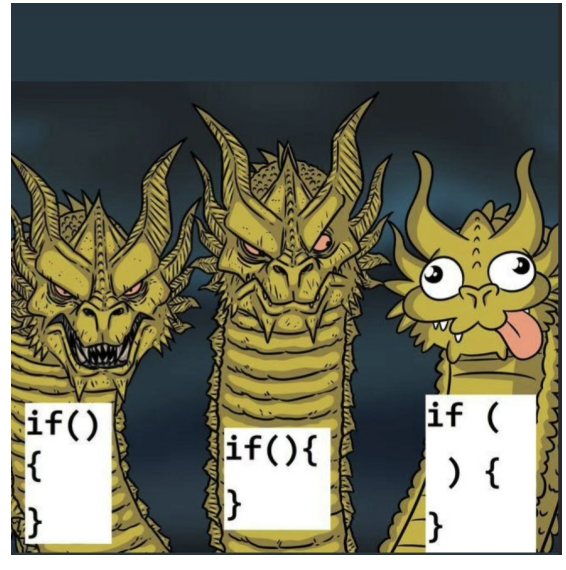
Discussion 3

DSC 20, Winter 2024

Meme of the week



Agenda

- Homework Autograder
- Homework Redemption Policy
- **Content**
    - Files
    - Text Processing
    - List Comprehension
    - Dictionary Comprehension
    - Assert Statements
- Practice Questions

Homework Autograder

PSA: Make sure that your code runs against our autograder after you submit on Gradescope. The autograder has an additional function as the logic check for the code you submit; if it cannot complete running your code, you will get a 0. If your code fails our autograder, make sure to fix your code and resubmit.

Cases that may break autograder:

- fatal syntax errors
- infinite loops

Homework Redemption Policy

Homework Resubmission/Regrade Policy:

- For each homework after the grades are released, you may resubmit the homework once to correct your mistakes within a week (7 days).
- You will get 80% of your score lost back for resubmission.
- If your original submission results in autograder failure (e.g. timed out, compile error, etc.), you may additionally ask for a regrade before the resubmission (all within 7 days of grade release).
  - For the first 2 homeworks compile errors and print vs. return issues can be regraded for free.
  - After the first 2 weeks compile errors cost 10 points; print vs. return issues cannot be regraded if the write-up explicitly says return or print (use the resubmission attempt).
  - If there are issues that are out of your control, we may regrade your submission for free.
- **Important**: Please test your code locally before uploading and wait for the autograder message before you leave Gradescope.
- See Regrade Requests for other general information.

For those who want to fix some basic errors that caused a lot of errors in their homework, refer to Homework Redemption Policy on the course website for more detail

Content

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 [51]:
```
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 [52]:
```
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)
```

```
the data in the file currently is:
Marina Langlois is the DSC20 professor
Suraj Rampure is the DSC180A professor
```

In [53]:
```
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 is now:
Marina Langlois is the DSC20 professor
Suraj Rampure is the DSC10 professor
```

In [7]:
```
# how can I ingest the data without the column names?
with open('files/names.csv', 'r') as f:
    print(f.read())
```

```
First Name,Last Name
Emily,Wilson
Jennifer,Sanchez
Denise,Ramos
Manuel,Barnett
Ashley,Hampton
```

In [8]:
```
with open('files/names.csv', 'r') as f:
    f.readline() # use readline to skip the first line
    print(f.read())
```

```
Emily,Wilson
Jennifer,Sanchez
Denise,Ramos
Manuel,Barnett
Ashley,Hampton
```

In [10]:
```
# If I want to ingest data in a more interpretable format
with open('files/names.csv', 'r') as f:
    data = f.readlines()
data
```

Out[10]:
```
['First Name,Last Name\n',
 'Emily,Wilson\n',
 'Jennifer,Sanchez\n',
 'Denise,Ramos\n',
 'Manuel,Barnett\n',
 'Ashley,Hampton']
```

Checkpoint

What happens when I try to open a file that doesn't exist in read mode?

A. Python will create the file

B. Nothing

C. Error

D. idk

Checkpoint Solution

In [20]:
```
with open('files/nonexistent.txt', 'w') as f:
    pass
```

In [19]:
```
with open('files/nonexistent.txt', 'r') as f:
    pass
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[19], line 1
----> 1 with open('files/nonexistent.txt', 'r') as f:
      2     pass

File /opt/anaconda3/envs/tutor/lib/python3.10/site-packages/IPython/core/interactiveshell.py:284, in _modified_open(file, *args, **kwargs)
    277 if file in {0, 1, 2}:
    278     raise ValueError(
    279         f"IPython won't let you open fd={file} by default "
    280         "as it is likely to crash IPython. If you know what you are doing, "
    281         "you can use builtins' open."
    282     )
--> 284 return io_open(file, *args, **kwargs)
```

List Comprehension

- Fancy, shorthand method of writing for loops
- Syntax changes depending on use case
- can be nested in each other, just like lists
- Can contain multiple for loops in one list comp
- Can also be a nested loop

  Syntax

  - [x for x in iterable]
  - [x for x in iterable if (condition)]
  - [x if (condition) else y for x in iterable]
  - [x if (condition) else y if (condition) else z for x in iterable]

In [17]:
```
# If I want to remove the newline from the previous output
data = [name.strip() for name in data[1:]]
data
```

Out[17]:
```
['Emily,Wilson',
 'Jennifer,Sanchez',
 'Denise,Ramos',
 'Manuel,Barnett',
 'Ashley,Hampton']
```

Checkpoint

What is the result of the following list comp?

In [ ]:
```
[(x*2, x**2) for x in range(0,10) if x**2 % 2 == 1]
```

A.[(0, 0), (2, 1), (4, 4), (6, 9), (8, 16), (10, 25), (12, 36), (14, 49), (16, 64), (18, 81)]

B. [(0, 0), (4, 4), (8, 16), (12, 36), (16, 64)]

C. [(2, 1), (6, 9), (10, 25), (14, 49), (18, 81)]

D. []

Checkpoint Solution

In [19]:
```
# C
[(x*2, x**2) for x in range(0,10) if x**2 % 2 == 1]
```

Out[19]:
```
[(2, 1), (6, 9), (10, 25), (14, 49), (18, 81)]
```

Dictionary Comprehension

- Fancy, shorthand method of populating dictionaries
- Syntax changes depending on use case

Syntax

- basically the same as list comp, but now it expects key:value
- can include a list comp!

In [20]:
```
# If I want to now create a dictionary of first name:last name
{x.split(',')[0]:x.split(',')[1] for x in data}
```

Out[20]:
```
{'Emily': 'Wilson',
 'Jennifer': 'Sanchez',
 'Denise': 'Ramos',
 'Manuel': 'Barnett',
 'Ashley': 'Hampton'}
```

Checkpoint

What is the result of the following dict comp?

In [ ]:
```
{x:[x + 2 for x in range(x)] for x in range(0,5) if x%2==0}
```

A. {0: [], 2: [2, 3], 4: [2, 3, 4, 5]}

B. {1: [2], 3: [2, 3, 4], 5: [2, 3, 4, 5, 6]}

C. {}

Checkpoint Solution

In [13]:
```
{x:[x + 2 for x in range(x)] for x in range(0,5) if x%2==0}
```

Out[13]:
```
{0: [], 2: [2, 3], 4: [2, 3, 4, 5]}
```

Assert Statements

- Used to evaluate written code
- **asserts** -> input validation (are the arguments the correct types?)
- Often combined with boolean functions (any(), all(), etc.)

In [54]:
```
def check_inputs(lst, word, number):
    assert isinstance(lst, list)
    assert isinstance(word, str)
    assert isinstance(number,int)
```

In [55]:
```
check_inputs([], 'a', 2)
```

In [56]:
```
check_inputs('a', 'a', 2)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent
call last)
Cell In[56], line 1
----> 1 check_inputs('a', 'a', 2)

Cell In[54], line 2, in check_inputs(lst, word, number)
      1 def check_inputs(lst, word, number):
----> 2     assert isinstance(lst, list)
      3     assert isinstance(word, str)
      4     assert isinstance(number,int)
```

Checkpoint

Given the following code, select all assert statements that logically check the input

In [12]:
```
def foo(input_lst):
    """
    function to flatten a nested list of strings
    """
    return [x for lst in input_lst for x in lst]
foo([['a'], ['b'], ['c']])
```

Out[12]:
```
['a', 'b', 'c']
```

A. assert isinstance(input_lst, list)

B. assert all([isinstance(lst, list) for lst in input_lst])

C. assert all([isinstance(x, str) for x in input_lst])

D. assert all([isinstance(x, str) for lst in input_lst for x in lst])

Checkpoint Solution

In [29]:
```
# A B D
input_lst = [['z']]
assert isinstance(input_lst, list)
assert all([isinstance(lst, list) for lst in input_lst])
assert all([isinstance(x, str) for lst in input_lst for x in lst])
```

practice question solutions

In [16]:
```
def convert_negs(lsts):
    """
    Write a function that uses list comprehension to
    convert negative numbers to positive and
    multiplies positive numbers by 2.

    >>> lsts = [[1,3,-11,6], [2,-5,-9,12], [3,19,-42]]
    >>> convert_negs(lsts)
    [[2, 6, 11, 12], [4, 5, 9, 24], [6, 38, 42]]
    """
    return [[element * -1 if element < 0 else \
    element * 2 for element in sublist] for sublist in lsts]

lsts = [[1,3,-11,6], [2,-5,-9,12], [3,19,-42]]
convert_negs(lsts)
```

Out[16]:
```
[[2, 6, 11, 12], [4, 5, 9, 24], [6, 38, 42]]
```

Write 3 assert statements for the previous function that validate the input

In [27]:
```
lsts = [[1]]
assert isinstance(lsts, list)
assert all([isinstance(sublist, list) for sublist in lsts])
assert all([isinstance(element, int) for sublist in lsts for element in
```

In [22]:
```
def dict_comp(key_lst, val_lst):
    """
    Write a function that takes in 2 lists
    and transforms them before converting them
    into a dictionary using dict comp.
    Transformations:
        1) keys should all be upper case
        2) values should be the square root of the
            original value

    >>> dict_comp(['max','ben','nikki'],[4,16,64])
    {'MAX': 2.0, 'BEN': 4.0, 'NIKKI': 8.0}
    """
    return {key.upper():val**0.5 for key,val in zip(key_lst,val_lst)}
dict_comp(['max','ben','nikki'],[4,16,64])
```

Out[22]:
```
{'MAX': 2.0, 'BEN': 4.0, 'NIKKI': 8.0}
```

In [4]:
```
def dict_comp(key_lst, val_lst):
    """
    Write a function that takes in 2 lists
    and transforms them before converting them
    into a dictionary using dict comp.
    Transformations:
        1) keys should all be upper case
        2) values should be the square root of the
            original value

    >>> dict_comp(['max','ben','nikki'],[4,16,64])
    {'MAX': 2.0, 'BEN': 4.0, 'NIKKI': 8.0}
    """
    return {key_lst[i].upper():val_lst[i]**0.5 for i in range(len(key_l
dict_comp(['max','ben','nikki'],[4,16,64])
```

Out[4]:
```
{'MAX': 2.0, 'BEN': 4.0, 'NIKKI': 8.0}
```

In [40]:
```
def class_reviews(reviews_filepath):
    """
    Write a function that finds all reviews that mention "DSC20".
    Each review is separated by a newline character "\n". Return
    the number of reviews that include "DSC20".

    >>> class_reviews('files/raw_reviews.txt')
    2
    """
    with open(reviews_filepath, 'r') as f:
        data = f.read()

    data = data.split('\n')
    return len([x for x in data if 'DSC20' in x])
class_reviews('files/raw_reviews.txt')
```

Out[40]:
```
2
```

In [49]:
```
def populate_csv(data, columns, filepath):
    """
    Write a function that takes in a dictionary of data and writes
    out a CSV to the provided filepath. CSV = Comma Separated Values.

    >>> data = ['marina langlois', 'ben chen']
    >>> columns = ['first name', 'last name']
    >>> populate_csv(data, 'files/class.csv')
    """
    with open(filepath, 'w') as f:
        write_in = ','.join(columns)
        strung_data = [','.join(name.split()) for name in data]
        write_in+='\n'
        write_in+=('\n'.join(strung_data))
        f.write(write_in)

data = ['marina langlois', 'ben chen']
columns = ['first name', 'last name']
populate_csv(data, columns, 'files/class.csv')
```

In [50]:
```
with open('files/class.csv', 'r') as f:
    print(f.read())
```

```
first name,last name
marina,langlois
ben,chen
```

Thanks for coming!