1. Open Visual Studio Code
2. At the top click on View->Integrated Terminal (if not already open)
3. In the terminal, first run: `npm run pull`
4. After this finishes run: `npm start`

More Objects

Lecture 20

“Who Let The Dogs Out”
Announcements

• PS05 – Walk and Jog will post this afternoon
  • Due Sunday

• Sprint coming soon

• Quiz on Friday
What is a class?

• User defined data type

• Compound type that serves as a wrapper (box) around a set of simpler types

• Allows us to more naturally model the real world

```java
class TwitterProfile {
    name: string = "";
    handle: string = "";
    bio: string = "";
    showVine: boolean = false;
    privateAccount: boolean = false;
    followers: number = 0;
    following: number = 0;
}
Class vs Objects

• A class is a model for the data – think blueprint

• An object is an *instance* of a class
  • All Twitter profiles have the same structure (class) but there are many (millions) of Twitter profiles (objects)
    • Each profile has its own distinct values for those properties
Object Initialization

- let jeffrey: TwitterProfile = new TwitterProfile();
- New space is allocated in memory and the properties inside the object are initialized to their default values.
- We can then modify the properties of our object using the dot (.) operator.

```javascript
class TwitterProfile {
    name: string = "";
    handle: string = "";
    bio: string = "";
    showVine: boolean = false;
    privateAccount: boolean = false;
    followers: number = 0;
    following: number = 0;
}
```

<table>
<thead>
<tr>
<th></th>
<th>jeffrey</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>handle</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>bio</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>showVine</td>
<td>false</td>
</tr>
<tr>
<td>privateAccount</td>
<td>false</td>
</tr>
<tr>
<td>followers</td>
<td>0</td>
</tr>
<tr>
<td>following</td>
<td>0</td>
</tr>
</tbody>
</table>
Today's Goal:
Use objects and function calls to find a prospective dog owner their perfect match
Today's Data

• I asked the TAs to scroll through petfinder.com and enter some basic info about the pets into a spreadsheet.

• It was saved as a special type of file known as a CSV (*.csv).

• This is a common data table format that is easy to work with in code.
Here's what the contents of the CSV file look like.

It is stored in: lec20/pet-data.csv

Notice it's just "plain text"!

Each row gets a line, each column is separated by a comma, hence "Comma Separated Values (CSV)" file.
Modeling a Pet with a class

• Each Pet has properties associated with it:
  • name
  • gender
  • Distance from user
  • and more...

• These are column names in our data table

• In our program, we'll declare a class to model a single Pet with properties for each column in the table we care about.

  • Note: we do not need to use every column but the names of properties much match the column headers in the CSV file.
How do we read a CSV file as an array of Objects?

• There's a function in the introcs library for that! It's named csvToArray.

• Like prompts, you must await it.

• Documentation:

```ts
csvToArray(prompt: string, className: Class): Class[]
```
Follow-along: Calling `csvToArray`

- Open `lec20/00-tinder-for-dogs-app.ts`

- In the `main` function, call:

```typescript
let pets: Pet[] = await csvToArray("Pets", Pet);
```

- In `main`, print the length of the `pets` array.
Follow-Along Part 1.1: \texttt{filterLessThan20Miles}

- Write a function named \texttt{filterLessThan20Miles}
  - Parameters: \texttt{pets: Pet[]}
  - Return type: \texttt{Pet[]}

1. Create an empty Pet array named \texttt{lessThan20Miles} to hold result

2. Loop over the \texttt{pets} array and for each pet:
   1. If a pet is house trained, add it to the \texttt{lessThan20Miles} array
      - Remember, if we have a pet variable \texttt{p} we can access its \texttt{distance} property by saying \texttt{p.distance}

3. Return the \texttt{lessThan20Miles}
let filterLessThan20Miles = (pets: Pet[]): Pet[] => {
    let lessThan20Miles: Pet[] = [];
    for (let i: number = 0; i < pets.length; i++) {
        if (pets[i].distance < 20) {
            lessThan20Miles[lessThan20Miles.length] = pets[i];
        }
    }
    return lessThan20Miles;
};
What is happening when we call `filterLessThan20Miles`?
Hands-On Part 1.2: filterByHouseTrained

• Write a function named filterByHouseTrained
  • Parameters: pets: Pet[]
  • Return type: Pet[]

1. Create an empty Pet array named areHouseTrained to hold result
2. Loop over the pets array and for each pet:
   1. If a pet is house trained, add it to the areHouseTrained array
      • Remember, if we have a pet variable p we can access its houseTrained property by saying p.houseTrained (which is a boolean)
3. Return the areHouseTrainedArray

• Stuck? Check out the filterLessThan20Miles function for hints on implementation
let filterByHouseTrained = (pets: Pet[]): Pet[] => {
    let areHouseTrained: Pet[] = [];
    for (let i: number = 0; i < pets.length; i++) {
        if (pets[i].houseTrained === true) {
            areHouseTrained[areHouseTrained.length] = pets[i];
        }
    }
    return areHouseTrained;
};
Relevant Aside: `image`

- The `introcs` library has a function named `image`

- Usage:

```python
image(url: string)
```

- This will result in the image referenced by that URL being shown on the screen!
Hands-On Part 2: Swipe

• Next, we’ll write a loop allowing us to “swipe” left or right

1. Create an empty Pet array named swipedRight to hold the pets we like
2. Loop over each pet in the filtered array
3. Show the picture of the pet using the url property
   • This can be done with the built-in image function, ex. `image(pets[0].url);`
4. Prompt the user for a string where they will enter “left” or “right” (promptString) and store this in a string variable named choice
5. If the choice equals “right”, add it to the swipedRight array
let swipedRight: Pet[] = [];
for (let i: number = 0; i < filtered.length; i++) {
    image(filtered[i].url);
    let choice: string = await promptString("left or right");
    if (choice === "right") {
        swipedRight[swipedRight.length] = filtered[i];
    }
}
Hands-On Part 3: Print matches

- Write a function named `printMatches`
  - Parameters: `matches`: `Pet[]`
  - Return type: `string`

1. Loop over each pet in the `matches` array
2. Print out the name of the pet and their distance from you
   - This info is in the `name` and `distance` properties
3. Print out the image of the pet (using the `image` function)
4. If there was at least one element in the array (one match) return the string “success” else return “failure :(“
let printMatches = (pets: Pet[]): string => {
    for (let i: number = 0; i < pets.length; i++) {
        print(pets[i].name + " is " + pets[i].distance + " miles away");
        image(pets[i].url);
    }
    if (pets.length > 0) {
        return "success";
    } else {
        return "failure :("
    }
};
At Home: Write your own filters

• Practice writing your own filter functions at home. Some ideas include:
  • Less than 5 years old
  • Only females
  • Small or medium dogs
  • etc.