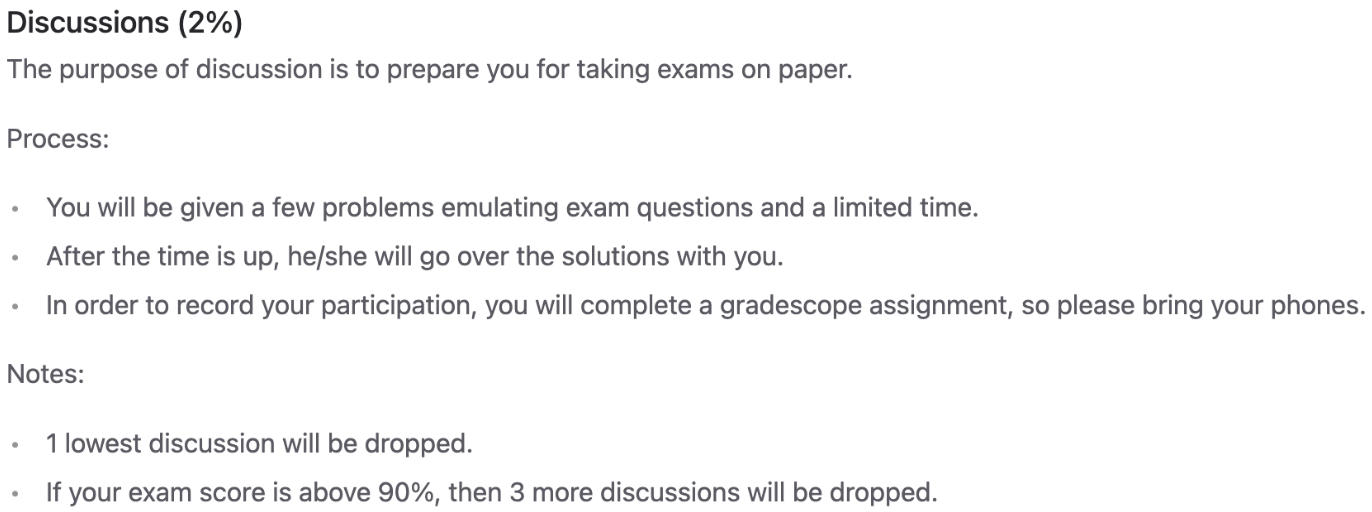
Discussion 2

DSC 20, Fall 2023

Agenda

- Discussion Attendance Update
- imports
- **Content**
    - in-place vs not
    - Indexing
    - List Methods
    - String Methods
    - Dictionaries
    - files
    - text processing
- Practice Questions
- PythonTutor (if we have time)

How discussion will work this quarter (UPDATE)



Discussions (2%)

The purpose of discussion is to prepare you for taking exams on paper.

Process:

- You will be given a few problems emulating exam questions and a limited time.
- After the time is up, he/she will go over the solutions with you.
- In order to record your participation, you will complete a gradescope assignment, so please bring your phones.

Notes:

- 1 lowest discussion will be dropped.
- If your exam score is above 90%, then 3 more discussions will be dropped.

Due to technical difficulties with Webclicker, how we record Discussion attendance will change. We will now be using a gradescope assignment that is open from beginning of discussion until 1 hour after - In order to receive credit, you need to respond to the multiple choice feedback questions and upload pictures to prove you attended lecture. You'll need to upload:

1. a picture of the slides with your student ID in the frame (or some ID)
2. the front and back of your worksheet

(which do you prefer?)

About Imports

"import" is another special keyword in python that has a unique function - it makes code from one body available in another. This can be something like a function from a different file or a whole package (ex. Pandas)

- We "ban" import in this class because many packages are too powerful (and out of the scope of this course)
    - ex. If we asked you to calculate Mean Squared Error on a dataset, you could just import a package for it
- When an import is necessary, we will explicitly import the package/module for you in the starter

Packages are powerful tools that you'll be using nonstop after this class, but for now they are a whole different beast that you will have to deal with later.

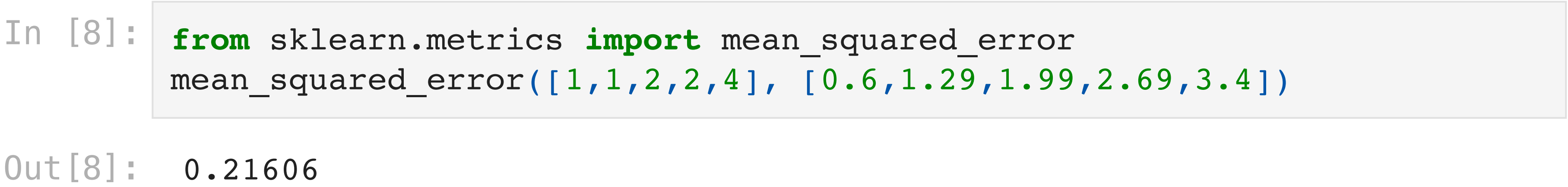
note: one of these makes you a better coder in the long run!

In [7]:

```
def MSE(y_1, y_2):
    sq_error = [(y_2[i] - y_1[i])**2 for i in range(len(y_1))]
    return sum(sq_error) / len(y_1)
MSE([1,1,2,2,4], [0.6,1.29,1.99,2.69,3.4])
```

Out[7]:

```
0.21606
```



In [8]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error([1,1,2,2,4], [0.6,1.29,1.99,2.69,3.4])
```

Out[8]:
```
0.21606
```

Content

in-place operations

Before we start exploring functions, it's important to understand what in-place operations are.

Definition: an operation is in-place if the result occurs directly on the original object, rather than a copy. Many in-place functions return None for an output.

What does this actually mean? The result of a not in-place function is a copy, the original object is not modified and the result has to be assigned to a variable to be retained. In-place functions modify the actual object passed in.

In [ ]:

```
lst = [1,2,3,4,5] # all of these work exactly the same on strings!
lst[::-1] # reverse
lst[1:] # every element after the first
lst[-3::2] # from the last 3 elements, take every other
```

In [25]:

```
lst = [1,2,3,4,5]
print('This result is temporary - unless I reassign \
lst to it, lst is not modified: ' + str(lst + [6]))
print("lst's current state: " + str(lst))
lst_new = lst + [6] # reassigning the result retains the output
print("the reassigned lst -> lst_new" + str(lst_new))
```

```
This result is temporary - unless I reassign lst to it, lst is n
ot modified: [1, 2, 3, 4, 5, 6]
lst's current state: [1, 2, 3, 4, 5]
the reassigned lst -> lst_new[1, 2, 3, 4, 5, 6]
```

In [21]:

```
lst = [1,2,3,4,5] # compare that to .append
print("The result of .append() is: " + str(lst.append(6))) # .append()
print("but we can see that the original lst is modified: " + str(lst))
```

```
The result of .append() is: None
but we can see that the original lst is modified: [1, 2, 3, 4,
5, 6]
```

Indexing/Slicing

Indexing/slicing refers to accessing specific element(s) from an iterable object. Two of the most common cases for this are lists and strings. Indexing results in a copy (unless reasssigned)!

- iterable[start:stop:skip] (start:inclusive, stop: NOT inclusive)
- not every section needs to be specified (can just use start or stop or skip)
- sub indexes can be applied (ex. lst[0][0] -> takes the first element of the first element)
- Trying to access an index that doesn't exist in the list will result in an error

In [51]:

```
lst = list(range(2,13))
print("original list: " + str(lst))
print("reversed list: " + str(lst[::-1]))
print("the 2nd to 4th element: " + str(lst[2:4]))
print("every third element from the 1st to 10th element: " + str(lst[1:
```

```
original list: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
reversed list: [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]
the 2nd to 4th element: [4, 5]
every third element from the 1st to 10th element: [3, 6, 9]
```

Broad Methods

- iterable.index(element, start, end) -> returns the index of the iterable where the argument is located, otherwise an error is raised.
- iterable.count(element) -> returns the number of times element occurs in the iterable.

In [5]:

```
lst = [5,4,3,2,1,0,0]
print(lst.index(2))
print(lst.count(0))
```

```
3
2
```

List Methods

- list.pop(index) -> removes the element at index. in-place operation; returns the value removed

- list.append(element) -> adds the element to the end of the list. in-place operation

- list.sort() -> sorts the list (as name implies). Can specify ascending or descending

- list.insert(index, element) -> inserts the element at index. in-place operation

In [49]:

```
lst = [5,4,3,2,1,0,0]
print("result of pop first element: " + str(lst.pop(0)))
lst.append(19)
print("after appending 19: "+ str(lst))
lst.sort()
print("after sorting list: " + str(lst))
lst.insert(2, 21)
print("after inserting 21 to index 2" + str(lst))
```

```
result of pop first element: 5
after appending 19: [4, 3, 2, 1, 0, 0, 19]
after sorting list: [0, 0, 1, 2, 3, 4, 19]
after inserting 21 to index 2[0, 0, 21, 1, 2, 3, 4, 19]
```

Checkpoint

Assume the following code has been ran:

```
In [7]: lst = [10, [12,13,11,10], (2,3,10,4), [4,6, 10, [10,11]], 1]
```

Which statement will extract 10 from the list? Select all that apply.

1. lst[0]

2. lst[1][-1]

3. lst[2][3]

4. lst[-2][-1][0]

Checkpoint Solution

Checkpoint Solution

```
In [8]:
print(lst[0])
print(lst[1][-1])
print(lst[2][3])
print(lst[-2][-1][0])
```

```
10
10
4
10
```

String Methods

- string.split(separator) -> splits a string into a list of elements on each separator
- "[string]".join(iterable) -> returns the iterable "joined" with the [string] in between each element
- string.lower()/string.upper() -> lowers/uppers all elements in a string
- string.strip() -> removes whitespace from beginning and end of the string
- string.format() -> method to format strings in sections

In [16]:

```
string = "Marina Langlois"
print(string.split())
print('-'.join(string.split()))
print(string.upper())
print("{} is the professor for {}.".format("marina", "DSC20"))
```

```
['Marina', 'Langlois']
Marina-Langlois
MARINA LANGLOIS
marina is the professor for DSC20.
```

Checkpoint

Assume the following code has been ran:

```
In [10]: statement = "Marina Langlois is the best DSC20 professor ever."
         temp = ' '.join(statement.split())
```

Is the following statement True or False? temp is equal to statement

Checkpoint Solution

Checkpoint Solution

```
In [11]: temp
```

```
Out[11]: 'Marina Langlois is the best DSC20 professor ever.'
```

Dictionaries

- Mutable storage of key, value pairs
- Can store any data type, multiple at a time
- Elements are acccessed via keys
- keys must be **hashable** and **unique**

methods

- accessing keys (as a list) -> dict.keys()
- accessing values (as a list) -> dict.values()
- accessing key,value pairs (as a list of tuples) -> dict.items()

note: hashablility correlates to the stability of the data - essentially, **data that can't change is hashable** (int, str, tuple, etc.) while **data that can change is not hashable (list, dictionary). Basically, it's all about mutability!**

In [39]:

```
hash('marina langlois')
```

Out[39]:

```
5351313138256952545
```

In [40]:

```
hash([1,2,3])
```

```
----------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
Cell In[40], line 1
----> 1 hash([1,2,3])

TypeError: unhashable type: 'list'
```

Checkpoint

Assume the following code has been ran:

In [14]:

```python
dct = {"nikki":"nikki", ('a','b','c'):[1,2,3], 'max':'nikki'}
```

Which of the following statements will extract "nikki" from the dictionary? Select all that apply.

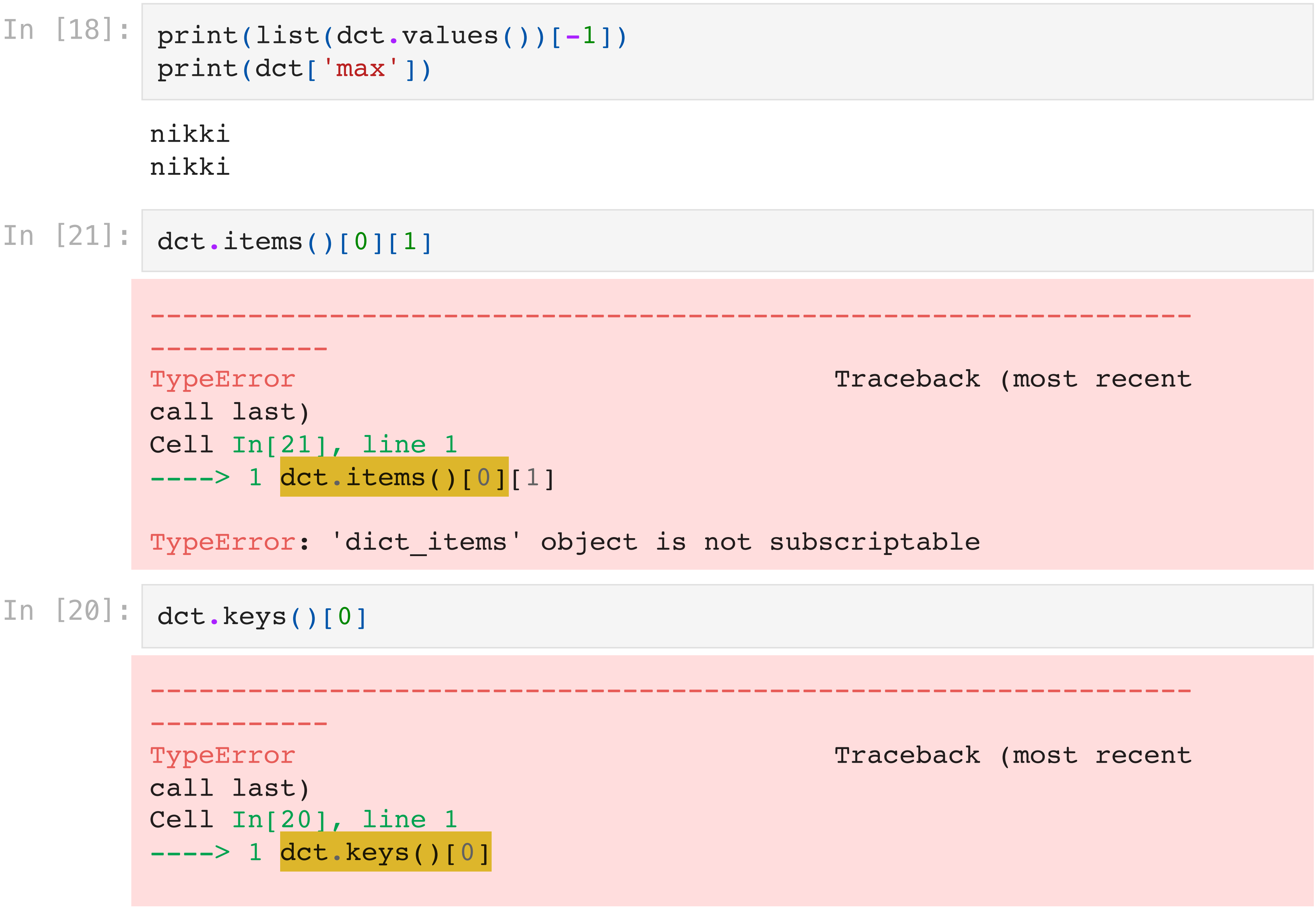
1. dct.items()[0][1]

2. dct.keys[0]

3. list(dct.values())[-1]

4. dct['max']

Checkpoint Solution



In [18]:
```
print(list(dct.values())[-1])
print(dct['max'])
```

```
nikki
nikki
```

In [21]:
```
dct.items()[0][1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 1
----> 1 dct.items()[0][1]

TypeError: 'dict_items' object is not subscriptable
```

In [20]:
```
dct.keys()[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 1
----> 1 dct.keys()[0]
```

```
TypeError: 'dict_keys' object is not subscriptable
```

Files

- storage for data (think csv's from DSC10, txt's from assignments, etc.)
- unique methods to access within code

Access Modes

Write : 'w' -> every time the file is opened in write mode, the file is wiped. Calling file.write() will add in your data.

Append : 'a' -> file.write() will append your data to what existed in the file beforehand.

Read : 'r' -> no writing privilege, can only pull the data from the file with relevant methods.

note: If you try to open a file in write mode that doesn't exist, python will create it.

note: you should almost NEVER use the eval() function when opening files you don't trust -> eval() automatically runs any code without protections (you could get a virus, or worse).

In [48]:

```
# method 1
file_object = open('files/disc.txt', 'w')
file_object.write('Marina Langlois')
file_object.close() # required for this method
```

Files

- storage for data (think csv's from DSC10, txt's from assignments, etc.)
- unique methods to access within code

Access Modes

Write : 'w' -> every time the file is opened in write mode, the file is wiped. Calling file.write() will add in your data.

Append : 'a' -> file.write() will append your data to what existed in the file beforehand.

Read : 'r' -> no writing privilege, can only pull the data from the file with relevant methods.

note: If you try to open a file in write mode that doesn't exist, python will create it.

note: you should almost NEVER use the eval() function when opening files you don't trust -> eval() automatically runs any code without protections (you could get a virus, or worse).

In [48]:

```
# method 1
file_object = open('files/disc.txt', 'w')
file_object.write('Marina Langlois')
file_object.close() # required for this method
```

In [57]:

```
# method 2
with open('files/data.txt', 'w') as f:
    f.write('Marina Langlois is the DSC20 professor\nSuraj Rampure is t
```

Text Processing

reading data:

- file.read() -> reads in all the data as a single string
- file.readline() -> reads in data line by line (has to be recalled)
- file.readlines() -> reads in all the data as a list where each line is another element of the list

After reading in the data, you can transform it however you'd like, and then rewrite it back into the file using .write() (if this is relevant).

In [67]:

```python
with open('files/data.txt', 'r') as f:
    # notice to read the data, I have to open in 'r' mode first
    data = f.read()
print("the data in the file currently is: ")
print(data)

print()

with open('files/data.txt', 'w') as f:
    data = data.replace('DSC180A', 'DSC10')
    f.write(data)
print("the data in the file is now: ")
with open('files/data.txt', 'r') as f: print(f.read())
```

```
the data in the file currently is:
Marina Langlois is the DSC20 professor
Suraj Rampure is the DSC10 professor

the data in the file is now:
Marina Langlois is the DSC20 professor
Suraj Rampure is the DSC10 professor
```

practice questions

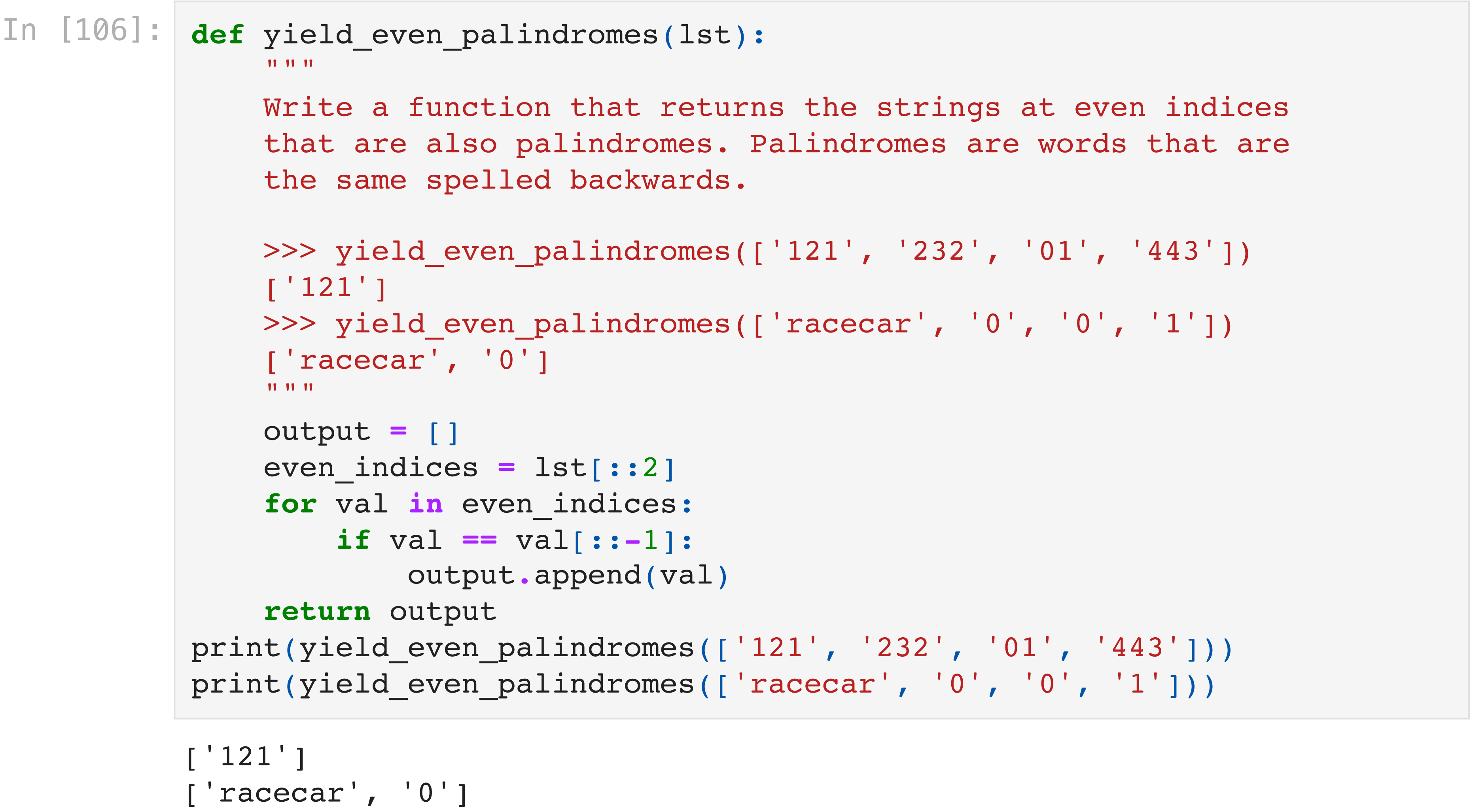
Time to do some practice questions! Take about 10-15 minutes to work on the questions.
Feel free to flag me down if you need help/clarification.

Make sure to handwrite! This is practice for your own sake.

YOU MAY KEEP YOUR WORKSHEETS!!

If you finish early, feel free to head over to gradescope and complete the discussion attendance assignment

practice question solutions



In [106]:

```python
def yield_even_palindromes(lst):
    """
    Write a function that returns the strings at even indices
    that are also palindromes. Palindromes are words that are
    the same spelled backwards.

    >>> yield_even_palindromes(['121', '232', '01', '443'])
    ['121']
    >>> yield_even_palindromes(['racecar', '0', '0', '1'])
    ['racecar', '0']
    """
    output = []
    even_indices = lst[::2]
    for val in even_indices:
        if val == val[::-1]:
            output.append(val)
    return output
print(yield_even_palindromes(['121', '232', '01', '443']))
print(yield_even_palindromes(['racecar', '0', '0', '1']))
```

```
['121']
['racecar', '0']
```

Assume the following code is ran:

```
In [33]: data = {
             "nikki": {"math": [20, 70, 40], "dsc": (100), "philosophy": 'socrat
              "bobby": {"math": [55, 42, 37, 2], "dsc": (88, 76), "philosophy":
               "max": {"math": [0, 22, 43, 17, 0], "dsc": (66), "philosophy": 'f
         }
```

Write a statement that results in nikki's highest math score.

```
In [34]: max(data['nikki']['math'])
```

```
Out[34]: 70
```

Write a statement that results in the 2nd score for bobby in dsc.

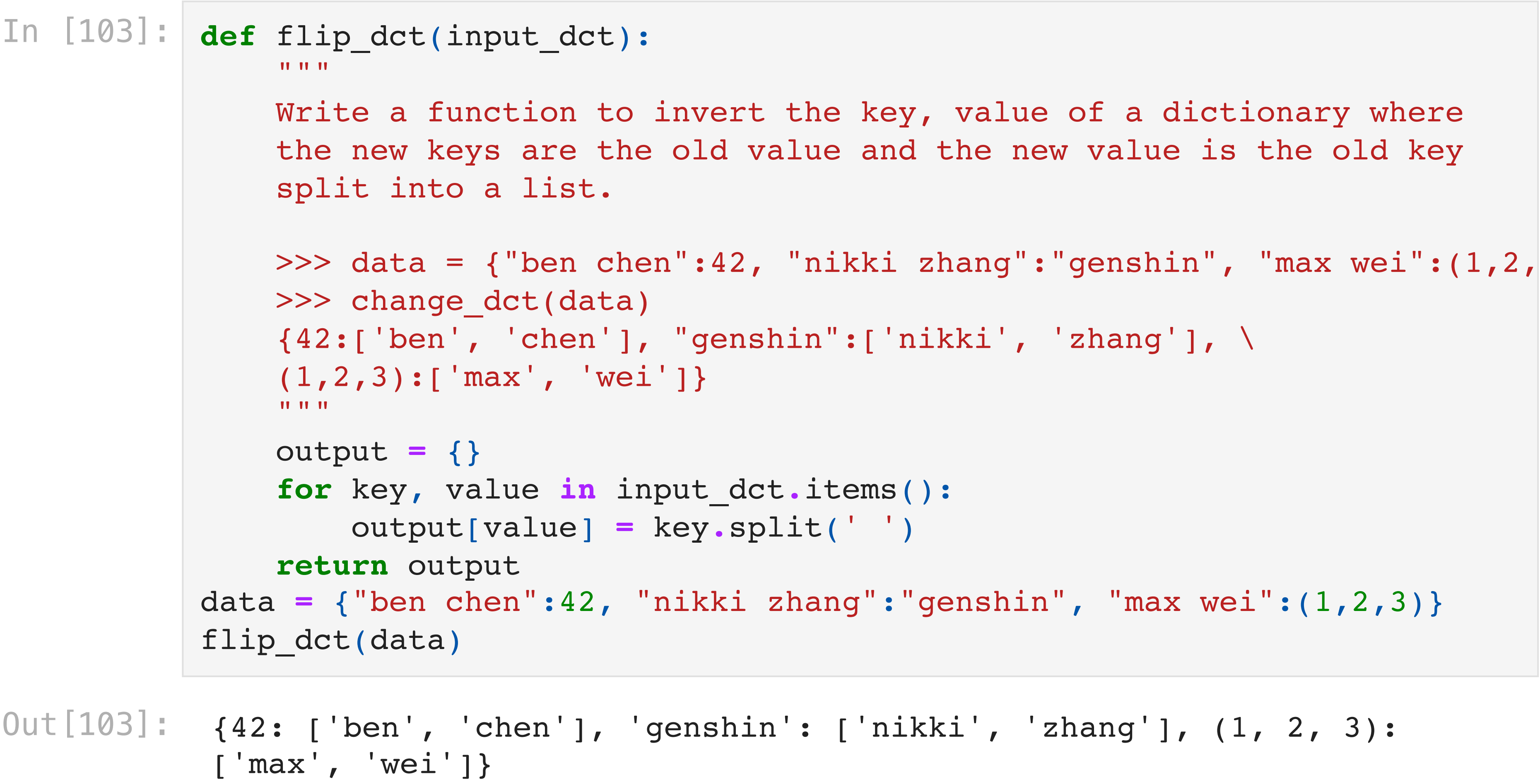
```
In [35]: data['bobby']['dsc'][1]
```

```
Out[35]: 76
```

Write a statement that results in max' philosophy last name.

```
In [36]: data['max']['philosophy'].split()[-1]
```

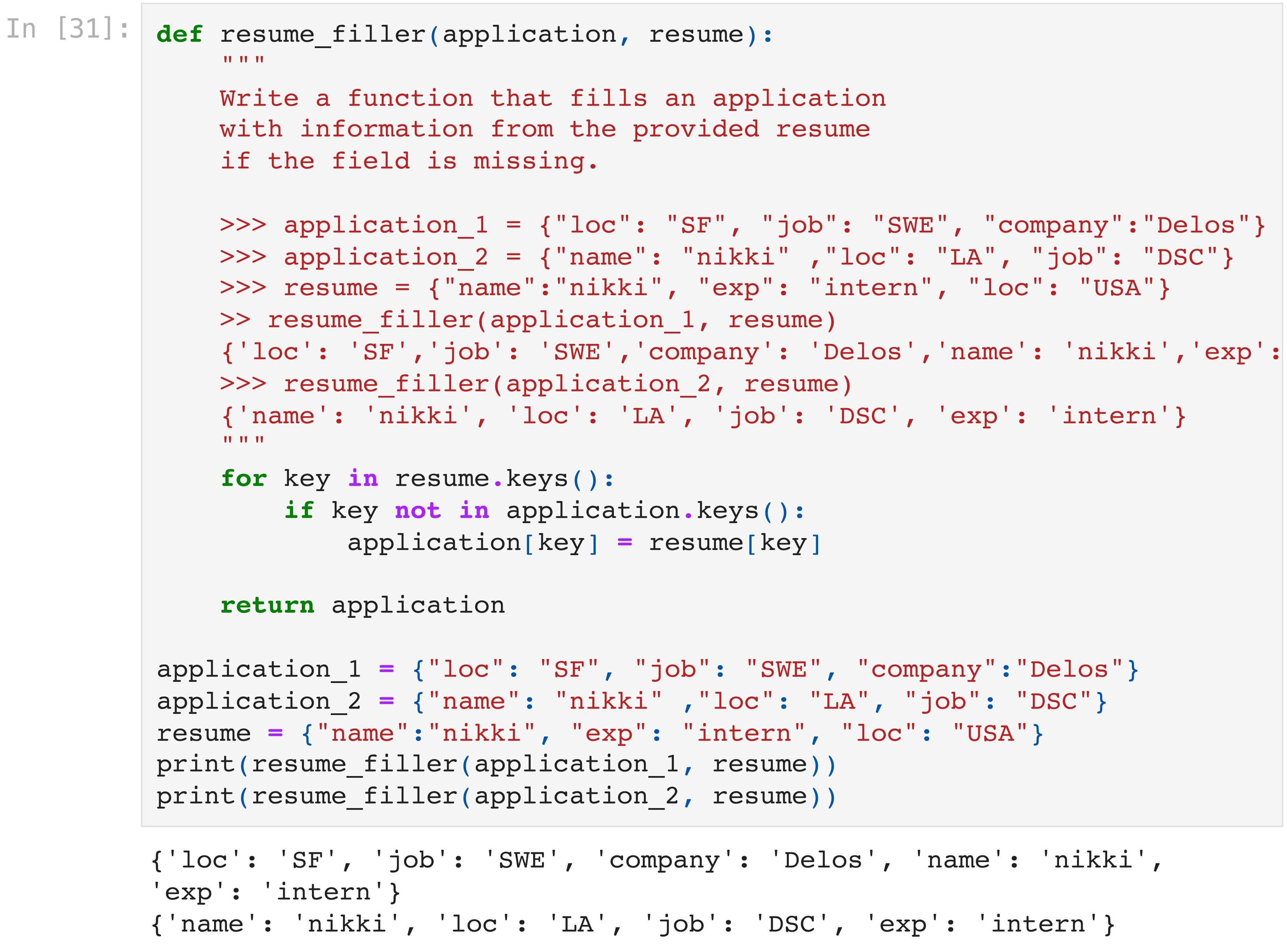
```
Out[36]: 'nietzsche'
```



```
In [103]: def flip_dct(input_dct):
              """
              Write a function to invert the key, value of a dictionary where
              the new keys are the old value and the new value is the old key
              split into a list.

              >>> data = {"ben chen":42, "nikki zhang":"genshin", "max wei":(1,2,
              >>> change_dct(data)
              {42:['ben', 'chen'], "genshin":['nikki', 'zhang'], \
              (1,2,3):['max', 'wei']}
              """
              output = {}
              for key, value in input_dct.items():
                  output[value] = key.split(' ')
              return output
          data = {"ben chen":42, "nikki zhang":"genshin", "max wei":(1,2,3)}
          flip_dct(data)

Out[103]: {42: ['ben', 'chen'], 'genshin': ['nikki', 'zhang'], (1, 2, 3):
           ['max', 'wei']}
```



```
In [31]: def resume_filler(application, resume):
             """
             Write a function that fills an application
             with information from the provided resume
             if the field is missing.

             >>> application_1 = {"loc": "SF", "job": "SWE", "company":"Delos"}
             >>> application_2 = {"name": "nikki" ,"loc": "LA", "job": "DSC"}
             >>> resume = {"name":"nikki", "exp": "intern", "loc": "USA"}
             >> resume_filler(application_1, resume)
             {'loc': 'SF','job': 'SWE','company': 'Delos','name': 'nikki','exp':
             >>> resume_filler(application_2, resume)
             {'name': 'nikki', 'loc': 'LA', 'job': 'DSC', 'exp': 'intern'}
             """
             for key in resume.keys():
                 if key not in application.keys():
                     application[key] = resume[key]

             return application

         application_1 = {"loc": "SF", "job": "SWE", "company":"Delos"}
         application_2 = {"name": "nikki" ,"loc": "LA", "job": "DSC"}
         resume = {"name":"nikki", "exp": "intern", "loc": "USA"}
         print(resume_filler(application_1, resume))
         print(resume_filler(application_2, resume))
```

```
{'loc': 'SF', 'job': 'SWE', 'company': 'Delos', 'name': 'nikki',
'exp': 'intern'}
{'name': 'nikki', 'loc': 'LA', 'job': 'DSC', 'exp': 'intern'}
```

Discussion Attendance

Take 2 minutes and head to gradescope to complete discussion attendance. The assignment is called Discussion 2 Participation.

Thanks for coming!