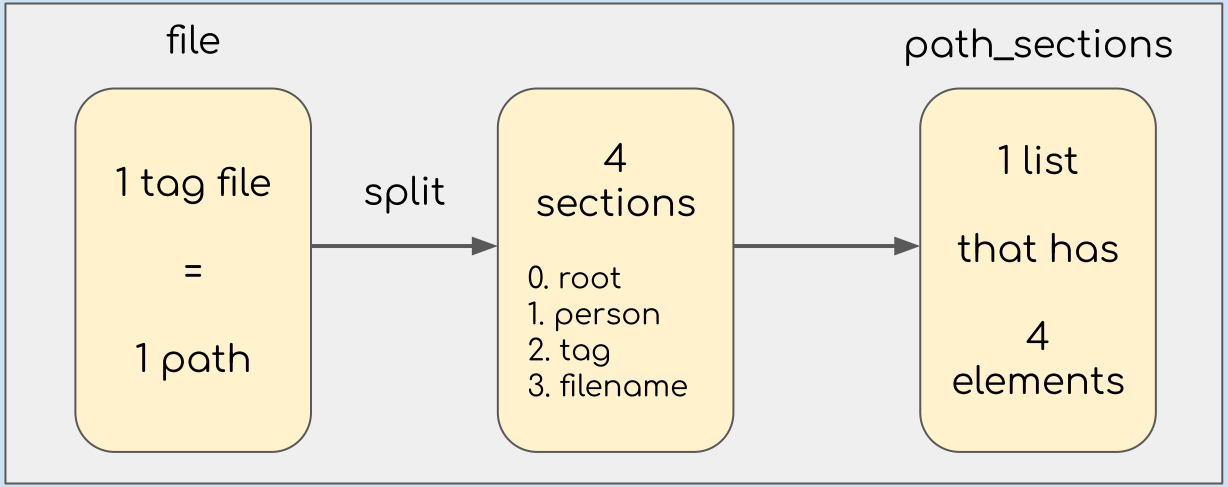
Then . . .

Populate the list of all tag files

★ Get the path of each file ★

- Find the path of each tag file → Split the path into 4 sections
    - Store the 4 sections into a 4-element list.

Each tag file will be represented by 1 list!



x 12,000 tag files

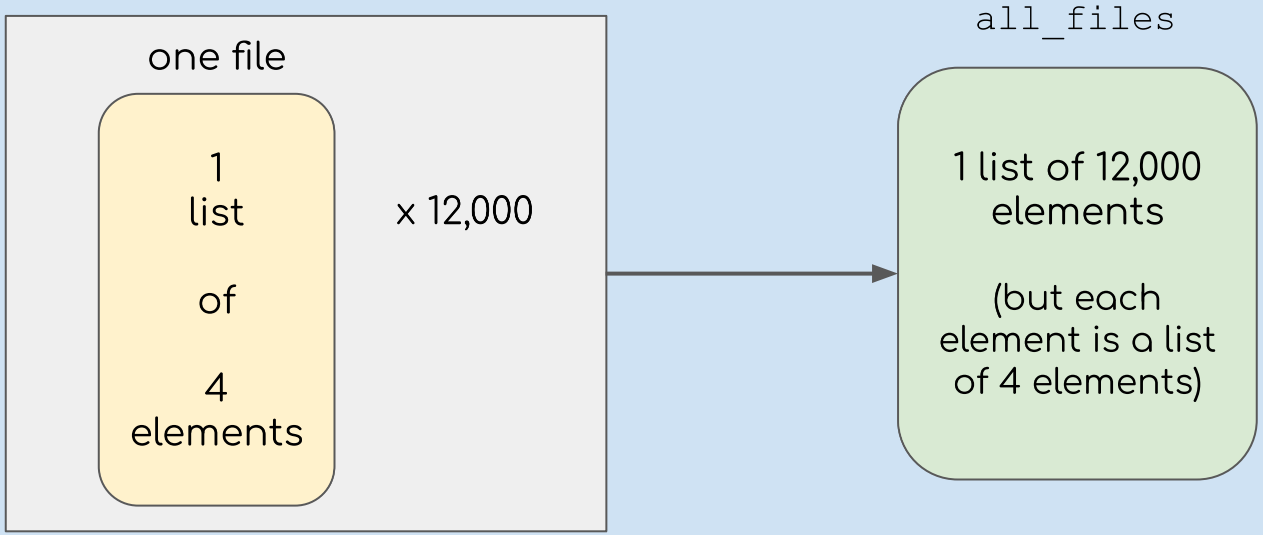
→ 12,000 lists!!!

Populate the list of all files

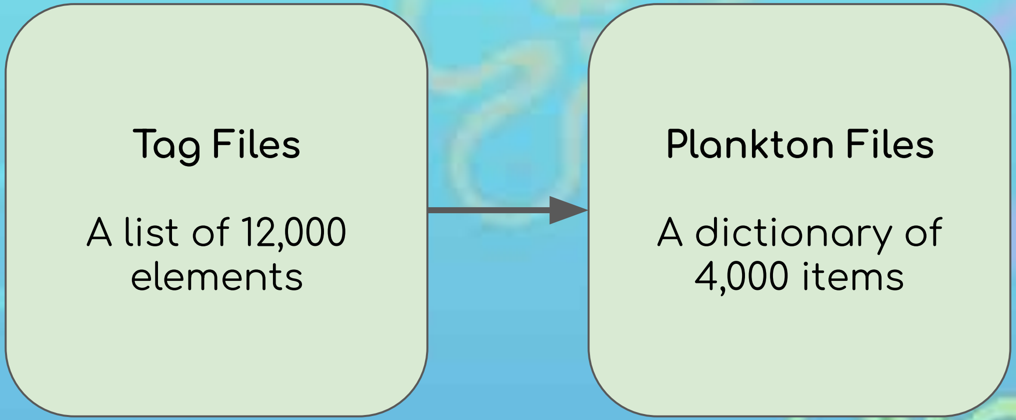
★ Finally, store all files in a list ★

Put each tag file's 4-element list into an **`all_files`** list

→ The **`all_files`** list will have 12,000 elements because there are 12,000 tags.


one file
1 list of 4 elements
x 12,000
all_files
1 list of 12,000 elements
(but each element is a list of 4 elements)

STEP 2
Populate the Dictionary of Plankton Files



FUNCTION: populate_dictionary

Parameters: a dictionary and a list of all files

★ Read each tag file and update one item of the dictionary each time.

Each item's key will be the name of the plankton file.

```
"SPCP2-1514880003-496630-000-2264-1552-48-72.jpg"
```

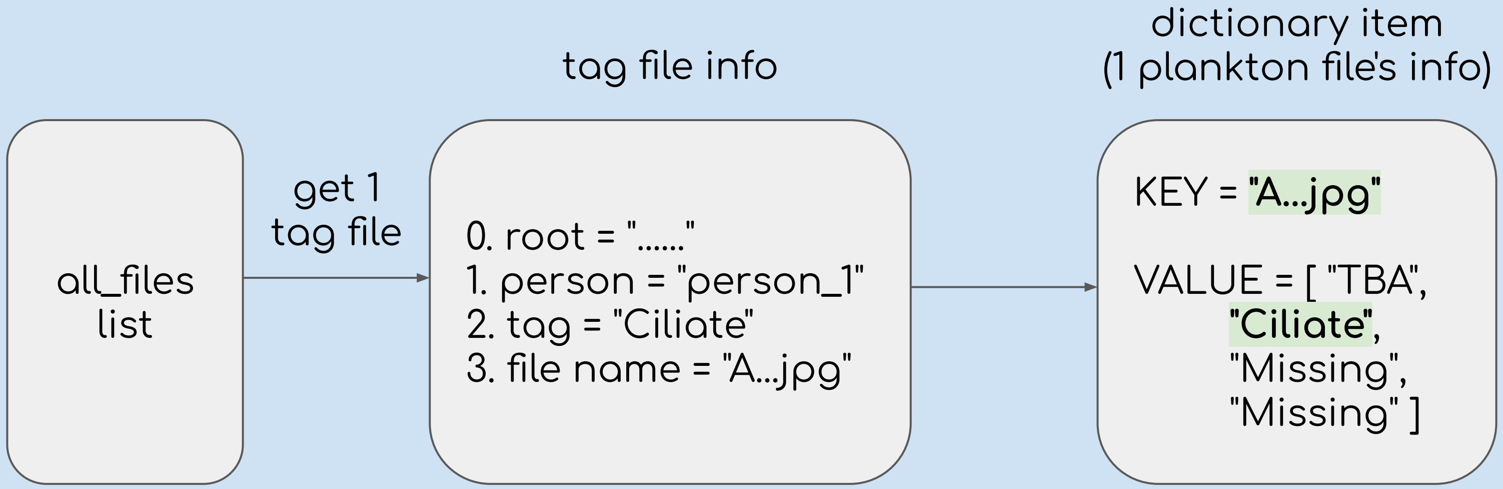
Each item's value will be a 4-element list.

```
[count_agreement, tag1, tag2, tag3]
```

★ There are 12,000 tag files and **each plankton file has 3 tags.**

→ **The dictionary will have 4,000 items.**

FUNCTION: populate_dictionary ★ example 1

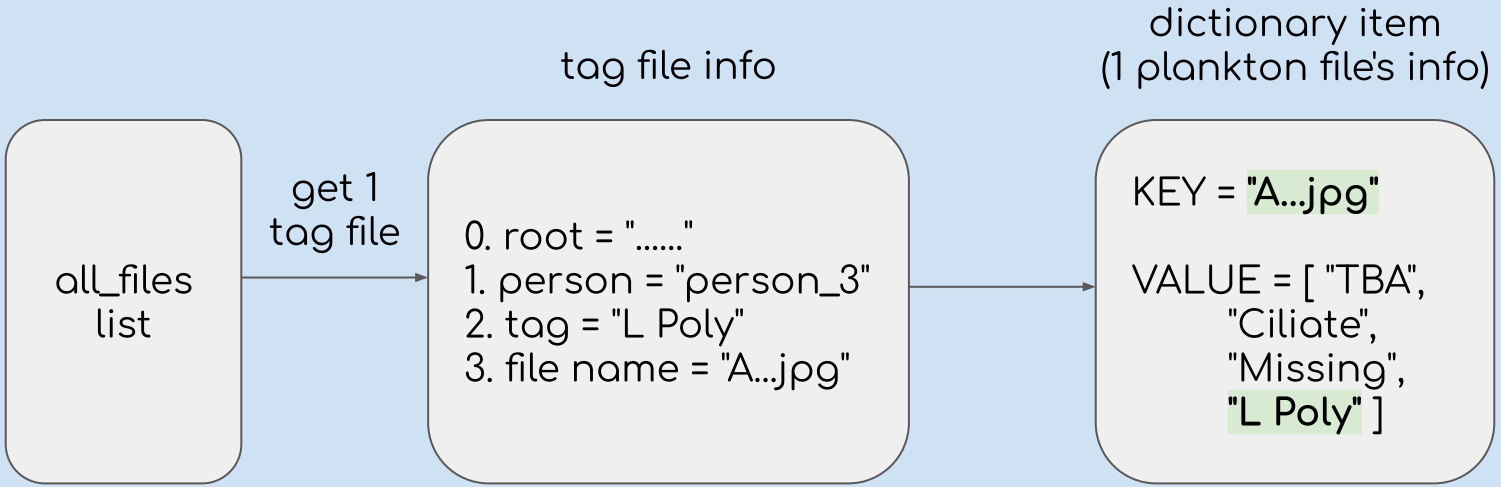


```
0. root = "......"
1. person = "person_1"
2. tag = "Ciliate"
3. file name = "A...jpg"
```

```
KEY = "A...jpg"

VALUE = [ "TBA",
          "Ciliate",
          "Missing",
          "Missing" ]
```

FUNCTION: populate_dictionary ★ example 2

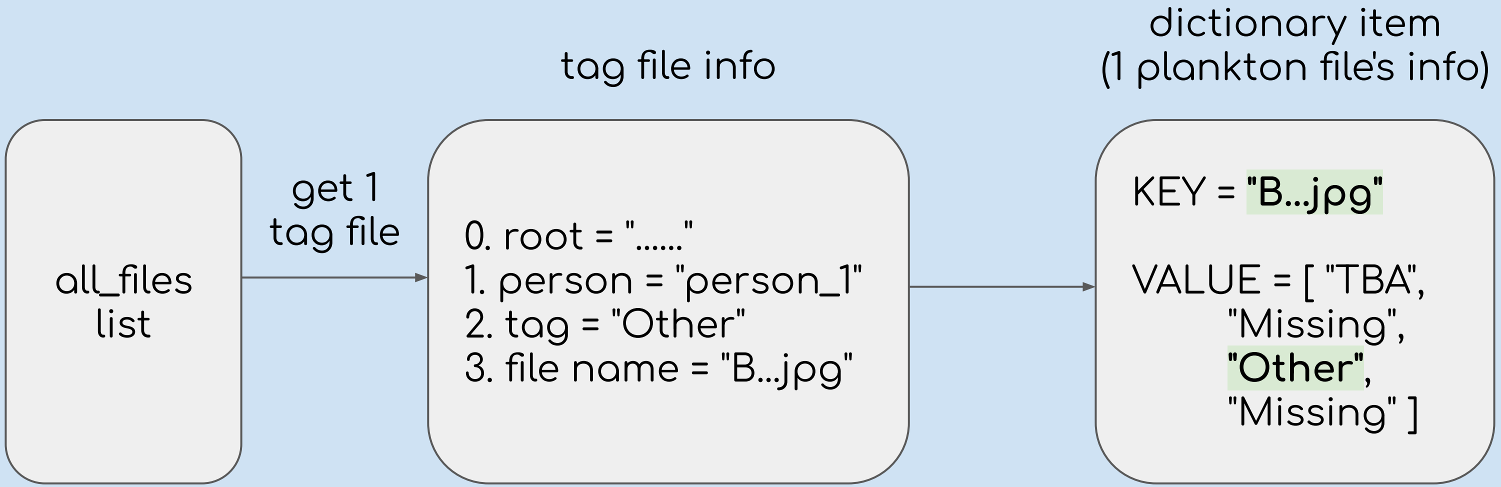


```
0. root = "......"
1. person = "person_3"
2. tag = "L Poly"
3. file name = "A...jpg"
```

```
KEY = "A...jpg"

VALUE = [ "TBA",
        "Ciliate",
        "Missing",
        "L Poly" ]
```

FUNCTION: populate_dictionary ★ example 3

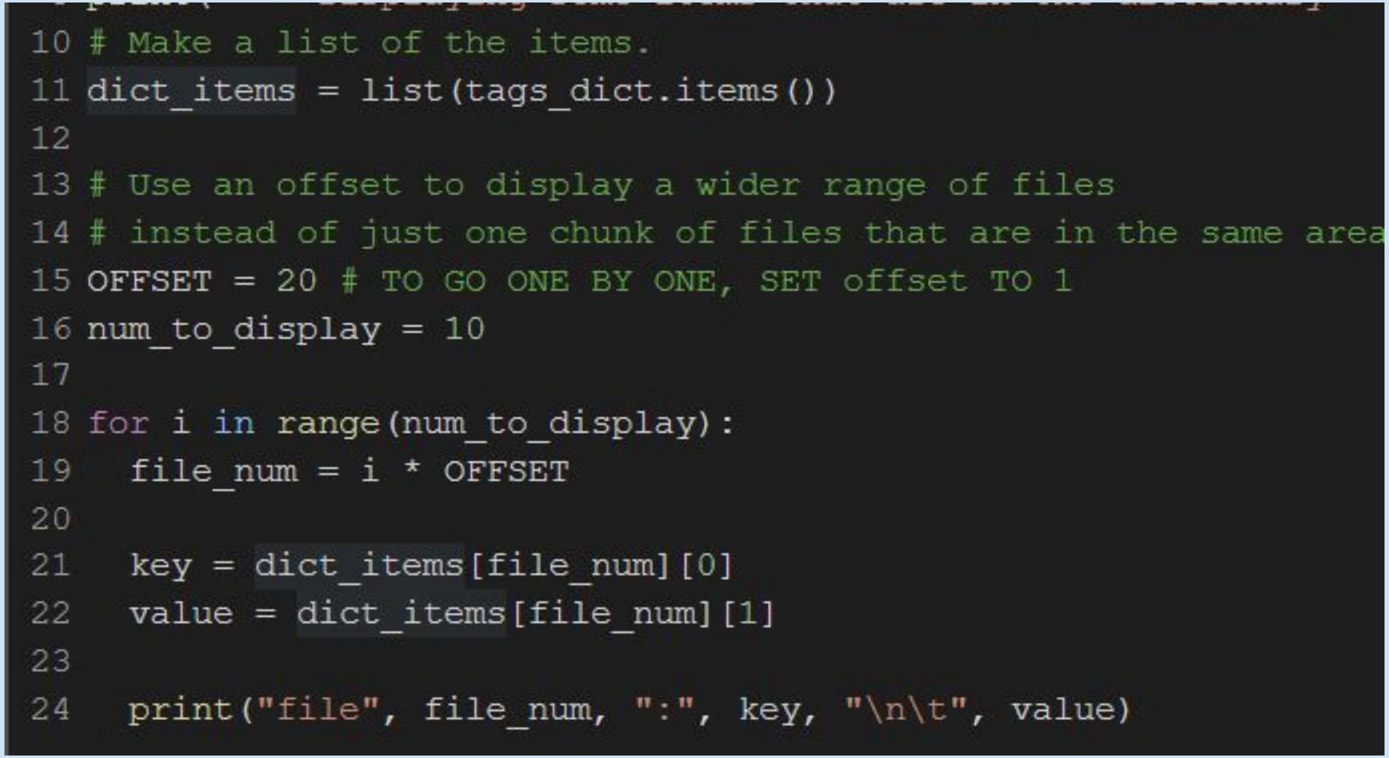


```
0. root = "......"
1. person = "person_1"
2. tag = "Other"
3. file name = "B...jpg"
```

```
KEY = "B...jpg"

VALUE = [ "TBA",
          "Missing",
          "Other",
          "Missing" ]
```

Display items from the dictionary



```
[illegible]
# Make a list of the items.
dict_items = list(tags_dict.items())

# Use an offset to display a wider range of files
# instead of just one chunk of files that are in the same area
OFFSET = 20 # TO GO ONE BY ONE, SET offset TO 1
num_to_display = 10

for i in range(num_to_display):
  file_num = i * OFFSET

  key = dict_items[file_num][0]
  value = dict_items[file_num][1]

  print("file", file_num, ":", key, "\n\t", value)
```

STEP 3
Count agreement for each file

FUNCTION: find_matched

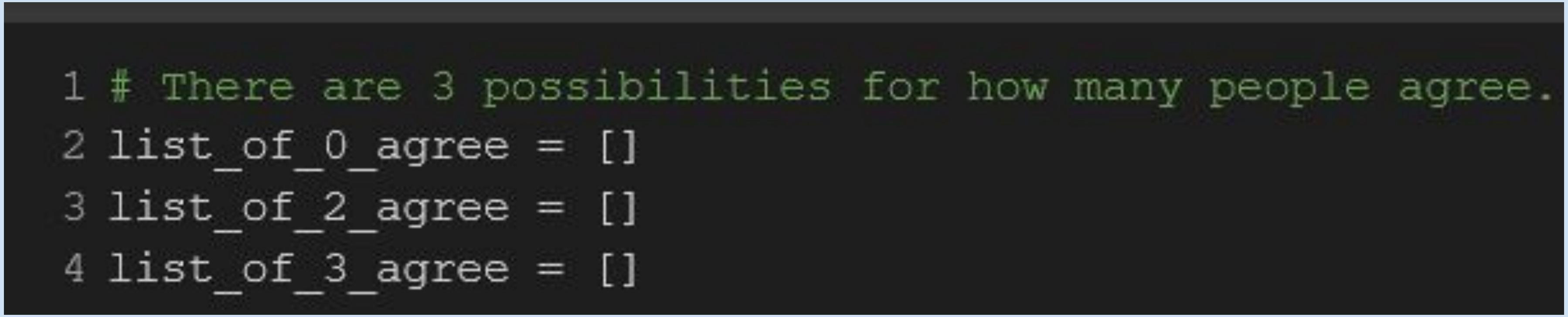
Parameters: tag1, tag2, tag3

Are **at least two** tags the same?

- Yes → Are **all three tags** are the same?
  - Yes, all 3 were matched. → `return 3`
  - No, there were only 2 matched. → `return 2`
- No → **None of the tags** match. → `return 0`

Lists: Grouping based on matches

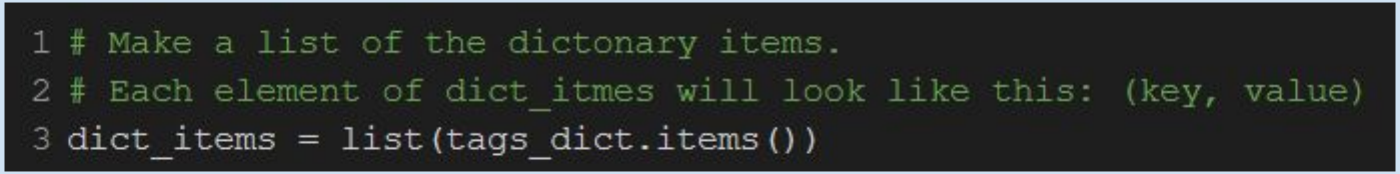
initialize an empty list for each group



```
# There are 3 possibilities for how many people agree.
list_of_0_agree = []
list_of_2_agree = []
list_of_3_agree = []
```

Make a list of all dictionary items

Dictionary items are accessed by the keys. But our keys are the plankton file names, and it would be more useful for us to **access items using index numbers instead.**



```
# Make a list of the dictonary items.
# Each element of dict_itmes will look like this: (key, value)
dict_items = list(tags_dict.items())
```

★ Example of accessing dictionary item **by the item's "key"**. The result is the item's "value".

`tags_dict["SPCP2-1514880003-496630-000-2264-1552-48-72.jpg"`

```
→ ["TBA", "Ciliate", "Questionable", Other"]
```

★ Example of accessing the same dictionary item, but by using **an index the dict_items list.**

The result is one variable with multiple parts (key, value), so it is called a tuple.

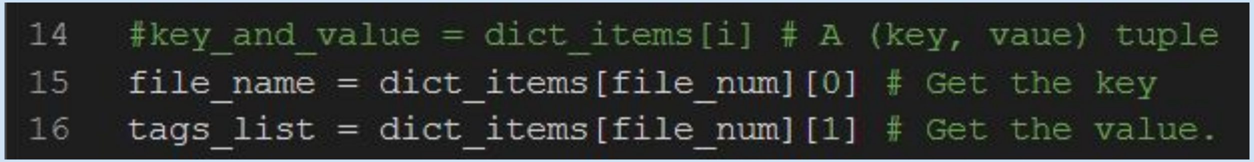
`dict_items[0]`

```
→ ("SPCP2-1514880003-496630-000-2264-1552-48-72.jpg", ["TBA", "Ciliate", "Questionable",
Other"])
```

Use the find_matched function

Use a **for loop** to look at each item (plankton file info) in the dictionary.

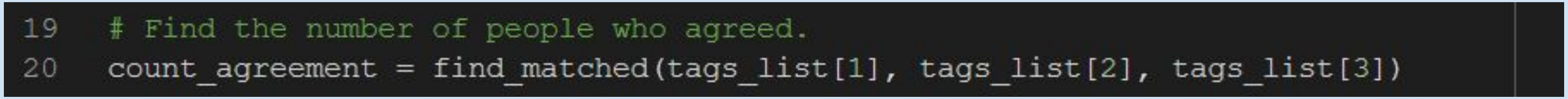
★ For each item (plankton file info), look at the file name and tags list.



```
#key_and_value = dict_items[i] # A (key, vaue) tuple
file_name = dict_items[file_num][0] # Get the key
tags_list = dict_items[file_num][1] # Get the value.
```

file_num is the index in the dict_items list

★ Use the **find_matched function** to find the count_agreement.



```
# Find the number of people who agreed.
count_agreement = find_matched(tags_list[1], tags_list[2], tags_list[3])
```

Then . . .

Update `tags_list[0]`

★ Update the "TBA" slot in the tags list to count_agreement.

```
tags_list[0] = count_agreement
```

example:

["TBA", "Ciliate", "Ciliate", "Other"] → [2, "Ciliate", "Ciliate", "Other]

And...

Update the grouping list

★ **Update one of the grouping lists** based on which group this item (plankton file) belongs in.

example:

[2, "Ciliate", "Ciliate", "Other]

→ `list_of_2_agree.append(file_name)`