Discussion 1

DSC 20, Fall 2023

Welcome to DSC 20! 🎉

Agenda

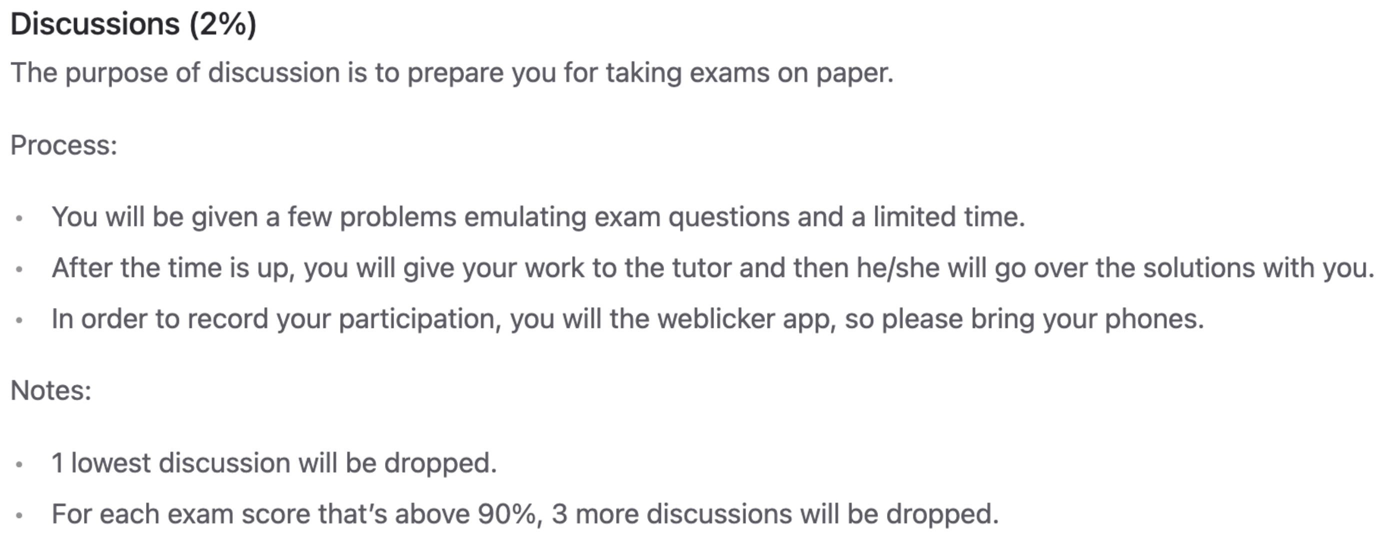
- How discussion will work this quarter
- Why bother learning how to code?
- **Content**
    - basic operators
    - basic data types
    - booleans
    - return keyword (short-circuit)
    - if conditions
    - for/while loop
    - mutability
    - lists, tuples, sets
- File Structure
- Style
- Doctests
- How coding works in this class (demo)
- Discussion Questions
- Debugging (demo)
- Class Advice (time permitting)

About Me

Benjamin (call me Ben)

- Originally from the Bay Area (San Jose/Fremont)
- Fourth year Data Science major
- 6th? 7th? quarter teaching DSC 20
- Outside the classroom 🧑‍🏫: lots of activities (track & field, cross country, badminton, marching band), reading, video games, anime, cooking, eating, travelling, mint addict, bracelet making, skateboarding, 3d renders, video/photo editing, etc.

How discussion will work this quarter - participation / attendance



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, you will give your work to the tutor and then he/she will go over the solutions with you.
- In order to record your participation, you will the weblicker app, so please bring your phones.

Notes:

- 1 lowest discussion will be dropped.
- For each exam score that's above 90%, 3 more discussions will be dropped.

- after reinforcing content, discussions will feature 4 - 5 practice questions.
- these questions will be on physical worksheets for exam prep (and good practice)
- attendance is taken using webclicker; they will be feedback regarding the practice questions
- webclicker is geo-locked; you have to be in/near the room to get points
- each exam > 90% correlates to 3 skipped discussions

Why bother learning how to code?

- "gpt-x can do everything"
- "*insert amazing futuristic model here* will replace our job"
- "*insert amazing futuristic model here* will cure cancer"

Scenario

Overpopulation is a looming issue that lots of people are worried about. Clearly, people are not coming up with good enough solutions, so why don't we just ask chatGPT or some other machine learning algorithm and use its solution? What could possibly go wrong?

Scenario Solution

A pretty realistic machine output to this problem would be the same solution Thanos came to in the MCU - simply remove half of all life and overpopulation is solved. Does this solution work? Sure, it does! Is it one that we would accept? No!! (I hope...)

Problem Statement

Although machine learning only gets better, at the end of the day, it is still a machine. It fails to capture all the qualities that go into complex scenarios. Yes, they are smart and honestly terrifying sometimes, but they are just **tools** at the end of the day.

Without a well versed user, tools are nothing more than scrap. This is where programmers come in. Until the day that machines are truly intelligent, we as programmers are the one who can use them effectively.

Programming is simply the language that translates human intention to computer actions. It instructs machines on how to carry out your logic.

So why bother learning how to code? So we can use cool tools like chatGPT or DALLE-2 to do amazing things!

Content Review

Jargon:

- function -> def function(x,y): # with code
- expression -> line(s) of code
- arguments -> what's passed into a function (function(**x,y**))
- evaluate -> code gets ran
- operators -> symbols/words that have unique, built-in functions

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)

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)

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. "else" is the final part of a chained conditional statement.

    //Do other other stuff

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
```

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**

```
In [4]: test_str = 'DSC20'
        test_str[-2] = '3'
        test_str
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 2
      1 test_str = 'DSC20'
----> 2 test_str[-2] = '3'
      3 test_str

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

```
In [5]: test_lst = ['D','S','C','2','0']
        test_lst[-2] = '3'
        test_lst

Out[5]: ['D', 'S', 'C', '3', '0']
```

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

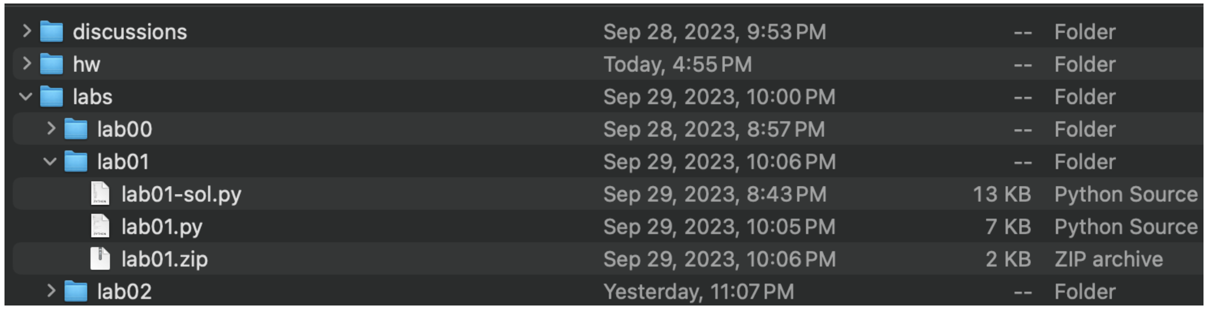
Set

- Mutable vector of values
- Only stores unique elements (removes duplicates)
- otherwise, list like behavior

In [6]:
```
to_5_list = [1,2,3,4,5]
to_5_tup = (1,2,3,4,5)
to_5_set = {1,2,3,4,5}
```

File Structure

- Staying organized is crucial to your success in both this class and later on
- Use a file structure that makes sense (we subtly enforce this)
- Enables tutors to help you faster, makes it easier for you to track things
- sample file structure:



discussions	Sep 28, 2023, 9:53 PM	--	Folder
hw	Today, 4:55 PM	--	Folder
labs	Sep 29, 2023, 10:00 PM	--	Folder
lab00	Sep 28, 2023, 8:57 PM	--	Folder
lab01	Sep 29, 2023, 10:06 PM	--	Folder
lab01-sol.py	Sep 29, 2023, 8:43 PM	13 KB	Python Source
lab01.py	Sep 29, 2023, 10:05 PM	7 KB	Python Source
lab01.zip	Sep 29, 2023, 10:06 PM	2 KB	ZIP archive
lab02	Yesterday, 11:07 PM	--	Folder

Style

Why enforce style? - having poor style while coding is a death sentence if you run into a bug. Having clean code makes identifying break points easy and smooths out your workflow if you don't finish in one sitting.

1. limit lines to 79 characters or less (use the ruler feature in your IDE)
2. avoid magic numbers (using a number randomly instead of identifying it) except for -1,0,1
3. avoid meaningless variable names (ex. ball, one=1)
4. add docstrings to every function (inner, helper) you write!
5. use snake_case for names

There are more rules, this is just a (very) brief overview. The entire ruleset can be found on the course website.

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.

How coding works in this class (demo)

practice questions

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

Make sure to work on this physically! This is practice for your own sake.

Practice Feedback

Go to webclicker: https://webclicker.web.app or open on your phone

How long have you been coding?

A. Less than 1 year

B. 1-2 years

C. 3-4 years

D. 4+ years

How many of the questions could you complete?

A. 0

B. 1

C. 2

D. 3

E. 4

Did you find the questions helpful?

A. Yes

B. Kind of

C. No

practice question solutions

In [11]:
```python
def power(x, a):
    """
    Write a function that calculates x^a.
    Assume both x and a are positive numbers.

    >>> power(2, 4)
    16
    >>> power(9,2)
    81
    """
    output = 1
    while a > 0:
        output *= x
        a-=1
    return output
print(power(2, 4))
print(power(9,2))
```

```
16
81
```

In [4]:
```python
# for loop solution
def power(x,a):
    output = 1
    for _ in range(a): # can use i instead of _
        output *= x
    return output
```

In [18]:

```python
def list_sorter(master_list):
    """
    Write a function that sorts numbers from
    master_list into a list of even numbers
    and a list of odd numbers.

    >>> list_sorter([1,2,3,4,5,6,7,8])
    ([1,3,5,7],[2,4,6,8])
    >>> list_sorter([1,3,5,7])
    ([1,3,5,7], [])
    """
    odd_list = []
    even_list = []
    for num in master_list:
        if num%2 == 1:
            odd_list+=[num]
        else:
            even_list+=[num]
    return (odd_list, even_list)

print(list_sorter([1,2,3,4,5,6,7,8]))
print(list_sorter([1,3,5,7]))
```

```
([1, 3, 5, 7], [2, 4, 6, 8])
([1, 3, 5, 7], [])
```

```
In [7]: def checker(string_lst, comparer):
            '''
            Write a function that counts the number of strings in a list
            that are equivalent to comparer.

            >>> checker(['she','sells','sea','shells','at','the','sea','shore']
            2
            >>> checker([], 'blank')
            0
            '''
            counter = 0
            for word in string_lst:
                if word == comparer:
                    counter+=1
            return counter
        print(checker(['she','sells','sea','shells','at','the','sea','shore'],
        print(checker([], 'blank'))
```

```
2
0
```

```
In [2]: def strictly_increasing(lst):
            """
            Write a function that takes in a list of integers and
            returns a new list containing only the numbers that are
            in increasing order. Numbers should appear in the same
            order in the input and output list.

            >>> increasing([1, 3, 2, 4, 5, 8, 7, 6, 9])
            [1, 3, 4, 5, 8, 9]
            """
            output = []
            for num in lst:
                if len(output) == 0:
                    output.append(num)
                elif num > output[-1]:
                    output.append(num)
            return output
        print(strictly_increasing([1, 3, 2, 4, 5, 8, 7, 6, 9]))
```

```
[1, 3, 4, 5, 8, 9]
```

PythonTutor - Debugging (demo)

Python Tutor

In [ ]:
```
# Wrong code
def list_sorter(master_list):
    odd_list = []
    even_list = []
    for num in master_list:
        if num%2 == 1:
            odd_list+=num
        else:
            even_list+=num
    return (odd_list, even_list)

list_sorter([1,2,3,4,5,6,7,8])
```

Things I would do if I were a student taking DSC20 now

1. Utilize Resources
2. Use Online Content, don't depend on them
3. Useful Tools
4. Practice
5. Take a break!

1. Utilize Resources

- Office hours are great, not just for getting help on assignments or studying for the class, but as interpersonal resources. All the tutors are amazing and they're open to just talking as peers - they're people you can ask about future classes, career advice, etc. Some tutors will have more input than others, but don't be shy! We're still students and we were/are in your shoes.

- Discussions are intended to be particuarly helpful. Rather than just reinforcing class concepts, they will also occassionally teach side topics that will help you become a strong programmer and help you adjust to exams (paper coding).

- If you're ever struggling, don't be afraid to reach out to the staff for help. We do our best to support students.

2. Use Online Content, don’t depend on them

- Tools like ChatGPT are amazing, but abusing them will only hurt your own learning
- Copying code from medium articles or w3school can get you a quick grade, but teaches you nothing and can get you AI'd
- Asking friends for a question can get not only you AI'd, but also your friend
- This class is best traversed with class content and staff support. You can consult your peers on programming ideas, but not the implementation itself

3. Useful Tools

1. Python Tutor - Debugging
    - invaluable for debugging, amazing for understanding why some code works, and why others don't
2. w3school, geeksforgeeks, other documentation and example websites - Reference
    - Stores documentation for what functions/methods do with examples
    - Always includes examples so that you can understand what's actually happening
3. Fig - File Navigation
    - terminal plugin for mac that autocompletes file names
    - can help you navigate your files much more easily
4. Iterm2
    - mac terminal alternative; has a lot of useful features
5. oh my zsh / power level 10k
    - framework to modify your terminal (enables plugins)
    - powerlevel10k is a ohmyzsh plugin to beautify terminal
    - makes things a lot more readable!(why my terminal looks nice^)

4. Practice

- like all new skills, practice makes perfect
- staring at code and not writing any will ultimately do nothing to make you a better coder
- Websites such as coding bat has practice problems for you to familiarize yourself with python
- codecademy is something that I personally used with DSC20 to help learn coding. It walks you through the fundamental functions of python and always frames it in sample code
- more challenging questions can be found on leetcode, though I don't recommend this for people new to coding.

more can be found on course website

5. Take a break!

- parking at the desk for 3+ hours and staring at a screen never helped anyone
- remember to take breaks while coding, sometimes your brain just comes up with the solution
- I personally have a lot of solutions come to me after a shower

Thanks for coming!