Exam Review!!!

Exam Review!

- Monday, April 7th @ 1:45pm in Towne 100
- Covers everything through **recursion**: NO PANDAS!
- Pencil & paper, so be ready…
- Practice exam on website:
    - two specifically for exam 2—very challenging
    - also the two previous practice exams, which are good for fundamentals and are still worthy of your time even if you did them a month ago.

Your Requests

- Help me understand jupyter notebook more
- How to count rows
- Help me understand pandas
- can we go over what the difference is between saying [schools[schools[""]]] vs schools[""]
- can we conceptually go over pandas and what they are on a higher level
- I'd like to see more examples on the difference between .iloc and .loc. I'm still a bit confused on what to use when.
- can you please help me understand how the panda works and its understanding throughout the course as its used a lot inside the course.
- how exactly does pandas work? and what does it due?

LATER!! Because `Pandas` is not on the exam!!!!

Your Other Requests: Higher Order Functions & Lambdas

(write your answers anywhere, there's just a ton of practice)

Write this function as a lambda:

```
def get_last(l: list[int]) -> int:
    return l[-1]
```

Write this lambda as a function:

```
lambda a, b: (3 * (a * a)) + (2 * a) + 1
```

Your Other Requests: Higher Order Functions & Lambdas

solutions

Write this function as a lambda:

```
def get_last(l: list[int]) -> int:
    return l[-1]
lambda l: l[-1]
```

Write this lambda as a function:

```
lambda a, c: (3 * (a * a)) + (2 * a) + c
def quadratic_func(a, c):
    return (3 * (a * a)) + (2 * a) + c
```

Your Other Requests: Higher Order Functions & Lambdas

Quick associations: pick the higher order function that seems most well-suited to the problem at hand.

- get a new copy of a list where we replace every negative number in a list with zero
- get a new copy of a list with only the positive numbers included
- count the number of positive integers in a list
- get a new copy of a list of strings that removes any strings starting with a `'+'`
- concatenate all of the strings in a list into one big string
- turn a list of tuples into a list of lists (where the contents of the lists are exactly the contents of the tuples)

- map
- filter
- reduce
- filter
- map
- reduce

Your Other Requests: Higher Order Functions & Lambdas

Do these two silently on your own! **WRITE IT OUT!!!!** Then, check with a partner.

- get a new copy of a list with only the positive numbers included
- concatenate all of the strings in a list into one big string

Your Other Requests: Recursion

Remember:

- Work towards a base case
    - Can you think of individual versions of the problem that are easier to solve?
    - If you started the problem with an iterative solution & accumulator variable, what would the initial value of that variable be
- Make the problem smaller
    - making a recursive call where one of the inputs literally gets smaller (smaller integer, smaller sublist of an input list)
    - making a recursive call where one of the inputs gets bigger, but therefore closer to the base case
- Figure out how to combine information from different recursive calls

Recursion: Through Iteration

```python
def count_even_values(l: list[int]) -> int:
    count = ?????
    for num in l:
        if num % 2 == 0:
            count += 1
    return count
```

What should `count` start at?

What would need to be true for you to return the initial value of `count`?

Recursion: Through Iteration

```python
def count_even_values(l: list[int]) -> int:
    count = ?????
    for num in l:
        if num % 2 == 0:
            count += 1
    return count
```

`count` starts at `0`, which is what we would return in the case of an empty list. In other words, `count` is the value returned in the base case, which happens when the list is empty

Recursion: Through Iteration

```python
def count_even_values(l: list[int]) -> int:
    if len(l) == 0:
        return 0

    first = l[0]
    rest = l[1:]
    if first % 2 == 0:
        return 1 + count_even_values(rest)
    else:
        return count_even_values(rest)
```

Recursion

Try this silently on your own! **WRITE IT OUT!!!!** Then, check with a partner.

```python
def take_only_positives(l: list[int]) -> list[int]:
    ...
```

(this will be equivalent to `filter(lambda e: e > 0, l)`)

Recursion

```
def take_only_positives(l: list[int]) -> list[int]:
    if len(l) == 0:
        return []

    first = l[0]
    rest = l[1:]
    if first > 0:
        return [first] + take_only_positives(rest)
    else:
        return take_only_positives(rest)
```

JSON

It's just lists and dictionaries that were saved to a file! Don't think of this as an especially separate unit, just think of dictionaries and lists (and then nesting these things inside of each other.)

JSON

```
{
  "data": [
    { "id": 1, "name": "Wei Zhang", "email": "wei@example.com", "status": "active"},
    { "id": 2, "name": "Aisha Patel", "email": "aisha@example.com", "status": "inactive"},
    { "id": 3, "name": "José Rodriguez", "email": "jose@example.com", "status": "pending"}
  ],
  "meta": {"total": 3}
}
```

If this JSON lives in a file called `people.json` and I write:

```
file = open("people.json", "r")
response = json.load(file)
```

What is the type of `response`? `response["data"]`? `response["meta"]`?

JSON

```
{
  "data": [
    { "id": 1, "name": "Wei Zhang", "email": "wei@example.com", "status": "active"},
    { "id": 2, "name": "Aisha Patel", "email": "aisha@example.com", "status": "inactive"},
    { "id": 3, "name": "José Rodriguez", "email": "jose@example.com", "status": "pending"}
  ],
  "meta": {"total": 3}
}
```

If this JSON lives in a file called `people.json` and I write:

```
file = open("people.json", "r")
response = json.load(file)
```

dict, list, dict

JSON

```
{
  "data": [
    { "id": 1, "name": "Wei Zhang", "email": "wei@example.com", "status": "active"},
    { "id": 2, "name": "Aisha Patel", "email": "aisha@example.com", "status": "inactive"},
    { "id": 3, "name": "José Rodriguez", "email": "jose@example.com", "status": "pending"}
  ],
  "meta": {"total": 3}
}
```

If this JSON lives in a file called `people.json` and I write:

```
file = open("people.json", "r")
response = json.load(file)
```

There are two simple expressions you can write to find the number of users whose data is included here. Write them both.

JSON

```
{
  "data": [
    { "id": 1, "name": "Wei Zhang", "email": "wei@example.com", "status": "active"},
    { "id": 2, "name": "Aisha Patel", "email": "aisha@example.com", "status": "inactive"},
    { "id": 3, "name": "José Rodriguez", "email": "jose@example.com", "status": "pending"}
  ],
  "meta": {"total": 3}
}
```

Write a short snippet to count the number of users whose status is pending:

```
file = open("people.json", "r")
response = json.load(file)
...
```

JSON

Count the number of users whose status is pending:

```python
file = open("people.json", "r")
response = json.load(file)
count = 0
for user in response["data"]:
    if user["status"] == "pending":
        count += 1
print(f"{count} pending users.")
```