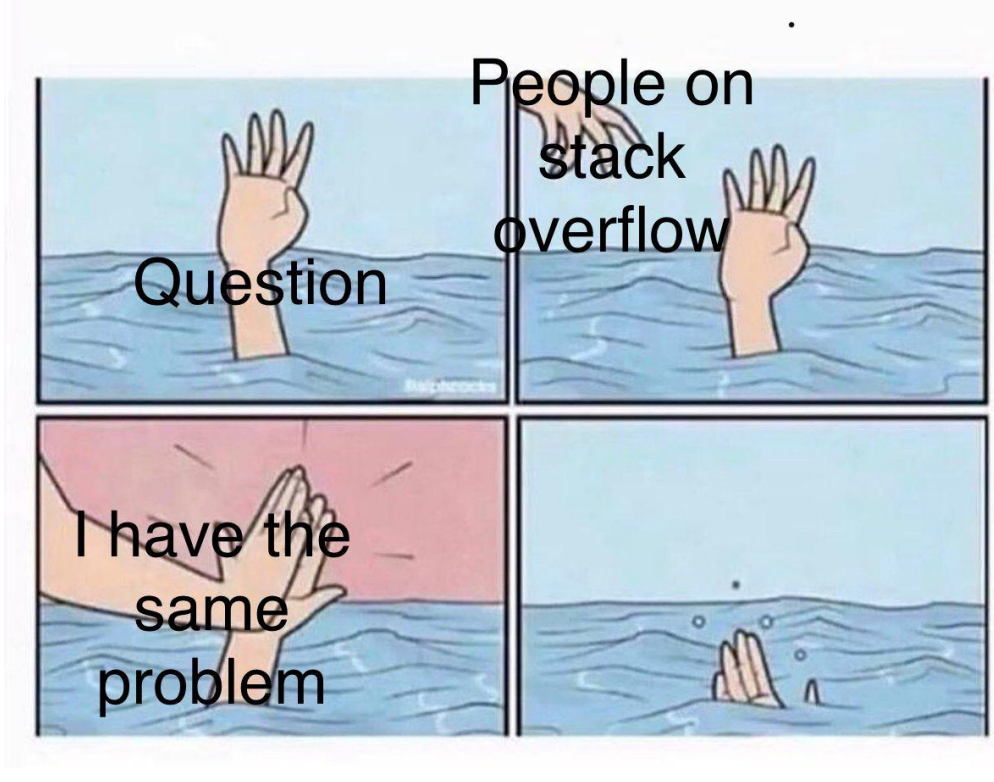
Discussion 4

DSC 20, Fall 2023

Midterm 1 Review 

Meme of the week



Logistics

- Midterm 1 takes place during lecture, Friday October 27th (CENTR 214 // normal lecture hall)
- No questions asked during exam
- Closed notes. Make sure to bring writing tools and your student ID(!!!)

Topics

- Basic Operations
- Basic Data Types
- Boolean Operators
- Conditional Statements
- short-circuits, return, print
- Doctests
- Asserts
- Loops
- Mutability
- Lists, tuples, sets
- List comprehension
- Slicing
- Dictionaries
- Dictionary comprehension (not explicit)
- Files

Basic Operations

+ - Numerical addition / concatenation operator

- - Numerical Subtraction operator

/ - Classic Division operator

// - Floor Division operator

***** - Numerical Multiplication / repetition operator

****** - Numerical Exponential operator

% - Numeric remainder operator

Basic Data Types

String - Data type for text

Int - Data type for whole numbers

Float - Data type for non-integers

Bool - True or False

note: **None** is technically its own type

Boolean Operators

Logical Operators (in order priority):

- **not** - reverses the outcome of the following expression
- **and** - all expressions compared with “and” must be True to evaluate True
- **or** - at least one expression compared with “or” must be True to evaluate True

Comparison Operators (generates booleans):

- **==** - equality check
- **!=** - inequality check
- **>, >=, <, <=** directional check

note: order of evaluation is overriden by parentheses (just like PEMDAS)

Checkpoint

What is the output of the following code?

In [ ]:
```
bool(-100)
```

Checkpoint Solution

In [3]:
```
bool(-100)
```

Out[3]:
```
True
```

Conditional Statements

if (boolean expression):

    //Do stuff

elif (other boolean expression):

note: elif is optional, can have as many elifs as necessary

    //Do other stuff

else:

note: else is optional, will execute only if the conditions in the “if” and “elif” statements are not true

    //Do other other stuff

Short-circuits, 'return', 'print'

- In python, expressions are evaluated from left to right
- With certain operations, a "short-ciruit" can happen if conditions are met
    - ex: True or 1/0 -> "or" only requires 1 True, so the expression is done evaluating before 1/0 can trigger an error.

- **'return'** is a very important and special keyword in python
    - all statements with return will short-circuit the function afterwards
    - It's python's functional way to pass a value out of a function
    - not every function needs a return (but most do)

- **'print'** displays some object (int, str, bool, etc.) onto a console
    - commonly used for debugging, sometimes integral to the purpose of a function.
    - print's result is None (it doesn't "return" a meaningful value)

Checkpoint

Which terminal command(s) runs all doctests?

1. python -m doctest code.py -v
2. python -i doctest code.py
3. python -m doctest code.py
4. python -m -v doctest code.py

Checkpoint Solution

1 - python -m doctest code.py -v

3 - python -m doctest code.py

Doctests

- Tests to check that your function works as intended
- denoted by the '>>> ' symbol (space included)!
- the line right after the '>>> ' represents the intended output
- well written doctests make sure your code is logically sound

to run doctests: terminal -> file location -> **python -m doctest [filename].py**

note: if you want to see the doctests you passed when you run them, add a '-v' at the end.

Assert Statements

- Used to evaluate written code
- **asserts** -> input validation (are the arguments the correct types?)
- Often combined with boolean functions (any(), all(), etc.) and list comp
- result should be a boolean (True = pass, False = error)
- check naive cases first, then more specific cases

Loops

Loops are used to **repeat computations** many times.

- Two types of loops:
    - While loop: Uses logical conditions, useful for when the number of iterations is unknown (as long as a condition is true, code will run).
    - For loop : Uses an iterable object (ex. list), usually for when the number repetition is known.

In [ ]:
```
while x is True:
    # do something
for value in x:
    # do something
```

Checkpoint

Which of the following are valid ways to create a list from 0-10 (inclusive)?

1. list(range(0,10))
2. list(range(10, -1, -1))[::-1]
3. list(range(0,11, 1))
4. None of the above

Checkpoint Solution

In [13]:
```
print("1: " + str(list(range(0,10))))
print("2: " + str(list(range(10, -1, -1))[::-1]))
print("3: " + str(list(range(0,11, 1))))
```

```
1: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Mutability

- Object is mutable if it can be changed after it is created
- If it can't, it is immutable
- Lists are **mutable**
- strings, tuples, and numbers are **immutable**

Lists, Tuples, Sets

List

- Mutable vector of values
- Can store any data type, multiple types at a time
- Elements are acccessed via indexing

Tuple

- **Immutable** vector of values
- all else equal to list

Set

- Mutable vector of values
- Only stores unique elements (removes duplicates)
- Order is not preserved

List Comprehension

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

  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]

Checkpoint

Which of the following list comprehensions will extract odd indexed elements from lst= [1,2,3,4,5,6] and convert them into a string and even indexed elements multiplied by 2?

1. [str(i) if i%2==1 else i*2 for i in lst]
2. [str(lst[i]) if i%2==1 else i*2 for i in range(len(lst))]
3. [str(lst[i]) if i%2==1 else i*2 for i in len(lst)]
4. None of the above

Checkpoint Solution

In [18]:
```
print('1: '+str([str(i) if i%2==1 else i*2 for i in lst]))
print('2: '+str([str(lst[i]) if i%2==1 else i*2 for i in range(len(lst)
print('3: '+[str(lst[i]) if i%2==1 else i*2 for i in len(lst)])
```

```
1: [4, '3', 8, '5', 12]
2: [0, '3', 4, '5', 8]
```

```
---------------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
Cell In[18], line 3
      1 print('1: '+str([str(i) if i%2==1 else i*2 for i in ls
t]))
      2 print('2: '+str([str(lst[i]) if i%2==1 else i*2 for i in
range(len(lst))]))
----> 3 print('3: '+[str(lst[i]) if i%2==1 else i*2 for i in len
(lst)])

TypeError: 'int' object is not iterable
```

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 reassigned)!

- 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 [2]:
```
lst = list(range(2,7))
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]
reversed list: [6, 5, 4, 3, 2]
the 2nd to 4th element: [4, 5]
every third element from the 1st to 10th element: [3, 6]
```

Checkpoint

Assume the following code has been ran. What are the results of the following expressions?

In [19]:
```
lst = [(1,2), (3,'a'), ([4],5)]
```

In [ ]:
```
x = lst[2][0] # what is x?
```

In [ ]:
```
x+=[6] # what is x now?
```

In [ ]:
```
print(lst) # what is the output?
```

Checkpoint Solution

In [20]:
```
x = lst[2][0]
x
```

Out[20]:
```
[4]
```

In [21]:
```
x+=[6]
x
```

Out[21]:
```
[4, 6]
```

In [22]:
```
print(lst)
```

```
[(1, 2), (3, 'a'), ([4, 6], 5)]
```

https://pythontutor.com/visualize.html#code=lst%20%3D%20%5B%281,2%29,%20%283,%
frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false

Dictionaries

- Mutable storage of key, value pairs
- Can store any data type, multiple at a time
- Elements are acccessed via keys (dct[key])
- 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!**

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!

Checkpoint

Assume the following code has been ran. What are the results of the following expressions?

In [23]:
```
data = {'a':['b','c','d'], 2:[3,4,10,5], 3:{'a':['b','c','d']}}
```

In [ ]:
```
data['a'][-1]
max(data[2])
data[3]['a'][2]
```

Checkpoint Solution

In [27]:
```
data
```

Out[27]:
```
{'a': ['b', 'c', 'd'], 2: [3, 4, 5], 3: {'a': ['b', 'c', 'd']}}
```

In [24]:
```
data['a'][-1]
```

Out[24]:
```
'd'
```

In [25]:
```
max(data[2])
```

Out[25]:
```
5
```

In [26]:
```
data[3]['a'][2]
```

Out[26]:
```
'd'
```

Files

- storage for data (csv, txt, json, parquet, 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.

Checkpoint

What happens when I try to open a file in my current folder that doesn't exist yet in write mode?

Checkpoint Solution

Python will automatically write an empty file with the provided filename.

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).

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. Since this is 2 days before the midterm, try to complete all the questions :)

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

practice question solutions

Given the following function and subsequent statement, what is the result? Why?

In [48]:
```
def indexer(item, index, new_val):
    """
    Function to assign a new value to an iterable
    item at a specific index.
    """
    item[index] = new_val
    return
```

In [49]:
```
indexer('star wars', 2, '@')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent
call last)
Cell In[49], line 1
----> 1 indexer('star wars', 2, '@')

Cell In[48], line 6, in indexer(item, index, new_val)
      1 def indexer(item, index, new_val):
      2     """
      3     Function to assign a new value to an iterable
      4     item at a specific index.
      5     """
----> 6     item[index] = new_val
      7     return

TypeError: 'str' object does not support item assignment
```

In [33]:

```
def extract_text(sentences):
    """
    Write a function that processes sentences by creating
    a dictionary that tracks the index of the sentences
    in which the word shows up.

    >>> sentences = ["a quick brown fox jumps",
    "a brown dog jumps at the fox", "dog"]
    >>> extract_text(sentences)
    {'a': [0, 1], 'quick': [0], 'brown': [0, 1], 'fox': [0, 1], \
    'jumps': [0, 1], 'dog': [1, 2], 'at': [1], 'the': [1]}
    """
    assert isinstance(sentences, list)
    assert all([isinstance(x, str) for x in sentences])
    output = {}
    for idx in range(len(sentences)):
        for word in sentences[idx].split():
            if word in output:
                output[word].append(idx)
            else:
                output[word] = [idx]
    return output

sentences = ["a quick brown fox jumps",
    "a brown dog jumps at the fox", "dog"]
print(extract_text(sentences))
```

```
{'a': [0, 1], 'quick': [0], 'brown': [0, 1], 'fox': [0, 1], 'jum
ps': [0, 1], 'dog': [1, 2], 'at': [1], 'the': [1]}
```

In [45]:

```
def string_comparer(lst, comparer):
    """
    Write a function that counts the number of equivalent
    strings in a list.
    Requirement: 1 line list comp

    >>> test(['good', 'luck', 'on', 'mt', 'luck'], 'luck')
    2
    >>> test([], 'lol')
    0
    """
    assert isinstance(lst, list)
    assert isinstance(comparer, str)
    assert all([isinstance(x,str) for x in lst])

    return len([item for item in lst if item==comparer])
print(string_comparer(['good','luck','on','mt','luck','luck'],'luck'))
print(string_comparer([],'lol'))
```

```
3
0
```

In [64]:

```
def process_file(file_path, sub):
    """
    Write a function that processes a file by replacing
    each instance of a specified string with another
    and returns the number of substitutes that occur
    as well as the corrected file content.

    >>> process_file('files/review.txt', ('hard', 'easy'))
    ("DSC20 is so easy. It's probably the easyest class \
    I've taken! I have so many easy classes this quarter. ",\
    3)
    """
    output = ''
    counter = 0
    with open(file_path, 'r') as fr:
        data = fr.read()
        for word in data.split():
            if sub[0] in word:
                counter+=1
                word = word.replace(sub[0], sub[1])
            output += word + ' '
    return output.strip(), counter

process_file('files/review.txt', ('hard', 'easy'))
```

Out[64]:

```
("DSC20 is so easy. It's probably the easyest class I've taken!
I have so many easy classes this quarter.",
 3)
```

Discussion Attendance

Take 2 minutes and head to gradescope to complete discussion attendance. The assignment is called Discussion 4 Participation.

Thanks for coming!