Design Patterns for Handling Data 1

Welcome back to CS 2100!

Prof. Rasika Bhalerao

Starting exercise: let's write a function that executes a `Callable` multiple times

```python
def execute_multiple(func: Callable[[], None], times: int) -> None:
    """Executes the function func times number of times"""
    for _ in range(times):
        func()
```

Recall: Functions are objects

- can pass functions as arguments into other function
    - (so a variable can represent a function)
- variable type for a function that takes an argument of type `T` and returns type `R` is `Callable[[T], R]`

Result: Functions have attributes, just like other objects

```
from typing import Callable

def my_function() -> None:
    print('um hi')

print(my_function.__name__)
```

Exercise: let's write a function that returns the sum of two numbers (in one line)

```python
add: Callable[[int, int], int] = lambda x, y: x + y
```

Poll: Which of these variables is a function that safely divides `int`s or returns `None` if that's impossible?

1. `divide: Callable[[int, int], Optional[float]] = lambda num, den: num / den`
2. `divide: Callable[[int, int], float] = lambda num, den: num / den if den != 0 else None`
3. `divide: Callable[[int, int], Optional[float]] = lambda num, den: num / den if den != 0 else None`
4. `divide: Callable[[int, int], Optional[float]] = lambda num, den: try: num / den except ZeroDivisionError: None if den != 0 else None`

Why is it useful that functions are objects?

Sorting with Pandas

```
words: List[str] = 'never gonna give you up'.split()

sorted_alphabetically: List[str] = sorted(words)
print(' '.join(sorted_alphabetically)) # give gonna never up you

sorted_by_length: List[str] = sorted(words, key = lambda word: len(word))
print(' '.join(sorted_by_length))  # up you give gonna never
```

The `sorted()` function has an optional argument `key`, which is a function that is applied to each element when determining the sorted order.

Poll: Which of these lists is sorted by the number of vowels?

1. `sorted(words, key=lambda word: 'aeiou' in word))`
2. `sorted(words, key=lambda word: len([i for i in word if i in 'aeiou']))`
3. `sorted(words, key=lambda word: len([i for i in word if word in 'aeiou']))`
4. `sorted(words, key=lambda word, i: len([i for i in word if word in 'aeiou']))`

Tip: `df.sort_values()` *also* has a `key` arg, but that arg's function accepts the entire column as an argument, and returns the entire transformed column.

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Frog'],
    'Age': [13, 10, 3]})

print(df.sort_values(by='Person'))
print(df.sort_values(by='Person', key=lambda col: [len(word) for word in col]))
```

```
     Person  Age
1       Cat   10
0  Elephant   13
2      Frog    3

     Person  Age
1       Cat   10
2      Frog    3
0  Elephant   13
```

Similar exercise, but filter instead of sort

Let's write a function that takes a dataframe, and returns a dataframe containing only the rows where the 'Person' column contains the specified character.

```python
def rows_with_Person_containing_char(df: pd.DataFrame, character: str) -> pd.DataFrame:
    return df[character in df['Person']]
```

Quick note: Recall that we can iterate through a dataframe like this:

```python
for index, row in df.iterrows():
    # use row as a dict where the column names are its keys
```

It's common to treat each row in a Pandas dataframe as a `dict`.

Filtering with Pandas

Select the rows where the age is more than 5:

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Frog'],
    'Age': [13, 10, 3]})

print(df[df['Age'] > 5])
```

```
     Person  Age
0  Elephant   13
1       Cat   10
```

Filtering with Pandas

- filtering function takes entire dataset as the arg
- returns a list of booleans that is the same length as the dataset
  - indexes marked `True` will be included

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Frog'],
    'Age': [13, 10, 3]})

print(df[[True, False, True]])
```

```
     Person  Age
0  Elephant   13
2      Frog    3
```

Filtering with Pandas

- filtering function takes entire dataset as the arg
- returns a list of booleans that is the same length as the dataset
  - indexes marked `True` will be included

```
df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat', 'Frog'],
    'Age': [13, 10, 3]})

print(df[lambda dataframe: ['a' in row['Person'] for _, row in dataframe.iterrows()]])
```

```
     Person  Age
0  Elephant   13
1       Cat   10
```

Poll: Let's say we have a dataset of books, and the columns are `Title` (`str`), `Author` (`str`), and `Year` (`int`). How can we get the authors of the books whose titles contain the letter 'e', sorted by year?

1. `df[lambda dataframe: ['e' in row['Title'] for _, row in dataframe.iterrows()]].sort_values(by='Year')['Author']`
2. `df[['e' in row['Title'] for _, row in df.iterrows()]].sort_values(by='Year')['Author']`
3. `df[['e' in row['Title']]].sort_values(by='Year')['Author']`
4. `df[['e' in 'Title']].sort_values(by='Year')['Author']`

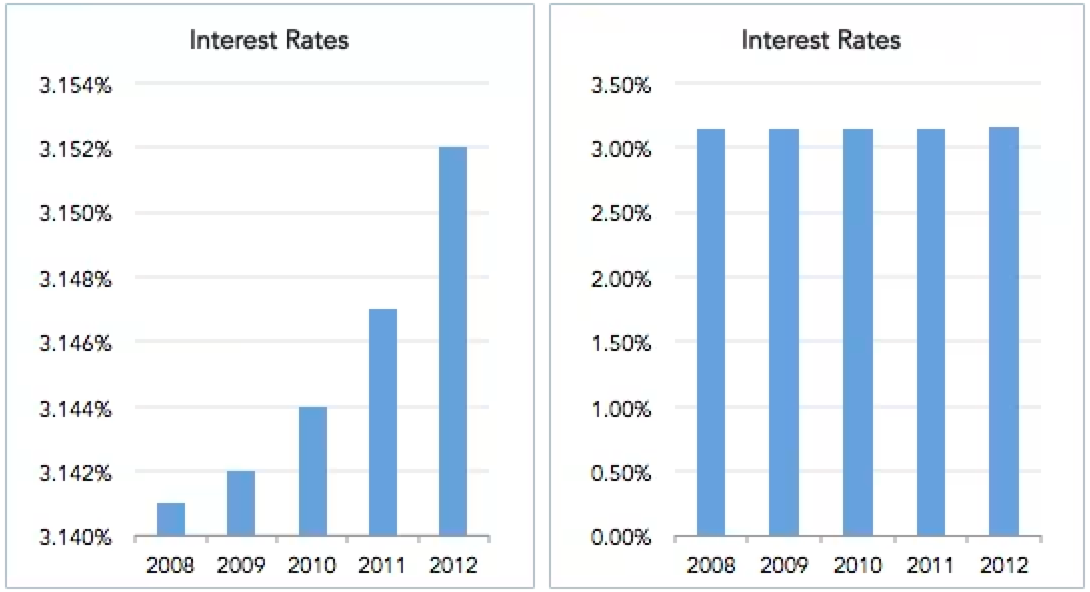
Let's plot cat heights and weights:

```python
import matplotlib.pyplot as plt

weight = [0, 50, 58, 51, 0, 47, 20, 19, 22, 0]
height = [0, 500, 480, 510, 2, 475, 200, 190, 220, 1]

plt.scatter(weight, height)
plt.xlim(0, 510)
plt.ylim(0, 510)

plt.title('Cat weight vs. height', fontsize=16, color='blue')
plt.xlabel('Cat name')
plt.ylabel('Height (millimeters)')

plt.show() # type: ignore
```

We love units, but they're messy

- Graph using inches versus centimeters looks different
- ML models give more "importance" to bigger numbers
- From https://www.heap.io/blog/how-to-lie-with-data-visualization:

Same Data, Different Y-Axis



Scaling and normalizing data

We often adjust the *scale* of a variable so it's in the same range as another variable.

Normalizing **data: scaling variables so that all of them are in the range between 0 and 1**

```
x_normalized = (x - x_min) / (x_max - x_min)
```

- `x_min` is the minimum data point for the variable `x`
- `x_max` is the maximum data point for the variable `x`
- Transform each value of `x` into `x_normalized` this way
- The set of `x_normalized` values is our new dataset comprised of numbers between 0 and 1

Poll: Let's say that this is our dataset of values for the variable `x`: [6, 8, 9, 8, 7]. What is the dataset after normalizing it?

1. [0.25, 0.75, 1, 0.75, 0.5]
2. [0, 0.67, 1, 0.67, 0.33]
3. [0, 0.5, 0.75, 0.5, 0.25]
4. [0, 4, 1, 3, 2]

Something we did: check in on students who didn't do well on Quiz 1, since it was before the deadline to switch to CS 2000

```python
students = np.array(['Mini', 'Meanie', 'Mega', 'Large', 'Tiny'])
q1_scores = np.array([90, 8, 96, 92, 85])

# Get indices that would sort Quiz 1 scores
priorities = np.argsort(q1_scores)

# Now we can see the ranking
print("Reach out to:")
for rank, idx in enumerate(priorities, 1):
    print(f"{rank}. {students[idx]}: {q1_scores[idx]}")
```

Argmax

Numpy's `argsort()` takes a list and returns a list of its indices, moved to the locations they would be in if the list was sorted.

```
words: List[str] = 'never gonna give you up'.split()

sorted_alphabetically: List[str] = sorted(words)

print(sorted_alphabetically)  # ['give', 'gonna', 'never', 'up', 'you']

indices_of_sorted_words = np.argsort(words)

print(indices_of_sorted_words)  # [2 1 0 4 3]
```

It's saying if the list of words was sorted alphabetically, the word at index 2 would be first, then the word at index 1, then the word at index 0, etc. until the word at index 3.

Poll:

1. What is your main takeaway from today?

2. What would you like to revisit next time?